

Logik

Jan Hladik



1. Einführung

1.1 Organisation

1.2 Motivation

1.3 Geschichte

2. Mengen

3. AussagenLogik

4. PrädikatenLogik

5. Prolog

1. Einführung

1.1 Organisation

1.2 Motivation

1.3 Geschichte

2. Mengen

3. AussagenLogik

4. PrädikatenLogik

5. Prolog

1993–2001 Diplom Informatik, RWTH Aachen

- Diplomarbeit: Tableau-Algorithmus
- Nebenfach: Philosophie



1993–2001 Diplom Informatik, RWTH Aachen

- Diplomarbeit: Tableau-Algorithmus
- Nebenfach: Philosophie

2001–2007 Promotion, TU Dresden

- Dissertation: Tableaus vs. Automaten
- Leitung von Übungsgruppen



1993–2001 Diplom Informatik, RWTH Aachen

- Diplomarbeit: Tableau-Algorithmus
- Nebenfach: Philosophie

2001–2007 Promotion, TU Dresden

- Dissertation: Tableaus vs. Automaten
- Leitung von Übungsgruppen

2008–2014 Industrielerfahrung, SAP Research Dresden

- Öffentlich geförderte Forschungsprojekte
- Betreuung von Studenten und Doktoranden



1993–2001 Diplom Informatik, RWTH Aachen

- Diplomarbeit: Tableau-Algorithmus
- Nebenfach: Philosophie

2001–2007 Promotion, TU Dresden

- Dissertation: Tableaus vs. Automaten
- Leitung von Übungsgruppen

2008–2014 Industrielerfahrung, SAP Research Dresden

- Öffentlich geförderte Forschungsprojekte
- Betreuung von Studenten und Doktoranden

seit 2014 Professor, DHBW Stuttgart

- Logik
- Formale Sprachen und Automaten
- Semantic Web



1993–2001 Diplom Informatik, RWTH Aachen

- Diplomarbeit: Tableau-Algorithmus
- Nebenfach: Philosophie

2001–2007 Promotion, TU Dresden

- Dissertation: Tableaus vs. Automaten
- Leitung von Übungsgruppen

2008–2014 Industrielerfahrung, SAP Research Dresden

- Öffentlich geförderte Forschungsprojekte
- Betreuung von Studenten und Doktoranden

seit 2014 Professor, DHBW Stuttgart

- Logik
- Formale Sprachen und Automaten
- Semantic Web

Forschung Semantische Technologien

- Beschreibungslogik
- logische Schlussfolgerungsverfahren
- Semantic Web



- Präsentation, Übungsaufgaben

- Präsentation, Übungsaufgaben
 - <http://www.lehre.dhbw-stuttgart.de/~hladik/Logik/>

- Präsentation, Übungsaufgaben
 - <http://www.lehre.dhbw-stuttgart.de/~hladik/Logik/>
- Klausur

- Präsentation, Übungsaufgaben
 - <http://www.lehre.dhbw-stuttgart.de/~hladik/Logik/>
- Klausur
 - Dauer: 90 min

- Präsentation, Übungsaufgaben
 - <http://www.lehre.dhbw-stuttgart.de/~hladik/Logik/>
- Klausur
 - Dauer: 90 min
 - Hilfsmittel: Formelsammlung 10 Seiten DIN A4

- Präsentation, Übungsaufgaben
 - <http://www.lehre.dhbw-stuttgart.de/~hladik/Logik/>
- Klausur
 - Dauer: 90 min
 - Hilfsmittel: Formelsammlung 10 Seiten DIN A4
- Literatur

- Präsentation, Übungsaufgaben
 - <http://www.lehre.dhbw-stuttgart.de/~hladik/Logik/>
- Klausur
 - Dauer: 90 min
 - Hilfsmittel: Formelsammlung 10 Seiten DIN A4
- Literatur
 - Logik

- Präsentation, Übungsaufgaben
 - <http://www.lehre.dhbw-stuttgart.de/~hladik/Logik/>
- Klausur
 - Dauer: 90 min
 - Hilfsmittel: Formelsammlung 10 Seiten DIN A4
- Literatur
 - Logik
 - Dirk W. Hoffmann: [Theoretische Informatik](#)

- Präsentation, Übungsaufgaben
 - <http://www.lehre.dhbw-stuttgart.de/~hladik/Logik/>
- Klausur
 - Dauer: 90 min
 - Hilfsmittel: Formelsammlung 10 Seiten DIN A4
- Literatur
 - Logik
 - Dirk W. Hoffmann: [Theoretische Informatik](#)
 - Kurt-Ulrich Witt: [Mathematische Grundlagen für die Informatik](#)

- Präsentation, Übungsaufgaben
 - <http://wwwlehre.dhbw-stuttgart.de/~hladik/Logik/>
- Klausur
 - Dauer: 90 min
 - Hilfsmittel: Formelsammlung 10 Seiten DIN A4
- Literatur
 - Logik
 - Dirk W. Hoffmann: [Theoretische Informatik](#)
 - Kurt-Ulrich Witt: [Mathematische Grundlagen für die Informatik](#)
 - Karl Stroetmann: [Theoretische Informatik I - Logik und Mengenlehre](#)
<http://wwwlehre.dhbw-stuttgart.de/~stroetma/Logic/>

- Präsentation, Übungsaufgaben
 - <http://wwwlehre.dhbw-stuttgart.de/~hladik/Logik/>
- Klausur
 - Dauer: 90 min
 - Hilfsmittel: Formelsammlung 10 Seiten DIN A4
- Literatur
 - Logik
 - Dirk W. Hoffmann: [Theoretische Informatik](#)
 - Kurt-Ulrich Witt: [Mathematische Grundlagen für die Informatik](#)
 - Karl Stroetmann: [Theoretische Informatik I - Logik und Mengenlehre](#)
<http://wwwlehre.dhbw-stuttgart.de/~stroetma/Logic/>
 - Motivation

- Präsentation, Übungsaufgaben
 - <http://wwwlehre.dhbw-stuttgart.de/~hladik/Logik/>
- Klausur
 - Dauer: 90 min
 - Hilfsmittel: Formelsammlung 10 Seiten DIN A4
- Literatur
 - Logik
 - Dirk W. Hoffmann: [Theoretische Informatik](#)
 - Kurt-Ulrich Witt: [Mathematische Grundlagen für die Informatik](#)
 - Karl Stroetmann: [Theoretische Informatik I - Logik und Mengenlehre](#)
<http://wwwlehre.dhbw-stuttgart.de/~stroetma/Logic/>
 - Motivation
 - Apostolos Doxiadis, Christos Papadimitriou: [Logicomix – Eine epische Suche nach der Wahrheit](#)

- Präsentation, Übungsaufgaben
 - <http://wwwlehre.dhbw-stuttgart.de/~hladik/Logik/>
- Klausur
 - Dauer: 90 min
 - Hilfsmittel: Formelsammlung 10 Seiten DIN A4
- Literatur
 - Logik
 - Dirk W. Hoffmann: [Theoretische Informatik](#)
 - Kurt-Ulrich Witt: [Mathematische Grundlagen für die Informatik](#)
 - Karl Stroetmann: [Theoretische Informatik I - Logik und Mengenlehre](#)
<http://wwwlehre.dhbw-stuttgart.de/~stroetma/Logic/>
 - Motivation
 - Apostolos Doxiadis, Christos Papadimitriou: [Logicomix – Eine epische Suche nach der Wahrheit](#)
 - Douglas R. Hofstadter: [Gödel Escher Bach](#)

- Präsentation, Übungsaufgaben
 - <http://wwwlehre.dhbw-stuttgart.de/~hladik/Logik/>
- Klausur
 - Dauer: 90 min
 - Hilfsmittel: Formelsammlung 10 Seiten DIN A4
- Literatur
 - Logik
 - Dirk W. Hoffmann: [Theoretische Informatik](#)
 - Kurt-Ulrich Witt: [Mathematische Grundlagen für die Informatik](#)
 - Karl Stroetmann: [Theoretische Informatik I - Logik und Mengenlehre](#)
<http://wwwlehre.dhbw-stuttgart.de/~stroetma/Logic/>
 - Motivation
 - Apostolos Doxiadis, Christos Papadimitriou: [Logicomix – Eine epische Suche nach der Wahrheit](#)
 - Douglas R. Hofstadter: [Gödel Escher Bach](#)
 - Prolog

- Präsentation, Übungsaufgaben

- <http://wwwlehre.dhbw-stuttgart.de/~hladik/Logik/>

- Klausur

- Dauer: 90 min
 - Hilfsmittel: Formelsammlung 10 Seiten DIN A4

- Literatur

- Logik

- Dirk W. Hoffmann: [Theoretische Informatik](#)
 - Kurt-Ulrich Witt: [Mathematische Grundlagen für die Informatik](#)
 - Karl Stroetmann: [Theoretische Informatik I - Logik und Mengenlehre](#)
<http://wwwlehre.dhbw-stuttgart.de/~stroetma/Logic/>

- Motivation

- Apostolos Doxiadis, Christos Papadimitriou: [Logicomix – Eine epische Suche nach der Wahrheit](#)
 - Douglas R. Hofstadter: [Gödel Escher Bach](#)

- Prolog

- Patrick Blackburn et al.: [Learn Prolog Now!](#)
HTML-Version: <http://learnprolognow.org/>
mit Prolog-Interpreter: <http://lpn.swi-prolog.org/>

- Präsentation, Übungsaufgaben
 - <http://wwwlehre.dhbw-stuttgart.de/~hladik/Logik/>
- Klausur
 - Dauer: 90 min
 - Hilfsmittel: Formelsammlung 10 Seiten DIN A4
- Literatur
 - Logik
 - Dirk W. Hoffmann: [Theoretische Informatik](#)
 - Kurt-Ulrich Witt: [Mathematische Grundlagen für die Informatik](#)
 - Karl Stroetmann: [Theoretische Informatik I - Logik und Mengenlehre](#)
<http://wwwlehre.dhbw-stuttgart.de/~stroetma/Logic/>
 - Motivation
 - Apostolos Doxiadis, Christos Papadimitriou: [Logicomix – Eine epische Suche nach der Wahrheit](#)
 - Douglas R. Hofstadter: [Gödel Escher Bach](#)
 - Prolog
 - Patrick Blackburn et al.: [Learn Prolog Now!](#)
HTML-Version: <http://learnprolognow.org/>
mit Prolog-Interpreter: <http://lpn.swi-prolog.org/>
- Prolog-Interpreter

- Präsentation, Übungsaufgaben
 - <http://wwwlehre.dhbw-stuttgart.de/~hladik/Logik/>
- Klausur
 - Dauer: 90 min
 - Hilfsmittel: Formelsammlung 10 Seiten DIN A4
- Literatur
 - Logik
 - Dirk W. Hoffmann: [Theoretische Informatik](#)
 - Kurt-Ulrich Witt: [Mathematische Grundlagen für die Informatik](#)
 - Karl Stroetmann: [Theoretische Informatik I - Logik und Mengenlehre](#)
<http://wwwlehre.dhbw-stuttgart.de/~stroetma/Logic/>
 - Motivation
 - Apostolos Doxiadis, Christos Papadimitriou: [Logicomix – Eine epische Suche nach der Wahrheit](#)
 - Douglas R. Hofstadter: [Gödel Escher Bach](#)
 - Prolog
 - Patrick Blackburn et al.: [Learn Prolog Now!](#)
HTML-Version: <http://learnprolognow.org/>
mit Prolog-Interpreter: <http://lpn.swi-prolog.org/>
- Prolog-Interpreter
 - <http://swi-prolog.org/>

1. Einführung

1.1 Organisation

1.2 Motivation

1.3 Geschichte

1. Einführung

1.1 Organisation

1.2 Motivation

1.3 Geschichte

2. Mengen

2.1 Grundbegriffe

2.2 Mengenoperationen

1. Einführung

- 1.1 Organisation
- 1.2 Motivation
- 1.3 Geschichte

2. Mengen

- 2.1 Grundbegriffe
- 2.2 Mengenoperationen

3. Aussagenlogik

- 3.1 Syntax
- 3.2 Semantik
- 3.3 Schlussfolgerungsverfahren

1. Einführung

- 1.1 Organisation
- 1.2 Motivation
- 1.3 Geschichte

2. Mengen

- 2.1 Grundbegriffe
- 2.2 Mengenoperationen

3. Aussagenlogik

- 3.1 Syntax
- 3.2 Semantik
- 3.3 Schlussfolgerungsverfahren

4. Prädikatenlogik

- 4.1 Relationen und Funktionen

4.2 Syntax

4.3 Semantik

4.4 Schlussfolgerungsverfahren

1. Einführung

- 1.1 Organisation
- 1.2 Motivation
- 1.3 Geschichte

2. Mengen

- 2.1 Grundbegriffe
- 2.2 Mengenoperationen

3. Aussagenlogik

- 3.1 Syntax
- 3.2 Semantik
- 3.3 Schlussfolgerungsverfahren

4. Prädikatenlogik

- 4.1 Relationen und Funktionen

4.2 Syntax

4.3 Semantik

4.4 Schlussfolgerungsverfahren

5. Prolog

5.1 Terme

5.2 Fakten, Regeln und Anfragen

5.3 Unifikation, Termgleichheit und Arithmetik

5.4 Horn-Resolution und Suchbäume

5.5 Rekursion

5.6 Listen

5.7 Der Cut

1. Einführung

1.1 Organisation

1.2 Motivation

1.3 Geschichte

2. Mengen

3. AussagenLogik

4. PrädikatenLogik

5. Prolog



*Logic is the beginning of wisdom,
not the end.*

Mr. Spock, 2293 (Stardate 9522.6)

Warum Logik?



*Logic is the beginning of wisdom,
not the end.*

Mr. Spock, 2293 (Stardate 9522.6)

*Programming is a creative art form
based in logic.*

John Romero, 1993





*Logic is the beginning of wisdom,
not the end.*

Mr. Spock, 2293 (Stardate 9522.6)

*Programming is a creative art form
based in logic.*

John Romero, 1993



- kein Selbstzweck im Informatik-Studium



*Logic is the beginning of wisdom,
not the end.*

Mr. Spock, 2293 (Stardate 9522.6)

*Programming is a creative art form
based in logic.*

John Romero, 1993



- kein Selbstzweck im Informatik-Studium
- **Grundlage** für Programmierung, formale Sprachen, Mathematik, ...



*Logic is the beginning of wisdom,
not the end.*

Mr. Spock, 2293 (Stardate 9522.6)

*Programming is a creative art form
based in logic.*

John Romero, 1993



- kein Selbstzweck im Informatik-Studium
- **Grundlage** für Programmierung, formale Sprachen, Mathematik, ...
- notwendig für ein **fundiertes** Verständnis der Informatik

Ohne Theorie

- Trial and Error

Ohne Theorie

- Trial and Error
- „Neuerfindung des Rades“ (und Wiederholung alter Fehler)

Ohne Theorie

- Trial and Error
- „Neuerfindung des Rades“ (und Wiederholung alter Fehler)
- Verwendung „bekannter und bewährter“ Techniken

Ohne Theorie

- Trial and Error
- „Neuerfindung des Rades“ (und Wiederholung alter Fehler)
- Verwendung „bekannter und bewährter“ Techniken
- „Bauchgefühl“ für Lösungsansatz

Ohne Theorie

- Trial and Error
- „Neuerfindung des Rades“ (und Wiederholung alter Fehler)
- Verwendung „bekannter und bewährter“ Techniken
- „Bauchgefühl“ für Lösungsansatz

Ohne Theorie

- Trial and Error
- „Neuerfindung des Rades“ (und Wiederholung alter Fehler)
- Verwendung „bekannter und bewährter“ Techniken
- „Bauchgefühl“ für Lösungsansatz

Mit Theorie

- Abschätzung des Verhaltens eines Algorithmus **im Voraus**

Ohne Theorie

- Trial and Error
- „Neuerfindung des Rades“ (und Wiederholung alter Fehler)
- Verwendung „bekannter und bewährter“ Techniken
- „Bauchgefühl“ für Lösungsansatz

Mit Theorie

- Abschätzung des Verhaltens eines Algorithmus **im Voraus**
- Verständnis der **Gründe**, warum etwas funktioniert oder nicht

Ohne Theorie

- Trial and Error
- „Neuerfindung des Rades“ (und Wiederholung alter Fehler)
- Verwendung „bekannter und bewährter“ Techniken
- „Bauchgefühl“ für Lösungsansatz

Mit Theorie

- Abschätzung des Verhaltens eines Algorithmus **im Voraus**
- Verständnis der **Gründe**, warum etwas funktioniert oder nicht
- Bewusstsein der **Grenzen** von Computern und Algorithmen

Ohne Theorie

- Trial and Error
- „Neuerfindung des Rades“ (und Wiederholung alter Fehler)
- Verwendung „bekannter und bewährter“ Techniken
- „Bauchgefühl“ für Lösungsansatz

Mit Theorie

- Abschätzung des Verhaltens eines Algorithmus **im Voraus**
- Verständnis der **Gründe**, warum etwas funktioniert oder nicht
- Bewusstsein der **Grenzen** von Computern und Algorithmen

Ohne Theorie

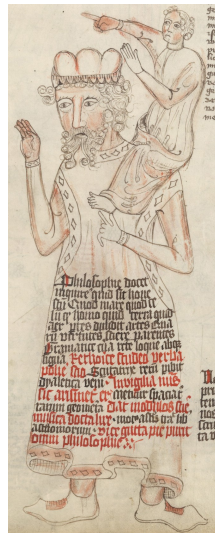
- Trial and Error
- „Neuerfindung des Rades“ (und Wiederholung alter Fehler)
- Verwendung „bekannter und bewährter“ Techniken
- „Bauchgefühl“ für Lösungsansatz

Mit Theorie

- Abschätzung des Verhaltens eines Algorithmus **im Voraus**
- Verständnis der **Gründe**, warum etwas funktioniert oder nicht
- Bewusstsein der **Grenzen** von Computern und Algorithmen

Wir sind gleichsam Zwerge, die auf den Schultern von Riesen sitzen, um mehr und Entfernteres als diese sehen zu können – freilich nicht dank eigener scharfer Sehkraft oder Körpergröße, sondern weil die Größe der Riesen uns emporhebt.

Bernhard von Chartres, ca. 1120



- Technische Grundlagen

- Technische Grundlagen
 - Boole'sche Schaltkreise ([Digitaltechnik](#))

- Technische Grundlagen
 - Boole'sche Schaltkreise ([Digitaltechnik](#))
- Anwendung innerhalb der Informatik

- Technische Grundlagen
 - Boole'sche Schaltkreise ([Digitaltechnik](#))
- Anwendung innerhalb der Informatik
 - [Programmierung](#)

- Technische Grundlagen
 - Boole'sche Schaltkreise ([Digitaltechnik](#))
- Anwendung innerhalb der Informatik
 - [Programmierung](#)
 - Syntax und Semantik von Programmiersprachen

- Technische Grundlagen
 - Boole'sche Schaltkreise ([Digitaltechnik](#))
- Anwendung innerhalb der Informatik
 - [Programmierung](#)
 - Syntax und Semantik von Programmiersprachen
 - [Datenbanken](#)

- Technische Grundlagen
 - Boole'sche Schaltkreise ([Digitaltechnik](#))
- Anwendung innerhalb der Informatik
 - [Programmierung](#)
 - Syntax und Semantik von Programmiersprachen
 - [Datenbanken](#)
 - Relationale Algebra

- Technische Grundlagen
 - Boole'sche Schaltkreise ([Digitaltechnik](#))
- Anwendung innerhalb der Informatik
 - [Programmierung](#)
 - Syntax und Semantik von Programmiersprachen
 - [Datenbanken](#)
 - Relationale Algebra
 - Spezifikation ([Software-Engineering](#))

- Technische Grundlagen
 - Boole'sche Schaltkreise ([Digitaltechnik](#))
- Anwendung innerhalb der Informatik
 - [Programmierung](#)
 - Syntax und Semantik von Programmiersprachen
 - [Datenbanken](#)
 - Relationale Algebra
 - Spezifikation ([Software-Engineering](#))
 - Programmverifikation

- Technische Grundlagen
 - Boole'sche Schaltkreise ([Digitaltechnik](#))
- Anwendung innerhalb der Informatik
 - [Programmierung](#)
 - Syntax und Semantik von Programmiersprachen
 - [Datenbanken](#)
 - Relationale Algebra
 - Spezifikation ([Software-Engineering](#))
 - Programmverifikation
 - „Erfüllt mein Programm die Spezifikation?“

- Technische Grundlagen
 - Boole'sche Schaltkreise ([Digitaltechnik](#))
- Anwendung innerhalb der Informatik
 - [Programmierung](#)
 - Syntax und Semantik von Programmiersprachen
 - [Datenbanken](#)
 - Relationale Algebra
 - Spezifikation ([Software-Engineering](#))
 - Programmverifikation
 - „Erfüllt mein Programm die Spezifikation?“
 - „Ermöglicht mein Kommunikationsprotokoll Deadlocks?“

- Technische Grundlagen
 - Boole'sche Schaltkreise ([Digitaltechnik](#))
- Anwendung innerhalb der Informatik
 - [Programmierung](#)
 - Syntax und Semantik von Programmiersprachen
 - [Datenbanken](#)
 - Relationale Algebra
 - Spezifikation ([Software-Engineering](#))
 - Programmverifikation
 - „Erfüllt mein Programm die Spezifikation?“
 - „Ermöglicht mein Kommunikationsprotokoll Deadlocks?“
 - Berechenbarkeit ([Formale Sprachen und Automaten](#))

- Technische Grundlagen
 - Boole'sche Schaltkreise ([Digitaltechnik](#))
- Anwendung innerhalb der Informatik
 - [Programmierung](#)
 - Syntax und Semantik von Programmiersprachen
 - [Datenbanken](#)
 - Relationale Algebra
 - Spezifikation ([Software-Engineering](#))
 - Programmverifikation
 - „Erfüllt mein Programm die Spezifikation?“
 - „Ermöglicht mein Kommunikationsprotokoll Deadlocks?“
 - Berechenbarkeit ([Formale Sprachen und Automaten](#))
 - Verständnis der Fähigkeiten und Grenzen von Computern

- Technische Grundlagen
 - Boole'sche Schaltkreise ([Digitaltechnik](#))
- Anwendung innerhalb der Informatik
 - [Programmierung](#)
 - Syntax und Semantik von Programmiersprachen
 - [Datenbanken](#)
 - Relationale Algebra
 - Spezifikation ([Software-Engineering](#))
 - Programmverifikation
 - „Erfüllt mein Programm die Spezifikation?“
 - „Ermöglicht mein Kommunikationsprotokoll Deadlocks?“
 - Berechenbarkeit ([Formale Sprachen und Automaten](#))
 - Verständnis der Fähigkeiten und Grenzen von Computern
 - „Schreiben Sie einen Test, ob zwei Programme sich gleich verhalten.“

- Technische Grundlagen
 - Boole'sche Schaltkreise ([Digitaltechnik](#))
- Anwendung innerhalb der Informatik
 - [Programmierung](#)
 - Syntax und Semantik von Programmiersprachen
 - [Datenbanken](#)
 - Relationale Algebra
 - Spezifikation ([Software-Engineering](#))
 - Programmverifikation
 - „Erfüllt mein Programm die Spezifikation?“
 - „Ermöglicht mein Kommunikationsprotokoll Deadlocks?“
 - Berechenbarkeit ([Formale Sprachen und Automaten](#))
 - Verständnis der Fähigkeiten und Grenzen von Computern
 - „Schreiben Sie einen Test, ob zwei Programme sich gleich verhalten.“
- Werkzeug für Anwendungen der Informatik außerhalb der Informatik

- Technische Grundlagen
 - Boole'sche Schaltkreise ([Digitaltechnik](#))
- Anwendung innerhalb der Informatik
 - [Programmierung](#)
 - Syntax und Semantik von Programmiersprachen
 - [Datenbanken](#)
 - Relationale Algebra
 - Spezifikation ([Software-Engineering](#))
 - Programmverifikation
 - „Erfüllt mein Programm die Spezifikation?“
 - „Ermöglicht mein Kommunikationsprotokoll Deadlocks?“
 - Berechenbarkeit ([Formale Sprachen und Automaten](#))
 - Verständnis der Fähigkeiten und Grenzen von Computern
 - „Schreiben Sie einen Test, ob zwei Programme sich gleich verhalten.“
- Werkzeug für Anwendungen der Informatik außerhalb der Informatik
 - Wissensrepräsentation ([Künstliche Intelligenz](#))

- Technische Grundlagen
 - Boole'sche Schaltkreise ([Digitaltechnik](#))
- Anwendung innerhalb der Informatik
 - [Programmierung](#)
 - Syntax und Semantik von Programmiersprachen
 - [Datenbanken](#)
 - Relationale Algebra
 - Spezifikation ([Software-Engineering](#))
 - Programmverifikation
 - „Erfüllt mein Programm die Spezifikation?“
 - „Ermöglicht mein Kommunikationsprotokoll Deadlocks?“
 - Berechenbarkeit ([Formale Sprachen und Automaten](#))
 - Verständnis der Fähigkeiten und Grenzen von Computern
 - „Schreiben Sie einen Test, ob zwei Programme sich gleich verhalten.“
- Werkzeug für Anwendungen der Informatik außerhalb der Informatik
 - Wissensrepräsentation ([Künstliche Intelligenz](#))
 - Automatisches Beweisen

- Begriffe

- Begriffe

- Syntax: Wann ist ein Ausdruck korrekt?

- Begriffe

- Syntax: Wann ist ein Ausdruck korrekt?
 - Korrektheit

- Begriffe

- Syntax: Wann ist ein Ausdruck korrekt?

- Korrektheit
 - Term, Formel

- Begriffe

- Syntax: Wann ist ein Ausdruck korrekt?

- Korrektheit
 - Term, Formel
 - Funktions-, Relationssymbole

■ Begriffe

- Syntax: Wann ist ein Ausdruck korrekt?
 - Korrektheit
 - Term, Formel
 - Funktions-, Relationssymbole
- Semantik: Was bedeutet ein korrekter Ausdruck?

■ Begriffe

- Syntax: Wann ist ein Ausdruck korrekt?
 - Korrektheit
 - Term, Formel
 - Funktions-, Relationssymbole
- Semantik: Was bedeutet ein korrekter Ausdruck?
 - Wahrheit

■ Begriffe

- Syntax: Wann ist ein Ausdruck korrekt?
 - Korrektheit
 - Term, Formel
 - Funktions-, Relationssymbole
- Semantik: Was bedeutet ein korrekter Ausdruck?
 - Wahrheit
 - Interpretation, Modell

■ Begriffe

- Syntax: Wann ist ein Ausdruck korrekt?
 - Korrektheit
 - Term, Formel
 - Funktions-, Relationssymbole
- Semantik: Was bedeutet ein korrekter Ausdruck?
 - Wahrheit
 - Interpretation, Modell
 - Gültigkeit, Erfüllbarkeit

■ Begriffe

- Syntax: Wann ist ein Ausdruck korrekt?
 - Korrektheit
 - Term, Formel
 - Funktions-, Relationssymbole
- Semantik: Was bedeutet ein korrekter Ausdruck?
 - Wahrheit
 - Interpretation, Modell
 - Gültigkeit, Erfüllbarkeit
 - Folgerung

■ Begriffe

- Syntax: Wann ist ein Ausdruck korrekt?
 - Korrektheit
 - Term, Formel
 - Funktions-, Relationssymbole
- Semantik: Was bedeutet ein korrekter Ausdruck?
 - Wahrheit
 - Interpretation, Modell
 - Gültigkeit, Erfüllbarkeit
 - Folgerung

■ Beweisverfahren

■ Begriffe

- Syntax: Wann ist ein Ausdruck korrekt?
 - Korrektheit
 - Term, Formel
 - Funktions-, Relationssymbole
- Semantik: Was bedeutet ein korrekter Ausdruck?
 - Wahrheit
 - Interpretation, Modell
 - Gültigkeit, Erfüllbarkeit
 - Folgerung

■ Beweisverfahren

- Resolution

■ Begriffe

- Syntax: Wann ist ein Ausdruck korrekt?
 - Korrektheit
 - Term, Formel
 - Funktions-, Relationssymbole
- Semantik: Was bedeutet ein korrekter Ausdruck?
 - Wahrheit
 - Interpretation, Modell
 - Gültigkeit, Erfüllbarkeit
 - Folgerung

■ Beweisverfahren

- Resolution
- Tableaus

■ Begriffe

- Syntax: Wann ist ein Ausdruck korrekt?
 - Korrektheit
 - Term, Formel
 - Funktions-, Relationssymbole
- Semantik: Was bedeutet ein korrekter Ausdruck?
 - Wahrheit
 - Interpretation, Modell
 - Gültigkeit, Erfüllbarkeit
 - Folgerung

■ Beweisverfahren

- Resolution
- Tableaus

■ Praktische Anwendung von Logik

■ Begriffe

- Syntax: Wann ist ein Ausdruck korrekt?
 - Korrektheit
 - Term, Formel
 - Funktions-, Relationssymbole
- Semantik: Was bedeutet ein korrekter Ausdruck?
 - Wahrheit
 - Interpretation, Modell
 - Gültigkeit, Erfüllbarkeit
 - Folgerung

■ Beweisverfahren

- Resolution
- Tableaus

■ Praktische Anwendung von Logik

- Formalisierung in Aussagen- und Prädikatenlogik

■ Begriffe

- Syntax: Wann ist ein Ausdruck korrekt?
 - Korrektheit
 - Term, Formel
 - Funktions-, Relationssymbole
- Semantik: Was bedeutet ein korrekter Ausdruck?
 - Wahrheit
 - Interpretation, Modell
 - Gültigkeit, Erfüllbarkeit
 - Folgerung

■ Beweisverfahren

- Resolution
- Tableaus

■ Praktische Anwendung von Logik

- Formalisierung in Aussagen- und Prädikatenlogik
- Prolog

1. Einführung

1.1 Organisation

1.2 Motivation

1.3 Geschichte

2. Mengen

3. Aussagenlogik

4. Prädikatenlogik

5. Prolog

Logos
λόγος

Logos

λόγος

- verwandt mit λέγειν (sprechen)

Logos

λόγος

- verwandt mit λέγειν (sprechen)
- Wort, Satz, Rede

Logos

λόγος

- verwandt mit λέγειν (sprechen)
- Wort, Satz, Rede
- Argumentation

Logos

λόγος

- verwandt mit λέγειν (sprechen)
- Wort, Satz, Rede
- Argumentation
- Beweis, Definition

Logos

λόγος

- verwandt mit λέγειν (sprechen)
- Wort, Satz, Rede
- Argumentation
- Beweis, Definition
- Vernunft, Rationalität

Logos

λόγος

- verwandt mit λέγειν (sprechen)
- Wort, Satz, Rede
- Argumentation
- Beweis, Definition
- Vernunft, Rationalität

Logos

λόγος

- verwandt mit λέγειν (sprechen)
- Wort, Satz, Rede
- Argumentation
- Beweis, Definition
- Vernunft, Rationalität

Gegenbegriff: Mythos

μῦθος

Logos

λόγος

- verwandt mit λέγειν (sprechen)
- Wort, Satz, Rede
- Argumentation
- Beweis, Definition
- Vernunft, Rationalität

Gegenbegriff: Mythos

μῦθος

- Wort, Rede, Erzählung

Logos

λόγος

- verwandt mit λέγειν (sprechen)
- Wort, Satz, Rede
- Argumentation
- Beweis, Definition
- Vernunft, Rationalität

Gegenbegriff: Mythos

μῦθος

- Wort, Rede, Erzählung
- Legende

Logos

λόγος

- verwandt mit λέγειν (sprechen)
- Wort, Satz, Rede
- Argumentation
- Beweis, Definition
- Vernunft, Rationalität

Gegenbegriff: Mythos

μῦθος

- Wort, Rede, Erzählung
- Legende
- Vermischung von Fakten und Glauben

Logos

λόγος

- verwandt mit λέγειν (sprechen)
- Wort, Satz, Rede
- Argumentation
- Beweis, Definition
- Vernunft, Rationalität

Gegenbegriff: Mythos

μῦθος

- Wort, Rede, Erzählung
- Legende
- Vermischung von Fakten und Glauben

Logos

λόγος

- verwandt mit λέγειν (sprechen)
- Wort, Satz, Rede
- Argumentation
- Beweis, Definition
- Vernunft, Rationalität

Gegenbegriff: Mythos

μῦθος

- Wort, Rede, Erzählung
- Legende
- Vermischung von Fakten und Glauben

Zentraler Begriff für Ursprung des abendländischen Denkens

- Übergang „[vom Mythos zum Logos](#)“ (Wilhelm Nestle, 1940)
- Wendung von Legenden zur Rationalität

Ἐν ἀρχῇ ἦν ὁ λόγος,
καὶ ὁ λόγος ἦν πρὸς τὸν θεόν,
καὶ θεὸς ἦν ὁ λόγος.

...
Καὶ ὁ λόγος σὰρξ ἐγένετο
καὶ ἐσκήνωσεν ἐν ἡμῖν, ...

*Im Anfang war das Wort
und das Wort war bei Gott
und Gott war das Wort.*

*...
Und das Wort ist Fleisch geworden
und hat unter uns gewohnt, ...*

Joh 1 1,14

Ἐν ἀρχῇ ἦν ὁ λόγος,
καὶ ὁ λόγος ἦν πρὸς τὸν θεόν,
καὶ θεὸς ἦν ὁ λόγος.

...
Καὶ ὁ λόγος σὰρξ ἐγένετο
καὶ ἐσκήνωσεν ἐν ἡμῖν, ...

*Im Anfang war das Wort
und das Wort war bei Gott
und Gott war das Wort.*

*...
Und das Wort ist Fleisch geworden
und hat unter uns gewohnt, ...*

Joh 1 1,14

Heutige Verwendung:

- Logik

Ἐν ἀρχῇ ἦν ὁ λόγος,
καὶ ὁ λόγος ἦν πρὸς τὸν θεόν,
καὶ θεὸς ἦν ὁ λόγος.

...
Καὶ ὁ λόγος σὰρξ ἐγένετο
καὶ ἐσκήνωσεν ἐν ἡμῖν, ...

*Im Anfang war das Wort
und das Wort war bei Gott
und Gott war das Wort.*

*...
Und das Wort ist Fleisch geworden
und hat unter uns gewohnt, ...*

Joh 1 1,14

Heutige Verwendung:

- Logik
- Wissenschaften enden auf -logie

Ἐν ἀρχῇ ἦν ὁ λόγος,
καὶ ὁ λόγος ἦν πρὸς τὸν θεόν,
καὶ θεὸς ἦν ὁ λόγος.

...
Καὶ ὁ λόγος σὰρξ ἐγένετο
καὶ ἐσκήνωσεν ἐν ἡμῖν, ...

*Im Anfang war das Wort
und das Wort war bei Gott
und Gott war das Wort.*

*...
Und das Wort ist Fleisch geworden
und hat unter uns gewohnt, ...*

Joh 1 1,14

Heutige Verwendung:

- Logik
- Wissenschaften enden auf -logie
- Logopädie (Sprecherziehung)

Ἐν ἀρχῇ ἦν ὁ λόγος,
καὶ ὁ λόγος ἦν πρὸς τὸν θεόν,
καὶ θεὸς ἦν ὁ λόγος.

...
Καὶ ὁ λόγος σὰρξ ἐγένετο
καὶ ἐσκήνωσεν ἐν ἡμῖν, ...

*Im Anfang war das Wort
und das Wort war bei Gott
und Gott war das Wort.*

*...
Und das Wort ist Fleisch geworden
und hat unter uns gewohnt, ...*

Joh 1 1,14

Heutige Verwendung:

- Logik
- Wissenschaften enden auf -logie
- Logopädie (Sprecherziehung)

Ἐν ἀρχῇ ἦν ὁ λόγος,
καὶ ὁ λόγος ἦν πρὸς τὸν θεόν,
καὶ θεὸς ἦν ὁ λόγος.

...
Καὶ ὁ λόγος σὰρξ ἐγένετο
καὶ ἐσκήνωσεν ἐν ἡμῖν, ...

*Im Anfang war das Wort
und das Wort war bei Gott
und Gott war das Wort.*

*...
Und das Wort ist Fleisch geworden
und hat unter uns gewohnt, ...*

Joh 1 1,14

Heutige Verwendung:

- Logik
- Wissenschaften enden auf -logie
- Logopädie (Sprecherziehung)

- Dialog („durch Wörter“)

Ἐν ἀρχῇ ἦν ὁ λόγος,
καὶ ὁ λόγος ἦν πρὸς τὸν θεόν,
καὶ θεὸς ἦν ὁ λόγος.

...
Καὶ ὁ λόγος σὰρξ ἐγένετο
καὶ ἐσκήνωσεν ἐν ἡμῖν, ...

*Im Anfang war das Wort
und das Wort war bei Gott
und Gott war das Wort.*

*...
Und das Wort ist Fleisch geworden
und hat unter uns gewohnt, ...*

Joh 1 1,14

Heutige Verwendung:

- Logik
- Wissenschaften enden auf -logie
- Logopädie (Sprecherziehung)

- Dialog („durch Wörter“)
- Trilogie („drei Werke“)

Ἐν ἀρχῇ ἦν ὁ λόγος,
καὶ ὁ λόγος ἦν πρὸς τὸν θεόν,
καὶ θεὸς ἦν ὁ λόγος.

...
Καὶ ὁ λόγος σὰρξ ἐγένετο
καὶ ἐσκήνωσεν ἐν ἡμῖν, ...

*Im Anfang war das Wort
und das Wort war bei Gott
und Gott war das Wort.*

*...
Und das Wort ist Fleisch geworden
und hat unter uns gewohnt, ...*

Joh 1 1,14

Heutige Verwendung:

- Logik
- Wissenschaften enden auf -logie
- Logopädie (Sprecherziehung)
- Dialog („durch Wörter“)
- Trilogie („drei Werke“)
- Dekalog (Zehn Gebote)

- Aristoteles: „Metaphysik“



Aristoteles
384–322 v.C.

- Aristoteles: „Metaphysik“
 - Satz vom ausgeschlossenen Widerspruch

Es ist unmöglich, dass dasselbe demselben in derselben Beziehung zugleich zukomme und nicht zukomme.



Aristoteles
384–322 v.C.

- Aristoteles: „Metaphysik“

- Satz vom ausgeschlossenen Widerspruch

Es ist unmöglich, dass dasselbe demselben in derselben Beziehung zugleich zukomme und nicht zukomme.

- Satz vom ausgeschlossenen Dritten

Es ist unmöglich, dass es ein Mittleres zwischen den beiden Gliedern des Widerspruchs gibt, sondern man muss eben eines von beiden entweder bejahen oder verneinen.



Aristoteles
384–322 v.C.

- Gottfried Wilhelm Leibniz



G.W. Leibniz
1646–1716

- Gottfried Wilhelm Leibniz
 - Dualsystem: 0 $\rightsquigarrow$ Nichts; 1 $\rightsquigarrow$ Gott

*Um alles aus dem Nichts herzuleiten,
genügt Eines.*

10^e. Tabulag ita stabil

1	2	2 ⁰
10	4	2 ¹
100	8	2 ²
1000	16	2 ³
10000	32	2 ⁴
100000	64	2 ⁵
1000000	128	2 ⁶
10000000	256	2 ⁷
100000000	512	2 ⁸
1000000000	1024	2 ⁹
10000000000		2 ¹⁰



G.W. Leibniz
1646–1716

■ Gottfried Wilhelm Leibniz

- Dualsystem: 0 $\rightsquigarrow$ Nichts; 1 $\rightsquigarrow$ Gott

*Um alles aus dem Nichts herzuleiten,
genügt Eines.*

- Rechenmaschinen

*Denn es ist ausgezeichnete Menschen unwürdig,
gleich Sklaven Stunden zu verlieren mit Berechnungen.*

Handwritten manuscript titled "Tabulae" showing powers of 2 and powers of 10. The left column lists powers of 10 (1, 10, 100, 1000, 10000, 100000, 1000000, 10000000, 100000000, 1000000000) and the right column lists powers of 2 (1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024). The manuscript is written in Latin and includes the phrase "ita stabili" (thus stable).

Powers of 10	Powers of 2
1	1
10	2
100	4
1000	8
10000	16
100000	32
1000000	64
10000000	128
100000000	256
1000000000	512
	1024



G.W. Leibniz
1646–1716

■ Gottfried Wilhelm Leibniz

- Dualsystem: $0 \rightsquigarrow$ Nichts; $1 \rightsquigarrow$ Gott

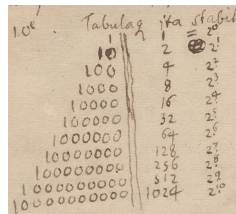
*Um alles aus dem Nichts herzuleiten,
genügt Eines.*

- Rechenmaschinen

Denn es ist ausgezeichnete Menschen unwürdig, gleich Sklaven Stunden zu verlieren mit Berechnungen.

- Idee der Formalisierung von Ausdrücken und Beweisen

*Philosophen werden nicht anders argumentieren als Rechenmeister. Sie werden die Feder in die Hand nehmen und sagen:
Calculemus!*



G.W. Leibniz
1646–1716

- George Boole:
„The Mathematical Analysis of Logic“

1st. Disjunctive Syllogism.	
Either X is true, or Y is true (exclusive),	$x + y - 2xy = 1$
But X is true,	$x = 1$
Therefore Y is not true,	$\therefore y = 0$
Either X is true, or Y is true (not exclusive),	
But X is not true,	$x + y - xy = 1$
Therefore Y is true,	$x = 0$
	$\therefore y = 1$
2nd. Constructive Conditional Syllogism.	
If X is true, Y is true,	$x(1 - y) = 0$
But X is true,	$x = 1$
Therefore Y is true,	$\therefore 1 - y = 0$ or $y = 1$.
3rd. Destructive Conditional Syllogism.	
If X is true, Y is true,	$x(1 - y) = 0$
But Y is not true,	$y = 0$
Therefore X is not true,	$\therefore x = 0$



George Boole
1815–1864

- George Boole:
„The Mathematical Analysis of Logic“
 - Operatoren $\times$ (und), $+$ (oder), $-$ (nicht)

1st. Disjunctive Syllogism.		
Either X is true, or Y is true (exclusive),	$x + y - 2xy = 1$	
But X is true,	$x = 1$	
Therefore Y is not true,	$\therefore y = 0$	
Either X is true, or Y is true (not exclusive),	$x + y - xy = 1$	
But X is not true,	$x = 0$	
Therefore Y is true,	$\therefore y = 1$	
2nd. Constructive Conditional Syllogism.		
If X is true, Y is true,	$x(1 - y) = 0$	
But X is true,	$x = 1$	
Therefore Y is true,	$\therefore 1 - y = 0$ or $y = 1$.	
3rd. Destructive Conditional Syllogism.		
If X is true, Y is true,	$x(1 - y) = 0$	
But Y is not true,	$y = 0$	
Therefore X is not true,	$\therefore x = 0$	



George Boole
1815–1864

- George Boole:
„The Mathematical Analysis of Logic“
 - Operatoren $\times$ (und), $+$ (oder), $-$ (nicht)
 - Rechengesetze

1st. Disjunctive Syllogism.	
Either X is true, or Y is true (exclusive),	$x + y - 2xy = 1$
But X is true,	$x = 1$
Therefore Y is not true,	$\therefore y = 0$
Either X is true, or Y is true (not exclusive),	
But X is not true,	$x + y - xy = 1$
Therefore Y is true,	$x = 0$
	$\therefore y = 1$
2nd. Constructive Conditional Syllogism.	
If X is true, Y is true,	$x(1 - y) = 0$
But X is true,	$x = 1$
Therefore Y is true,	$\therefore 1 - y = 0$ or $y = 1$.
3rd. Destructive Conditional Syllogism.	
If X is true, Y is true,	$x(1 - y) = 0$
But Y is not true,	$y = 0$
Therefore X is not true,	$\therefore x = 0$



George Boole
1815–1864

- George Boole:
„The Mathematical Analysis of Logic“
 - Operatoren $\times$ (und), $+$ (oder), $-$ (nicht)
 - Rechengesetze
 - disjunktive Normalform

1st. Disjunctive Syllogism.	
Either X is true, or Y is true (exclusive),	$x + y - 2xy = 1$
But X is true,	$x = 1$
Therefore Y is not true,	$\therefore y = 0$
Either X is true, or Y is true (not exclusive),	$x + y - xy = 1$
But X is not true,	$x = 0$
Therefore Y is true,	$\therefore y = 1$
2nd. Constructive Conditional Syllogism.	
If X is true, Y is true,	$x(1 - y) = 0$
But X is true,	$x = 1$
Therefore Y is true,	$\therefore 1 - y = 0$ or $y = 1$.
3rd. Destructive Conditional Syllogism.	
If X is true, Y is true,	$x(1 - y) = 0$
But Y is not true,	$y = 0$
Therefore X is not true,	$\therefore x = 0$



George Boole
1815–1864

- George Boole:
„The Mathematical Analysis of Logic“
 - Operatoren $\times$ (und), $+$ (oder), $-$ (nicht)
 - Rechengesetze
 - disjunktive Normalform
 - Entscheidungsverfahren

1st. Disjunctive Syllogism.	
Either X is true, or Y is true (exclusive),	$x + y - 2xy = 1$
But X is true,	$x = 1$
Therefore Y is not true,	$\therefore y = 0$
Either X is true, or Y is true (not exclusive),	$x + y - xy = 1$
But X is not true,	$x = 0$
Therefore Y is true,	$\therefore y = 1$
2nd. Constructive Conditional Syllogism.	
If X is true, Y is true,	$x(1 - y) = 0$
But X is true,	$x = 1$
Therefore Y is true,	$\therefore 1 - y = 0$ or $y = 1$.
3rd. Destructive Conditional Syllogism.	
If X is true, Y is true,	$x(1 - y) = 0$
But Y is not true,	$y = 0$
Therefore X is not true,	$\therefore x = 0$



George Boole
1815–1864

- George Boole:
„The Mathematical Analysis of Logic“
 - Operatoren $\times$ (und), $+$ (oder), $-$ (nicht)
 - Rechengesetze
 - disjunktive Normalform
 - Entscheidungsverfahren
- ↪ Aussagenlogik

1st. Disjunctive Syllogism.	
Either X is true, or Y is true (exclusive),	$x + y - 2xy = 1$
But X is true,	$x = 1$
Therefore Y is not true,	$\therefore y = 0$
Either X is true, or Y is true (not exclusive),	$x + y - xy = 1$
But X is not true,	$x = 0$
Therefore Y is true,	$\therefore y = 1$
2nd. Constructive Conditional Syllogism.	
If X is true, Y is true,	$x(1 - y) = 0$
But X is true,	$x = 1$
Therefore Y is true,	$\therefore 1 - y = 0$ or $y = 1$.
3rd. Destructive Conditional Syllogism.	
If X is true, Y is true,	$x(1 - y) = 0$
But Y is not true,	$y = 0$
Therefore X is not true,	$\therefore x = 0$



George Boole
1815–1864

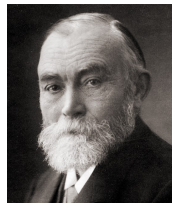
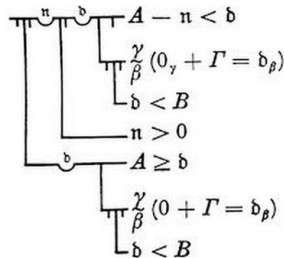
- George Boole:
„The Mathematical Analysis of Logic“
 - Operatoren $\times$ (und), $+$ (oder), $-$ (nicht)
 - Rechengesetze
 - disjunktive Normalform
 - Entscheidungsverfahren $\rightsquigarrow$ Aussagenlogik

1st. Disjunctive Syllogism.	
Either X is true, or Y is true (exclusive),	$x + y - 2xy = 1$
But X is true,	$x = 1$
Therefore Y is not true,	$\therefore y = 0$
Either X is true, or Y is true (not exclusive),	$x + y - xy = 1$
But X is not true,	$x = 0$
Therefore Y is true,	$\therefore y = 1$
2nd. Constructive Conditional Syllogism.	
If X is true, Y is true,	$x(1 - y) = 0$
But X is true,	$x = 1$
Therefore Y is true,	$\therefore 1 - y = 0$ or $y = 1$.
3rd. Destructive Conditional Syllogism.	
If X is true, Y is true,	$x(1 - y) = 0$
But Y is not true,	$y = 0$
Therefore X is not true,	$\therefore x = 0$



George Boole
1815–1864

- Gottlob Frege: „Begriffsschrift“



Gottlob Frege
1848–1925

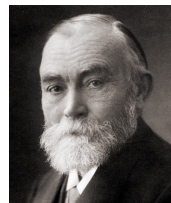
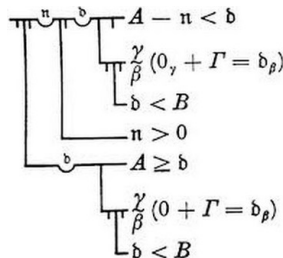
- George Boole:
„The Mathematical Analysis of Logic“
 - Operatoren $\times$ (und), $+$ (oder), $-$ (nicht)
 - Rechengesetze
 - disjunktive Normalform
 - Entscheidungsverfahren $\rightsquigarrow$ Aussagenlogik

- Gottlob Frege: „Begriffsschrift“
 - Formalisierung mathematischer Aussagen

1st. Disjunctive Syllogism.	
Either X is true, or Y is true (exclusive),	$x + y - 2xy = 1$
But X is true,	$x = 1$
Therefore Y is not true,	$\therefore y = 0$
Either X is true, or Y is true (not exclusive),	$x + y - xy = 1$
But X is not true,	$x = 0$
Therefore Y is true,	$\therefore y = 1$
2nd. Constructive Conditional Syllogism.	
If X is true, Y is true,	$x(1 - y) = 0$
But X is true,	$x = 1$
Therefore Y is true,	$\therefore 1 - y = 0$ or $y = 1$.
3rd. Destructive Conditional Syllogism.	
If X is true, Y is true,	$x(1 - y) = 0$
But Y is not true,	$y = 0$
Therefore X is not true,	$\therefore x = 0$



George Boole
1815–1864



Gottlob Frege
1848–1925

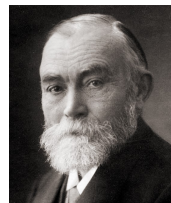
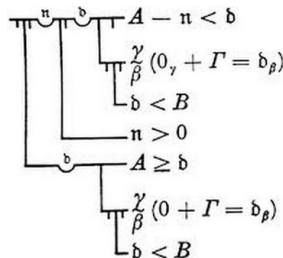
- George Boole:
„The Mathematical Analysis of Logic“
 - Operatoren $\times$ (und), $+$ (oder), $-$ (nicht)
 - Rechengesetze
 - disjunktive Normalform
 - Entscheidungsverfahren $\rightsquigarrow$ Aussagenlogik

- Gottlob Frege: „Begriffsschrift“
 - Formalisierung mathematischer Aussagen
 - „für alle x gilt ...“

1st. Disjunctive Syllogism.	
Either X is true, or Y is true (exclusive),	$x + y - 2xy = 1$
But X is true,	$x = 1$
Therefore Y is not true,	$\therefore y = 0$
Either X is true, or Y is true (not exclusive),	$x + y - xy = 1$
But X is not true,	$x = 0$
Therefore Y is true,	$\therefore y = 1$
2nd. Constructive Conditional Syllogism.	
If X is true, Y is true,	$x(1 - y) = 0$
But X is true,	$x = 1$
Therefore Y is true,	$\therefore 1 - y = 0$ or $y = 1$.
3rd. Destructive Conditional Syllogism.	
If X is true, Y is true,	$x(1 - y) = 0$
But Y is not true,	$y = 0$
Therefore X is not true,	$\therefore x = 0$



George Boole
1815–1864



Gottlob Frege
1848–1925

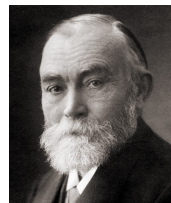
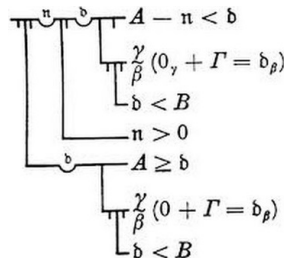
- George Boole:
„The Mathematical Analysis of Logic“
 - Operatoren $\times$ (und), $+$ (oder), $-$ (nicht)
 - Rechengesetze
 - disjunktive Normalform
 - Entscheidungsverfahren $\rightsquigarrow$ Aussagenlogik

1st. Disjunctive Syllogism.	
Either X is true, or Y is true (exclusive),	$x + y - 2xy = 1$
But X is true,	$x = 1$
Therefore Y is not true,	$\therefore y = 0$
Either X is true, or Y is true (not exclusive),	$x + y - xy = 1$
But X is not true,	$x = 0$
Therefore Y is true,	$\therefore y = 1$
2nd. Constructive Conditional Syllogism.	
If X is true, Y is true,	$x(1 - y) = 0$
But X is true,	$x = 1$
Therefore Y is true,	$\therefore 1 - y = 0$ or $y = 1$.
3rd. Destructive Conditional Syllogism.	
If X is true, Y is true,	$x(1 - y) = 0$
But Y is not true,	$y = 0$
Therefore X is not true,	$\therefore x = 0$



George Boole
1815–1864

- Gottlob Frege: „Begriffsschrift“
 - Formalisierung mathematischer Aussagen
 - „für alle x gilt ...“
 - „es gibt ein x , für das gilt ...“



Gottlob Frege
1848–1925

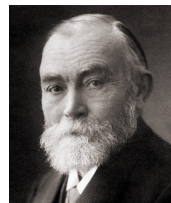
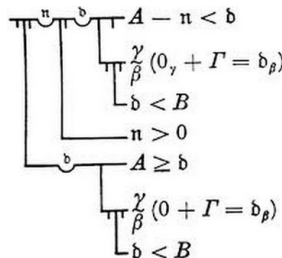
- George Boole:
„The Mathematical Analysis of Logic“
 - Operatoren $\times$ (und), $+$ (oder), $-$ (nicht)
 - Rechengesetze
 - disjunktive Normalform
 - Entscheidungsverfahren $\rightsquigarrow$ Aussagenlogik

- Gottlob Frege: „Begriffsschrift“
 - Formalisierung mathematischer Aussagen
 - „für alle x gilt ...“
 - „es gibt ein x , für das gilt ...“ $\rightsquigarrow$ Prädikatenlogik

1st. Disjunctive Syllogism.	
Either X is true, or Y is true (exclusive),	$x + y - 2xy = 1$
But X is true,	$x = 1$
Therefore Y is not true,	$\therefore y = 0$
Either X is true, or Y is true (not exclusive),	$x + y - xy = 1$
But X is not true,	$x = 0$
Therefore Y is true,	$\therefore y = 1$
2nd. Constructive Conditional Syllogism.	
If X is true, Y is true,	$x(1 - y) = 0$
But X is true,	$x = 1$
Therefore Y is true,	$\therefore 1 - y = 0$ or $y = 1$.
3rd. Destructive Conditional Syllogism.	
If X is true, Y is true,	$x(1 - y) = 0$
But Y is not true,	$y = 0$
Therefore X is not true,	$\therefore x = 0$

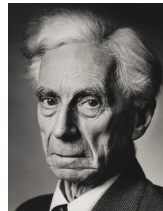


George Boole
1815–1864



Gottlob Frege
1848–1925

- Bertrand Russell, Alfred North Whitehead: „Principia Mathematica“

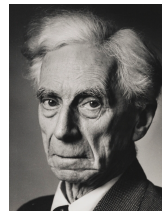


Bertrand Russell
1872–1970



A.N. Whitehead
1861–1947

- Bertrand Russell, Alfred North Whitehead: „Principia Mathematica“
 - Ziel: Fundierung der Mathematik auf Logik

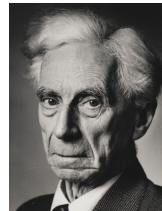


Bertrand Russell
1872–1970



A.N. Whitehead
1861–1947

- Bertrand Russell, Alfred North Whitehead: „Principia Mathematica“
 - Ziel: Fundierung der Mathematik auf Logik
 - Formalisierung von Beweisen



Bertrand Russell
1872–1970



A.N. Whitehead
1861–1947

■ Bertrand Russell, Alfred North Whitehead: „Principia Mathematica“

- Ziel: Fundierung der Mathematik auf Logik
- Formalisierung von Beweisen
- Inferenz-Regeln

*54·43. $\vdash \therefore \alpha, \beta \in 1 \supset : \alpha \cap \beta = \Lambda \equiv . \alpha \cup \beta \in 2$

Dem.

$\vdash . *54\cdot26 \supset \vdash \therefore \alpha = \iota'x . \beta = \iota'y \supset : \alpha \cup \beta \in 2 \equiv . x \neq y .$

[*51·231] $\equiv . \iota'x \cap \iota'y = \Lambda .$

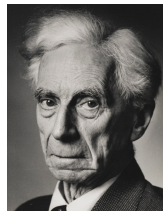
[*13·12] $\equiv . \alpha \cap \beta = \Lambda \quad (1)$

$\vdash . (1) . *11\cdot11\cdot35 \supset$

$\vdash \therefore (\exists x, y) . \alpha = \iota'x . \beta = \iota'y \supset : \alpha \cup \beta \in 2 \equiv . \alpha \cap \beta = \Lambda \quad (2)$

$\vdash . (2) . *11\cdot54 . *52\cdot1 \supset \vdash . \text{Prop}$

From this proposition it will follow, when arithmetical addition has been defined, that $1 + 1 = 2$.



Bertrand Russell
1872–1970



A.N. Whitehead
1861–1947

- David Hilbert: Hilbert-Programm



David Hilbert
1862–1943

- David Hilbert: [Hilbert-Programm](#)
 - Formalisierung aller Bereiche der Mathematik



David Hilbert
1862–1943

- David Hilbert: [Hilbert-Programm](#)
 - Formalisierung aller Bereiche der Mathematik
 - Nachweis der Widerspruchsfreiheit der Formalisierung



David Hilbert
1862–1943

- David Hilbert: Hilbert-Programm
 - Formalisierung aller Bereiche der Mathematik
 - Nachweis der Widerspruchsfreiheit der Formalisierung
 - Nachweis der Vollständigkeit der Beweisverfahren



David Hilbert
1862–1943

■ David Hilbert: Hilbert-Programm

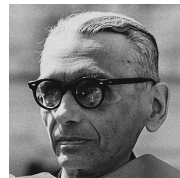
- Formalisierung aller Bereiche der Mathematik
- Nachweis der Widerspruchsfreiheit der Formalisierung
- Nachweis der **Vollständigkeit** der Beweisverfahren
- ~> Mechanisches Ableiten aller wahren Sätze

Wir müssen wissen, wir werden wissen!



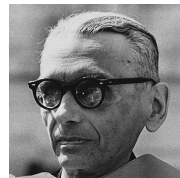
David Hilbert
1862–1943

- Kurt Gödel: „Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme“



Kurt Gödel
1906–1978

- Kurt Gödel: „Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme“
 - Vollständigkeit der Prädikatenlogik erster Stufe:
Jeder wahre Satz (in PL1) ist beweisbar.

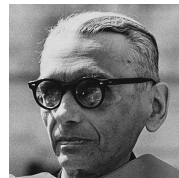


Kurt Gödel
1906–1978

- Kurt Gödel: „Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme“
 - Vollständigkeit der Prädikatenlogik erster Stufe:
Jeder wahre Satz (in PL1) ist beweisbar.
 - Unvollständigkeit der Prädikatenlogik zweiter Stufe:
Es gibt (in PL2) wahre Sätze, die nicht beweisbar sind.

Die Logik wird nie mehr dieselbe sein.

John von Neumann (1903–1957)



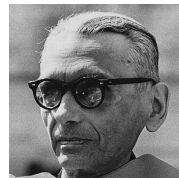
Kurt Gödel
1906–1978

- Kurt Gödel: „Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme“
 - Vollständigkeit der Prädikatenlogik erster Stufe:
Jeder wahre Satz (in PL1) ist beweisbar.
 - Unvollständigkeit der Prädikatenlogik zweiter Stufe:
Es gibt (in PL2) wahre Sätze, die nicht beweisbar sind.

Die Logik wird nie mehr dieselbe sein.

John von Neumann (1903–1957)

- Alan Turing: „On computable numbers, with an application to the Entscheidungsproblem“



Kurt Gödel
1906–1978



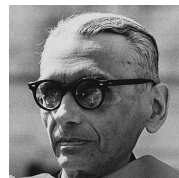
Alan Turing
1912–1954

- Kurt Gödel: „Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme“
 - Vollständigkeit der Prädikatenlogik erster Stufe:
Jeder wahre Satz (in PL1) ist beweisbar.
 - Unvollständigkeit der Prädikatenlogik zweiter Stufe:
Es gibt (in PL2) wahre Sätze, die nicht beweisbar sind.

Die Logik wird nie mehr dieselbe sein.

John von Neumann (1903–1957)

- Alan Turing: „On computable numbers, with an application to the Entscheidungsproblem“
 - Unentscheidbarkeit des Halteproblems für Turing-Maschinen (Computer)



Kurt Gödel
1906–1978



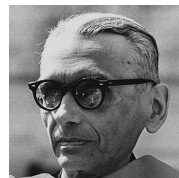
Alan Turing
1912–1954

- Kurt Gödel: „Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme“
 - Vollständigkeit der Prädikatenlogik erster Stufe:
Jeder wahre Satz (in PL1) ist beweisbar.
 - Unvollständigkeit der Prädikatenlogik zweiter Stufe:
Es gibt (in PL2) wahre Sätze, die nicht beweisbar sind.

Die Logik wird nie mehr dieselbe sein.

John von Neumann (1903–1957)

- Alan Turing: „On computable numbers, with an application to the Entscheidungsproblem“
 - Unentscheidbarkeit des [Halteproblems](#) für Turing-Maschinen (Computer)
 - Unentscheidbarkeit des [Erfüllbarkeitsproblems](#) für PL1



Kurt Gödel
1906–1978



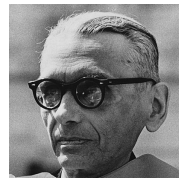
Alan Turing
1912–1954

- Kurt Gödel: „Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme“
 - Vollständigkeit der Prädikatenlogik erster Stufe:
Jeder wahre Satz (in PL1) ist beweisbar.
 - Unvollständigkeit der Prädikatenlogik zweiter Stufe:
Es gibt (in PL2) wahre Sätze, die nicht beweisbar sind.

Die Logik wird nie mehr dieselbe sein.

John von Neumann (1903–1957)

- Alan Turing: „On computable numbers, with an application to the Entscheidungsproblem“
 - Unentscheidbarkeit des **Halteproblems** für Turing-Maschinen (Computer)
 - Unentscheidbarkeit des **Erfüllbarkeitsproblems** für PL1
- ~> 3. Semester



Kurt Gödel
1906–1978



Alan Turing
1912–1954

- „Dieser Satz ist nicht beweisbar.“

- „Dieser Satz ist nicht beweisbar.“
- Russell'sche Antinomie: $U = \{T \mid T \notin T\}$

- „Dieser Satz ist nicht beweisbar.“
- Russell'sche Antinomie: $U = \{T \mid T \notin T\}$
- „Der Barbier rasiert genau die Männer, die sich nicht selbst rasieren.“

- „Dieser Satz ist nicht beweisbar.“
- Russell'sche Antinomie: $U = \{T \mid T \notin T\}$
- „Der Barbier rasiert genau die Männer, die sich nicht selbst rasieren.“

Douglas R. Hofstadter: Gödel Escher Bach

7.7 Incompleteness 253

$$\vdash \psi \leftrightarrow \exists y [\overline{Prov}(\overline{\neg\psi}, y) \wedge \forall z < y \neg \overline{Prov}(\overline{\neg\psi}, z)] \quad (1)$$

Claim: **PA** is consistent $\Rightarrow \not\vdash \psi$ and $\not\vdash \neg\psi$

Proof. (i) Suppose $\vdash \psi$ then there exists a n such that $Prov(\overline{\neg\psi}, n)$ so

$$\vdash \overline{Prov}(\overline{\neg\psi}, \overline{n}) \quad (2)$$

From (1) and (2) it follows that $\vdash \exists y < \overline{n} \overline{Prov}(\overline{\neg\psi}, y)$, i.e. $\vdash \overline{Prov}(\overline{\neg\psi}, \overline{0}) \vee \dots \vee \overline{Prov}(\overline{\neg\psi}, \overline{n-1})$. Note that $\overline{Prov}$ is Δ_0 , thus the following holds: $\vdash \sigma \vee \tau \Leftrightarrow \vdash \sigma$ or $\vdash \tau$, so $\vdash \overline{Prov}(\overline{\neg\psi}, \overline{0})$ or $\dots$ or $\vdash \overline{Prov}(\overline{\neg\psi}, \overline{n-1})$ hence $Prov(\overline{\neg\psi}, i)$ for some $i < n \Rightarrow \vdash \neg\psi \Rightarrow$ **PA** is inconsistent.

- „Dieser Satz ist nicht beweisbar.“
- Russell'sche Antinomie: $U = \{T \mid T \notin T\}$
- „Der Barbier rasiert genau die Männer, die sich nicht selbst rasieren.“

Douglas R. Hofstadter: *Gödel* *Escher* *Bach*

$\vdash \psi \leftarrow$

Claim:

Proof.

$\vdash \overline{Prov}$

From (

$\dots \vee \overline{P}$

$\vdash \sigma \vee \tau$

hence



7.7 Incompleteness 253

$(\overline{\vdash \psi}, z)] \quad (1)$

such that $Prov(\overline{\vdash \psi}, n)$ so

$\overline{\vdash (\neg \psi)}, y)$, i.e. $\vdash \overline{Prov}(\overline{\vdash \neg \psi}, 0) \vee$

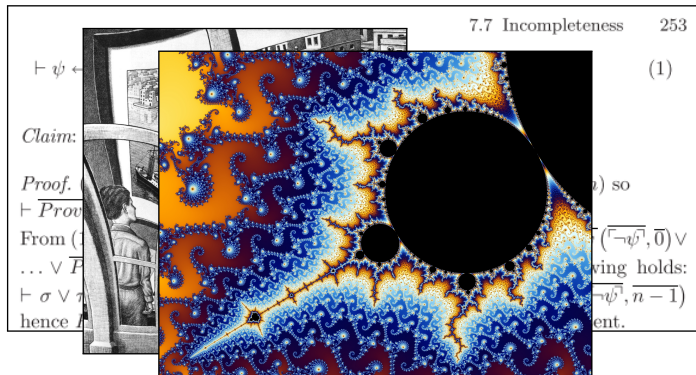
is Δ_0 , thus the following holds:

) or ... or $\vdash \overline{Prov}(\overline{\vdash \neg \psi}, n-1)$

$\Rightarrow$ **PA** is inconsistent.

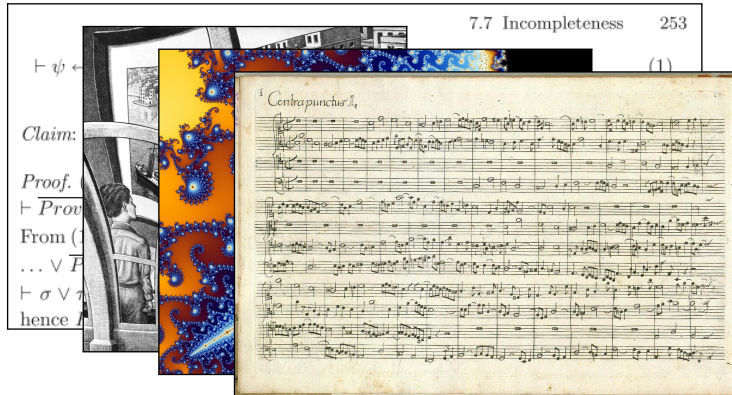
- „Dieser Satz ist nicht beweisbar.“
- Russell'sche Antinomie: $U = \{T \mid T \notin T\}$
- „Der Barbier rasiert genau die Männer, die sich nicht selbst rasieren.“

Douglas R. Hofstadter: *Gödel* *Escher* *Bach*



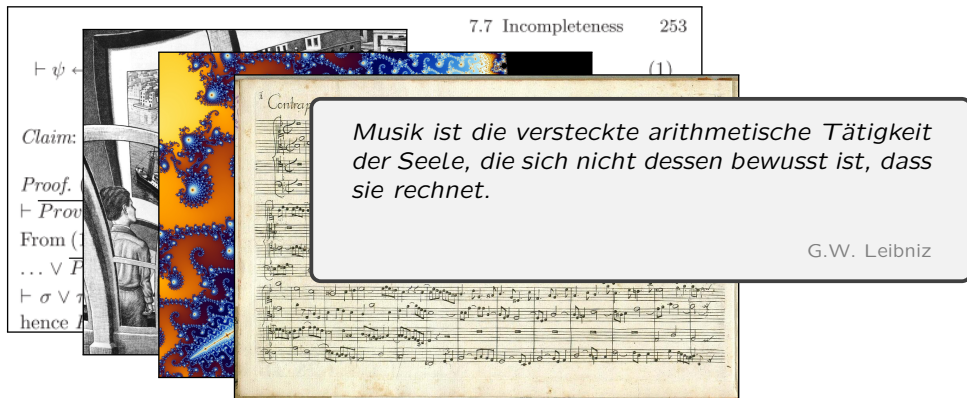
- „Dieser Satz ist nicht beweisbar.“
- Russell'sche Antinomie: $U = \{T \mid T \notin T\}$
- „Der Barbier rasiert genau die Männer, die sich nicht selbst rasieren.“

Douglas R. Hofstadter: Gödel Escher Bach



- „Dieser Satz ist nicht beweisbar.“
- Russell'sche Antinomie: $U = \{T \mid T \notin T\}$
- „Der Barbier rasiert genau die Männer, die sich nicht selbst rasieren.“

Douglas R. Hofstadter: Gödel Escher Bach



- Alle Wörter bestehen aus den Buchstaben M, I und U

Nach Douglas R. Hofstadter, *Gödel Escher Bach*

- Alle Wörter bestehen aus den Buchstaben M, I und U
- Eine Ableitung beginnt immer mit MI

Nach Douglas R. Hofstadter, *Gödel Escher Bach*

- Alle Wörter bestehen aus den Buchstaben M, I und U
- Eine Ableitung beginnt immer mit MI
- Es gelten die folgenden Ableitungsregeln:

Nach Douglas R. Hofstadter, *Gödel Escher Bach*

- Alle Wörter bestehen aus den Buchstaben M, I und U
- Eine Ableitung beginnt immer mit MI
- Es gelten die folgenden Ableitungsregeln:
 - 1 Wenn ein Wort mit I endet, darf man U anhängen

Nach Douglas R. Hofstadter, *Gödel Escher Bach*

- Alle Wörter bestehen aus den Buchstaben M, I und U
- Eine Ableitung beginnt immer mit MI
- Es gelten die folgenden Ableitungsregeln:
 - 1 Wenn ein Wort mit I endet, darf man U anhängen
 - 2 III darf durch U ersetzt werden

Nach Douglas R. Hofstadter, *Gödel Escher Bach*

- Alle Wörter bestehen aus den Buchstaben M, I und U
- Eine Ableitung beginnt immer mit MI
- Es gelten die folgenden Ableitungsregeln:
 - 1 Wenn ein Wort mit I endet, darf man U anhängen
 - 2 III darf durch U ersetzt werden
 - 3 UU darf entfernt werden

Nach Douglas R. Hofstadter, *Gödel Escher Bach*

- Alle Wörter bestehen aus den Buchstaben M, I und U
- Eine Ableitung beginnt immer mit MI
- Es gelten die folgenden Ableitungsregeln:
 - 1 Wenn ein Wort mit I endet, darf man U anhängen
 - 2 III darf durch U ersetzt werden
 - 3 UU darf entfernt werden
 - 4 Das Teilwort nach einem M darf verdoppelt werden

Nach Douglas R. Hofstadter, *Gödel Escher Bach*

- Alle Wörter bestehen aus den Buchstaben M, I und U

- Alle Wörter bestehen aus den Buchstaben M, I und U
- Jede Ableitung beginnt mit MI

- Alle Wörter bestehen aus den Buchstaben M, I und U
- Jede Ableitung beginnt mit MI
- x und y stehen für beliebige (Teil-)wörter

- Alle Wörter bestehen aus den Buchstaben M, I und U
- Jede Ableitung beginnt mit MI
- x und y stehen für beliebige (Teil-)wörter
- Ableitungsregeln:

- Alle Wörter bestehen aus den Buchstaben M, I und U
- Jede Ableitung beginnt mit MI
- x und y stehen für beliebige (Teil-)wörter
- Ableitungsregeln:
 - 1 $xI \rightarrow xIU$

- Alle Wörter bestehen aus den Buchstaben M, I und U
- Jede Ableitung beginnt mit MI
- x und y stehen für beliebige (Teil-)wörter
- Ableitungsregeln:
 - 1 $xI \rightarrow xIU$
 - 2 $xIIIIy \rightarrow xUy$

- Alle Wörter bestehen aus den Buchstaben M, I und U
- Jede Ableitung beginnt mit MI
- x und y stehen für beliebige (Teil-)wörter
- Ableitungsregeln:
 - 1 $xI \rightarrow xIU$
 - 2 $xIIIIy \rightarrow xUy$
 - 3 $xUUy \rightarrow xy$

- Alle Wörter bestehen aus den Buchstaben M, I und U
- Jede Ableitung beginnt mit MI
- x und y stehen für beliebige (Teil-)wörter
- Ableitungsregeln:
 - 1 $xI \rightarrow xIU$
 - 2 $xIIIIy \rightarrow xUy$
 - 3 $xUUy \rightarrow xy$
 - 4 $Mx \rightarrow Mxx$

- Alle Wörter bestehen aus den Buchstaben M, I und U
- Jede Ableitung beginnt mit MI
- x und y stehen für beliebige (Teil-)wörter
- Ableitungsregeln:
 - 1 $xI \rightarrow xIU$
 - 2 $xIIIIy \rightarrow xUy$
 - 3 $xUUy \rightarrow xy$
 - 4 $Mx \rightarrow Mxx$
- Man schreibt $x \vdash_i y$, wenn sich x durch Anwendung der Regel Nr. i in y überführen lässt

- Alle Wörter bestehen aus den Buchstaben M, I und U
- Jede Ableitung beginnt mit MI
- x und y stehen für beliebige (Teil-)wörter
- Ableitungsregeln:
 - 1 $xI \rightarrow xIU$
 - 2 $xIIIIy \rightarrow xUy$
 - 3 $xUUy \rightarrow xy$
 - 4 $Mx \rightarrow Mxx$
- Man schreibt $x \vdash_i y$, wenn sich x durch Anwendung der Regel Nr. i in y überführen lässt

Beispiel 1.1 (Ableitung im MIU-System)

$MI \vdash_4 MII \vdash_4 MIIII \vdash_2 MUI \vdash_4 MUIUI$

- Jede Ableitung beginnt mit MI
- Ableitungsregeln:
 - 1 $xI \rightarrow xIU$
 - 2 $xIIIIy \rightarrow xUy$
 - 3 $xUUy \rightarrow xy$
 - 4 $Mx \rightarrow Mxx$
- Man schreibt $x \vdash_i y$, wenn sich x durch Anwendung der Regel Nr. i in y überführen lässt

Übung 1.2

Geben Sie, falls möglich, für die folgenden Wörter Ableitungen an:

- 1 MUIU
- 2 MIIIII
- 3 MUUUI
- 4 MU

$$1 \quad xI \rightarrow xIU$$

$$2 \quad xIIIIy \rightarrow xUy$$

$$3 \quad xUUy \rightarrow xy$$

$$4 \quad Mx \rightarrow Mxx$$

1 $xI \rightarrow xIU$

2 $xIIIIy \rightarrow xUy$

3 $xUUy \rightarrow xy$

4 $Mx \rightarrow Mxx$

Lösung 1.2

1 $MI \vdash_4 MII \vdash_4 MIIII \vdash_1 MIIIIU \vdash_2 MUIU$

- 1 $xI \rightarrow xIU$
- 2 $xIIIIy \rightarrow xUy$
- 3 $xUUy \rightarrow xy$
- 4 $Mx \rightarrow Mxx$

Lösung 1.2

- 1 $MI \vdash_4 MII \vdash_4 MIIII \vdash_1 MIIIIU \vdash_2 MUIU$
- 2 $MI \vdash_4 MII \vdash_4 MIIII \vdash_4 MIIIIIIII \vdash_1 MIIIIIIIIU \vdash_2 MIIIIIUU \vdash_3 MIIII$

- 1 $xI \rightarrow xIU$
- 2 $xIIIIy \rightarrow xUy$
- 3 $xUUy \rightarrow xy$
- 4 $Mx \rightarrow Mxx$

Lösung 1.2

- 1 $MI \vdash_4 MII \vdash_4 MIIII \vdash_1 MIIIIU \vdash_2 MUIU$
- 2 $MI \vdash_4 MII \vdash_4 MIIII \vdash_4 MIIIIIIII \vdash_1 MIIIIIIIIU \vdash_2 MIIIIIUU \vdash_3 MIIIII$
- 3 $MI \vdash_4 MII \vdash_4 MIIII \vdash_4 MIIIIIIII \vdash_2 MUIIIII \vdash_2 MUUII \vdash_4 MUUIIUUII \vdash_3 MUUIIII \vdash_2 MUUII$

- 1 $xI \rightarrow xIU$
- 2 $xIIIIy \rightarrow xUy$
- 3 $xUUy \rightarrow xy$
- 4 $Mx \rightarrow Mxx$

Lösung 1.2

- 4 MU kann nicht hergeleitet werden!

- 1 $xI \rightarrow xIU$
- 2 $xIIIIy \rightarrow xUy$
- 3 $xUUy \rightarrow xy$
- 4 $Mx \rightarrow Mxx$

Lösung 1.2

- 4 MU kann nicht hergeleitet werden!

- 1 $xI \rightarrow xIU$
- 2 $xIIIIy \rightarrow xUy$
- 3 $xUUy \rightarrow xy$
- 4 $Mx \rightarrow Mxx$

Lösung 1.2

- 4 MU kann nicht hergeleitet werden!

Beweis.

- Betrachte Anzahl der I in ableitbaren Wörtern

- 1 $xI \rightarrow xIU$
- 2 $xIIIy \rightarrow xUy$
- 3 $xUUy \rightarrow xy$
- 4 $Mx \rightarrow Mxx$

Lösung 1.2

- 4 MU kann nicht hergeleitet werden!

Beweis.

- Betrachte Anzahl der I in ableitbaren Wörtern
- **Invariante**: in allen herleitbaren Wörtern ist $|x|_I$ nicht ohne Rest durch 3 teilbar

- 1 $xI \rightarrow xIU$
- 2 $xIIIy \rightarrow xUy$
- 3 $xUUy \rightarrow xy$
- 4 $Mx \rightarrow Mxx$

Lösung 1.2

- 4 MU kann nicht hergeleitet werden!

Beweis.

- Betrachte Anzahl der I in ableitbaren Wörtern
- **Invariante:** in allen herleitbaren Wörtern ist $|x|_I$ nicht ohne Rest durch 3 teilbar
 - $|MI|_I = 1$; Rest 1

- 1 $xI \rightarrow xIU$
- 2 $xIIIy \rightarrow xUy$
- 3 $xUUy \rightarrow xy$
- 4 $Mx \rightarrow Mxx$

Lösung 1.2

- 4 MU kann nicht hergeleitet werden!

Beweis.

- Betrachte Anzahl der I in ableitbaren Wörtern
- **Invariante:** in allen herleitbaren Wörtern ist $|x|_I$ nicht ohne Rest durch 3 teilbar
 - $|MI|_I = 1$; Rest 1
 - Regeln 1 und 3 ändern die Anzahl der I nicht

- 1 $xI \rightarrow xIU$
- 2 $xIIIy \rightarrow xUy$
- 3 $xUUy \rightarrow xy$
- 4 $Mx \rightarrow Mxx$

Lösung 1.2

- 4 MU kann nicht hergeleitet werden!

Beweis.

- Betrachte Anzahl der I in ableitbaren Wörtern
- **Invariante:** in allen herleitbaren Wörtern ist $|x|_I$ nicht ohne Rest durch 3 teilbar
 - $|MI|_I = 1$; Rest 1
 - Regeln 1 und 3 ändern die Anzahl der I nicht
 - Regel 4 verdoppelt die Anzahl der I
Rest 1 $\rightsquigarrow$ Rest 2, Rest 2 $\rightsquigarrow$ Rest 1

- 1 $xI \rightarrow xIU$
- 2 $xIIIy \rightarrow xUy$
- 3 $xUUy \rightarrow xy$
- 4 $Mx \rightarrow Mxx$

Lösung 1.2

- 4 MU kann nicht hergeleitet werden!

Beweis.

- Betrachte Anzahl der I in ableitbaren Wörtern
- **Invariante:** in allen herleitbaren Wörtern ist $|x|_I$ nicht ohne Rest durch 3 teilbar
 - $|MI|_I = 1$; Rest 1
 - Regeln 1 und 3 ändern die Anzahl der I nicht
 - Regel 4 verdoppelt die Anzahl der I
Rest 1 $\rightsquigarrow$ Rest 2, Rest 2 $\rightsquigarrow$ Rest 1
 - Regel 2 reduziert die Anzahl der I um 3; Rest bleibt gleich

- 1 $xI \rightarrow xIU$
- 2 $xIIIy \rightarrow xUy$
- 3 $xUUy \rightarrow xy$
- 4 $Mx \rightarrow Mxx$

Lösung 1.2

- 4 MU kann nicht hergeleitet werden!

Beweis.

- Betrachte Anzahl der I in ableitbaren Wörtern
- **Invariante:** in allen herleitbaren Wörtern ist $|x|_I$ nicht ohne Rest durch 3 teilbar
 - $|MI|_I = 1$; Rest 1
 - Regeln 1 und 3 ändern die Anzahl der I nicht
 - Regel 4 verdoppelt die Anzahl der I
Rest 1 $\rightsquigarrow$ Rest 2, Rest 2 $\rightsquigarrow$ Rest 1
 - Regel 2 reduziert die Anzahl der I um 3; Rest bleibt gleich
- In keinem der Fälle wird aus einem nicht-Vielfachen von 3 ein Vielfaches von 3. Aber $|MU|_I = 0$. Also ist MU nicht herleitbar. □

MIU-System	Logik
Wörter über $\{M, I, U\}$	Syntaktisch korrekte Ausdrücke

MIU-System	Logik
Wörter über $\{M, I, U\}$	Syntaktisch korrekte Ausdrücke
MI	Axiome

MIU-System	Logik
Wörter über $\{M, I, U\}$	Syntaktisch korrekte Ausdrücke
MI	Axiome
Ableitungsregeln	Beweisschritte

MIU-System	Logik
Wörter über $\{M, I, U\}$	Syntaktisch korrekte Ausdrücke
MI	Axiome
Ableitungsregeln	Beweisschritte
Ableitbare Wörter	Beweisbare Sätze

MIU-System	Logik
Wörter über $\{M, I, U\}$	Syntaktisch korrekte Ausdrücke
MI	Axiome
Ableitungsregeln	Beweisschritte
Ableitbare Wörter	Beweisbare Sätze

- Regelanwendung terminiert nicht

MIU-System	Logik
Wörter über $\{M, I, U\}$	Syntaktisch korrekte Ausdrücke
MI	Axiome
Ableitungsregeln	Beweisschritte
Ableitbare Wörter	Beweisbare Sätze

- Regelanwendung terminiert nicht
- Ableitungsregeln liefern kein [Entscheidungsverfahren](#)

1. Einführung

2. Mengen

2.1 Grundbegriffe

2.2 Mengenoperationen

3. Aussagenlogik

4. Prädikatenlogik

5. Prolog

1. Einführung

2. Mengen

2.1 Grundbegriffe

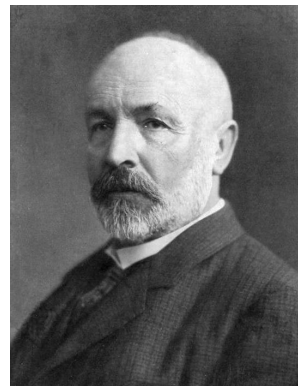
2.2 Mengenoperationen

3. Aussagenlogik

4. Prädikatenlogik

5. Prolog

*Unter einer **Menge** verstehen wir jede Zusammenfassung M von bestimmten wohlunterschiedenen Objekten m unserer Anschauung oder unseres Denkens (welche die **Elemente** von M genannt werden) zu einem Ganzen.*



Georg Cantor
(1845–1918)

Definition 2.1 (Menge, Element)

- Eine **Menge** ist eine Sammlung von Objekten, betrachtet als Einheit.
- Die Objekte heißen auch **Elemente** der Menge.

Definition 2.1 (Menge, Element)

- Eine **Menge** ist eine Sammlung von Objekten, betrachtet als Einheit.
 - Die Objekte heißen auch **Elemente** der Menge.
-
- Elemente können beliebige Objekte sein:
 - Zahlen
 - Wörter
 - Andere Mengen (!)
 - Listen, Paare, Funktionen, ...
 - ... aber auch Menschen, Fahrzeuge, Kurse an der DHBW, ...

Definition 2.1 (Menge, Element)

- Eine **Menge** ist eine Sammlung von Objekten, betrachtet als Einheit.
 - Die Objekte heißen auch **Elemente** der Menge.
-
- Elemente können beliebige Objekte sein:
 - Zahlen
 - Wörter
 - Andere Mengen (!)
 - Listen, Paare, Funktionen, ...
 - ... aber auch Menschen, Fahrzeuge, Kurse an der DHBW, ...

Die Menge **aller** in einem bestimmten Kontext betrachteten Objekte heißt oft **Universum** oder **Trägermenge**.

- Explizite Aufzählung:

- Explizite Aufzählung:
 - $A = \{2, 3, 5, 7, 11, 13\}$

- Explizite Aufzählung:
 - $A = \{2, 3, 5, 7, 11, 13\}$
 - $\mathbb{N} = \{0, 1, 2, 3, \dots\}$

- Explizite Aufzählung:
 - $A = \{2, 3, 5, 7, 11, 13\}$
 - $\mathbb{N} = \{0, 1, 2, 3, \dots\}$
- Beschreibung („Deskriptive Form“):

- Explizite Aufzählung:
 - $A = \{2, 3, 5, 7, 11, 13\}$
 - $\mathbb{N} = \{0, 1, 2, 3, \dots\}$
- Beschreibung („Deskriptive Form“):
 - $A = \{n \in \mathbb{N} \mid n \text{ ist Primzahl und } n \leq 13\}$

- Explizite Aufzählung:

- $A = \{2, 3, 5, 7, 11, 13\}$

- $\mathbb{N} = \{0, 1, 2, 3, \dots\}$

- Beschreibung („Deskriptive Form“):

- $A = \{n \in \mathbb{N} \mid n \text{ ist Primzahl und } n \leq 13\}$

- $B = \{3 \cdot n \mid n \in \mathbb{N}\}$

- Explizite Aufzählung:

- $A = \{2, 3, 5, 7, 11, 13\}$
- $\mathbb{N} = \{0, 1, 2, 3, \dots\}$

- Beschreibung („Deskriptive Form“):

- $A = \{n \in \mathbb{N} \mid n \text{ ist Primzahl und } n \leq 13\}$
- $B = \{3 \cdot n \mid n \in \mathbb{N}\}$
- $C = \{2^n \mid n \in \mathbb{N}\}$

- Explizite Aufzählung:
 - $A = \{2, 3, 5, 7, 11, 13\}$
 - $\mathbb{N} = \{0, 1, 2, 3, \dots\}$
- Beschreibung („Deskriptive Form“):
 - $A = \{n \in \mathbb{N} \mid n \text{ ist Primzahl und } n \leq 13\}$
 - $B = \{3 \cdot n \mid n \in \mathbb{N}\}$
 - $C = \{2^n \mid n \in \mathbb{N}\}$
- Mengenzugehörigkeit

- Explizite Aufzählung:

- $A = \{2, 3, 5, 7, 11, 13\}$
- $\mathbb{N} = \{0, 1, 2, 3, \dots\}$

- Beschreibung („Deskriptive Form“):

- $A = \{n \in \mathbb{N} \mid n \text{ ist Primzahl und } n \leq 13\}$
- $B = \{3 \cdot n \mid n \in \mathbb{N}\}$
- $C = \{2^n \mid n \in \mathbb{N}\}$

- Mengenzugehörigkeit

- $2 \in A$ (2 ist in A , 2 ist Element von A)

- Explizite Aufzählung:
 - $A = \{2, 3, 5, 7, 11, 13\}$
 - $\mathbb{N} = \{0, 1, 2, 3, \dots\}$
- Beschreibung („Deskriptive Form“):
 - $A = \{n \in \mathbb{N} \mid n \text{ ist Primzahl und } n \leq 13\}$
 - $B = \{3 \cdot n \mid n \in \mathbb{N}\}$
 - $C = \{2^n \mid n \in \mathbb{N}\}$
- Mengenzugehörigkeit
 - $2 \in A$ (2 ist in A, 2 ist Element von A)
 - $4 \notin A$ (4 ist nicht in A, 4 ist kein Element von A)

- Mengen sind ungeordnet

- Mengen sind ungeordnet
 - $\{a, b, c\} = \{b, c, a\} = \{c, a, b\}$

- Mengen sind ungeordnet
 - $\{a, b, c\} = \{b, c, a\} = \{c, a, b\}$
 - Geordnet sind z. B. Folgen

- Mengen sind ungeordnet
 - $\{a, b, c\} = \{b, c, a\} = \{c, a, b\}$
 - Geordnet sind z. B. Folgen
- Jedes Element kommt in einer Menge maximal einmal vor

- Mengen sind ungeordnet
 - $\{a, b, c\} = \{b, c, a\} = \{c, a, b\}$
 - Geordnet sind z. B. Folgen
- Jedes Element kommt in einer Menge maximal einmal vor
 - $\{1, 1, 1\}$ hat ein Element

- Mengen sind ungeordnet
 - $\{a, b, c\} = \{b, c, a\} = \{c, a, b\}$
 - Geordnet sind z. B. Folgen
- Jedes Element kommt in einer Menge maximal einmal vor
 - $\{1, 1, 1\}$ hat ein Element
 - Mehrfaches Vorkommen des gleichen Elements erlauben z. B. Multimengen oder Folgen

- Mengen sind ungeordnet
 - $\{a, b, c\} = \{b, c, a\} = \{c, a, b\}$
 - Geordnet sind z. B. Folgen
- Jedes Element kommt in einer Menge maximal einmal vor
 - $\{1, 1, 1\}$ hat ein Element
 - Mehrfaches Vorkommen des gleichen Elements erlauben z. B. Multimengen oder Folgen

- Mengen sind ungeordnet
 - $\{a, b, c\} = \{b, c, a\} = \{c, a, b\}$
 - Geordnet sind z. B. Folgen
- Jedes Element kommt in einer Menge maximal einmal vor
 - $\{1, 1, 1\}$ hat ein Element
 - Mehrfaches Vorkommen des gleichen Elements erlauben z. B. Multimengen oder Folgen

Definition 2.2 (Mächtigkeit)

Die Mächtigkeit einer Menge M , $|M|$, ist die Anzahl der Elemente von M .

- Mengen sind ungeordnet
 - $\{a, b, c\} = \{b, c, a\} = \{c, a, b\}$
 - Geordnet sind z. B. Folgen
- Jedes Element kommt in einer Menge maximal einmal vor
 - $\{1, 1, 1\}$ hat ein Element
 - Mehrfaches Vorkommen des gleichen Elements erlauben z. B. Multimengen oder Folgen

Definition 2.2 (Mächtigkeit)

Die Mächtigkeit einer Menge M , $|M|$, ist die Anzahl der Elemente von M .

- Einfach für endliche Mengen: $|\{a, b, c\}| = 3$

- Mengen sind ungeordnet
 - $\{a, b, c\} = \{b, c, a\} = \{c, a, b\}$
 - Geordnet sind z. B. Folgen
- Jedes Element kommt in einer Menge maximal einmal vor
 - $\{1, 1, 1\}$ hat ein Element
 - Mehrfaches Vorkommen des gleichen Elements erlauben z. B. Multimengen oder Folgen

Definition 2.2 (Mächtigkeit)

Die Mächtigkeit einer Menge M , $|M|$, ist die Anzahl der Elemente von M .

- Einfach für endliche Mengen: $|\{a, b, c\}| = 3$
- Schwierig für unendliche Mengen: $|\mathbb{N}| = |\mathbb{Z}| = |\mathbb{Q}| < |\mathbb{R}|$

Definition 2.3 (Teilmenge)

Eine Menge M_1 heißt **Teilmenge** von M_2 ($M_1 \subseteq M_2$), wenn für alle $x \in M_1$ auch $x \in M_2$ gilt.

Definition 2.3 (Teilmenge)

Eine Menge M_1 heißt **Teilmenge** von M_2 ($M_1 \subseteq M_2$), wenn für alle $x \in M_1$ auch $x \in M_2$ gilt.

Definition 2.4 (Mengengleichheit)

Zwei Mengen M_1 und M_2 sind **einander gleich** ($M_1 = M_2$), wenn für alle Elemente x gilt: $x \in M_1$ gdw. $x \in M_2$, d. h. wenn M_1 und M_2 die selben Elemente enthalten.

Definition 2.3 (Teilmenge)

Eine Menge M_1 heißt **Teilmenge** von M_2 ($M_1 \subseteq M_2$), wenn für alle $x \in M_1$ auch $x \in M_2$ gilt.

Definition 2.4 (Mengengleichheit)

Zwei Mengen M_1 und M_2 sind **einander gleich** ($M_1 = M_2$), wenn für alle Elemente x gilt: $x \in M_1$ gdw. $x \in M_2$, d. h. wenn M_1 und M_2 die selben Elemente enthalten.

Es gilt: $M_1 = M_2$ gdw. $M_1 \subseteq M_2$ und $M_2 \subseteq M_1$.

Definition 2.3 (Teilmenge)

Eine Menge M_1 heißt **Teilmenge** von M_2 ($M_1 \subseteq M_2$), wenn für alle $x \in M_1$ auch $x \in M_2$ gilt.

Definition 2.4 (Mengengleichheit)

Zwei Mengen M_1 und M_2 sind **einander gleich** ($M_1 = M_2$), wenn für alle Elemente x gilt: $x \in M_1$ gdw. $x \in M_2$, d. h. wenn M_1 und M_2 die selben Elemente enthalten.

Es gilt: $M_1 = M_2$ gdw. $M_1 \subseteq M_2$ und $M_2 \subseteq M_1$.

Vokabular:

- **gdw.** steht für „genau dann, wenn“ (dann und nur dann, wenn)
- **englisch: iff**; „if and only if“

Definition 2.5 (Echte Teilmenge)

Eine Menge M_1 heißt **echte Teilmenge** von M_2 ($M_1 \subset M_2$), wenn $M_1 \subseteq M_2$ und $M_1 \neq M_2$ gilt.

Definition 2.5 (Echte Teilmenge)

Eine Menge M_1 heißt **echte Teilmenge** von M_2 ($M_1 \subset M_2$), wenn $M_1 \subseteq M_2$ und $M_1 \neq M_2$ gilt.

■ Analog definieren wir Obermengen:

- $M_1 \supseteq M_2$ gdw. $M_2 \subseteq M_1$
- $M_1 \supset M_2$ gdw. $M_2 \subset M_1$

Definition 2.5 (Echte Teilmenge)

Eine Menge M_1 heißt **echte Teilmenge** von M_2 ($M_1 \subset M_2$), wenn $M_1 \subseteq M_2$ und $M_1 \neq M_2$ gilt.

- Analog definieren wir Obermengen:
 - $M_1 \supseteq M_2$ gdw. $M_2 \subseteq M_1$
 - $M_1 \supset M_2$ gdw. $M_2 \subset M_1$
- Wir schreiben $M_1 \not\subseteq M_2$, falls M_1 keine Teilmenge von M_2 ist.

Definition 2.5 (Echte Teilmenge)

Eine Menge M_1 heißt **echte Teilmenge** von M_2 ($M_1 \subset M_2$), wenn $M_1 \subseteq M_2$ und $M_1 \neq M_2$ gilt.

- Analog definieren wir Obermengen:
 - $M_1 \supseteq M_2$ gdw. $M_2 \subseteq M_1$
 - $M_1 \supset M_2$ gdw. $M_2 \subset M_1$
- Wir schreiben $M_1 \not\subseteq M_2$, falls M_1 keine Teilmenge von M_2 ist.

Notation: Manche Autoren verwenden $\subset$ statt $\subseteq$ und $\subsetneq$ statt $\subset$.

- $\emptyset = \{\}$, die leere Menge

- $\emptyset = \{\}$, die **leere Menge**
 - enthält kein Element

- $\emptyset = \{\}$, die **leere Menge**
 - enthält kein Element
 - es gilt: $\emptyset \subseteq M$ für alle Mengen M

- $\emptyset = \{\}$, die **leere Menge**
 - enthält kein Element
 - es gilt: $\emptyset \subseteq M$ für alle Mengen M
- $\mathbb{N} = \{0, 1, 2, 3 \dots\}$, die **natürlichen Zahlen**

- $\emptyset = \{\}$, die **leere Menge**
 - enthält kein Element
 - es gilt: $\emptyset \subseteq M$ für alle Mengen M
- $\mathbb{N} = \{0, 1, 2, 3 \dots\}$, die **natürlichen Zahlen**
 - Informatiker fangen bei 0 an zu zählen!

- $\emptyset = \{\}$, die **leere Menge**
 - enthält kein Element
 - es gilt: $\emptyset \subseteq M$ für alle Mengen M
- $\mathbb{N} = \{0, 1, 2, 3, \dots\}$, die **natürlichen Zahlen**
 - Informatiker fangen bei 0 an zu zählen!
- $\mathbb{N}^{\geq 1} = \{1, 2, 3, \dots\}$, die **positiven natürlichen Zahlen**

- $\emptyset = \{\}$, die **leere Menge**
 - enthält kein Element
 - es gilt: $\emptyset \subseteq M$ für alle Mengen M
- $\mathbb{N} = \{0, 1, 2, 3, \dots\}$, die **natürlichen Zahlen**
 - Informatiker fangen bei 0 an zu zählen!
- $\mathbb{N}^{\geq 1} = \{1, 2, 3, \dots\}$, die **positiven natürlichen Zahlen**
- $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$, die **ganzen Zahlen**

- $\emptyset = \{\}$, die **leere Menge**
 - enthält kein Element
 - es gilt: $\emptyset \subseteq M$ für alle Mengen M
- $\mathbb{N} = \{0, 1, 2, 3, \dots\}$, die **natürlichen Zahlen**
 - Informatiker fangen bei 0 an zu zählen!
- $\mathbb{N}^{\geq 1} = \{1, 2, 3, \dots\}$, die **positiven natürlichen Zahlen**
- $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$, die **ganzen Zahlen**
- $\mathbb{Q} = \{\frac{p}{q} \mid p \in \mathbb{Z}, q \in \mathbb{N}^{\geq 1}\}$ (die **rationalen Zahlen**)

- $\emptyset = \{\}$, die **leere Menge**
 - enthält kein Element
 - es gilt: $\emptyset \subseteq M$ für alle Mengen M
- $\mathbb{N} = \{0, 1, 2, 3, \dots\}$, die **natürlichen Zahlen**
 - Informatiker fangen bei 0 an zu zählen!
- $\mathbb{N}^{\geq 1} = \{1, 2, 3, \dots\}$, die **positiven natürlichen Zahlen**
- $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$, die **ganzen Zahlen**
- $\mathbb{Q} = \{\frac{p}{q} \mid p \in \mathbb{Z}, q \in \mathbb{N}^{\geq 1}\}$ (die **rationalen Zahlen**)
- $\mathbb{R}$, die **reellen Zahlen**

- $\emptyset = \{\}$, die **leere Menge**
 - enthält kein Element
 - es gilt: $\emptyset \subseteq M$ für alle Mengen M
- $\mathbb{N} = \{0, 1, 2, 3, \dots\}$, die **natürlichen Zahlen**
 - Informatiker fangen bei 0 an zu zählen!
- $\mathbb{N}^{\geq 1} = \{1, 2, 3, \dots\}$, die **positiven natürlichen Zahlen**
- $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$, die **ganzen Zahlen**
- $\mathbb{Q} = \{\frac{p}{q} \mid p \in \mathbb{Z}, q \in \mathbb{N}^{\geq 1}\}$ (die **rationalen Zahlen**)
- $\mathbb{R}$, die **reellen Zahlen**
 - alle Zahlen auf der Zahlengerade
(keine „Löcher“ wie bei $\mathbb{Q}$)

- $\emptyset = \{\}$, die **leere Menge**
 - enthält kein Element
 - es gilt: $\emptyset \subseteq M$ für alle Mengen M
- $\mathbb{N} = \{0, 1, 2, 3, \dots\}$, die **natürlichen Zahlen**
 - Informatiker fangen bei 0 an zu zählen!
- $\mathbb{N}^{\geq 1} = \{1, 2, 3, \dots\}$, die **positiven natürlichen Zahlen**
- $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$, die **ganzen Zahlen**
- $\mathbb{Q} = \{\frac{p}{q} \mid p \in \mathbb{Z}, q \in \mathbb{N}^{\geq 1}\}$ (die **rationalen Zahlen**)
- $\mathbb{R}$, die **reellen Zahlen**
 - alle Zahlen auf der Zahlengerade (keine „Löcher“ wie bei $\mathbb{Q}$)
 - enthält auch π , e , $\sqrt{2}$, ...

- $\emptyset = \{\}$, die **leere Menge**
 - enthält kein Element
 - es gilt: $\emptyset \subseteq M$ für alle Mengen M
- $\mathbb{N} = \{0, 1, 2, 3, \dots\}$, die **natürlichen Zahlen**
 - Informatiker fangen bei 0 an zu zählen!
- $\mathbb{N}^{\geq 1} = \{1, 2, 3, \dots\}$, die **positiven natürlichen Zahlen**
- $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$, die **ganzen Zahlen**
- $\mathbb{Q} = \{\frac{p}{q} \mid p \in \mathbb{Z}, q \in \mathbb{N}^{\geq 1}\}$ (die **rationalen Zahlen**)
- $\mathbb{R}$, die **reellen Zahlen**
 - alle Zahlen auf der Zahlengerade (keine „Löcher“ wie bei $\mathbb{Q}$)
 - enthält auch π , e , $\sqrt{2}$, ...
- $\mathbb{B} = \{0, 1\}$, die **Boole'sche Menge**

Übung 2.6

Geben Sie formale Beschreibungen für die folgenden Teilmengen der natürlichen Zahlen:

- 1 Alle geraden Zahlen
- 2 Alle Quadratzahlen
- 3 Alle Zahlen, die eine gerade Anzahl von Teilern haben

1. Einführung

2. Mengen

2.1 Grundbegriffe

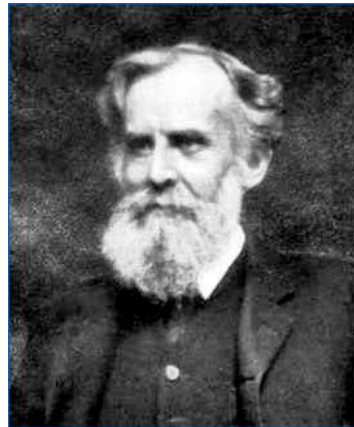
2.2 Mengenoperationen

3. Aussagenlogik

4. Prädikatenlogik

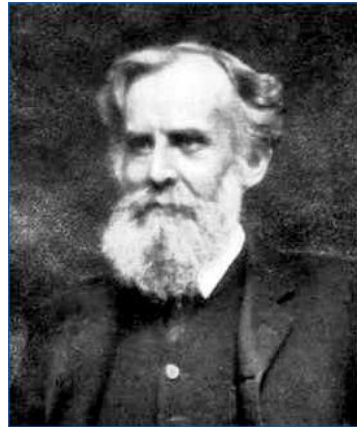
5. Prolog

- Grafische Mengendarstellung



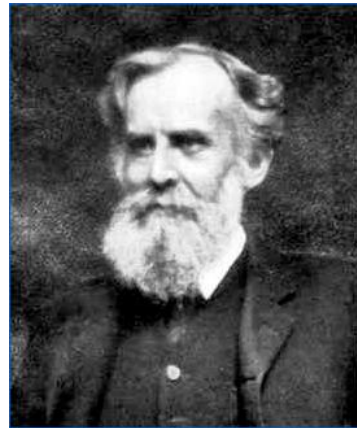
John Venn
(1834–1923)

- Grafische Mengendarstellung
 - Mengen sind zusammenhängende Flächen



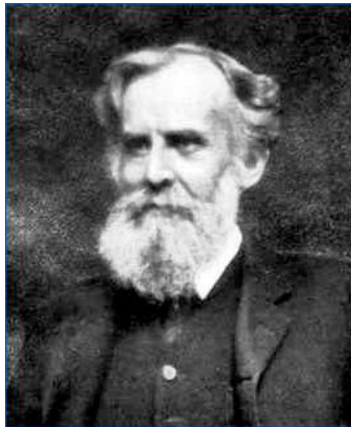
John Venn
(1834–1923)

- Grafische Mengendarstellung
 - Mengen sind zusammenhängende Flächen
 - Überlappungen visualisieren gemeinsame Elemente



John Venn
(1834–1923)

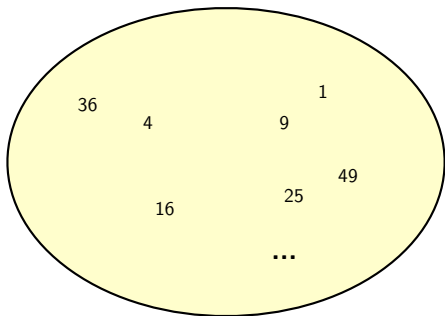
- Grafische Mengendarstellung
 - Mengen sind zusammenhängende Flächen
 - Überlappungen visualisieren gemeinsame Elemente
- Zeigen alle möglichen Beziehungen



John Venn
(1834–1923)

- Grafische Mengendarstellung
 - Mengen sind zusammenhängende Flächen
 - Überlappungen visualisieren gemeinsame Elemente
- Zeigen alle möglichen Beziehungen

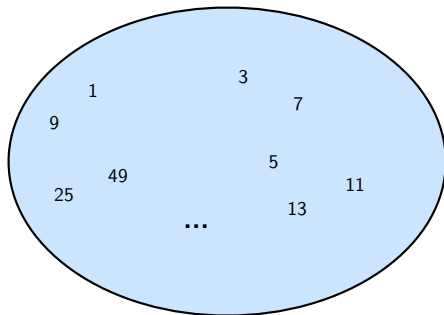
Quadratzahlen



John Venn
(1834–1923)

- Grafische Mengendarstellung
 - Mengen sind zusammenhängende Flächen
 - Überlappungen visualisieren gemeinsame Elemente
- Zeigen alle möglichen Beziehungen

Ungerade Zahlen

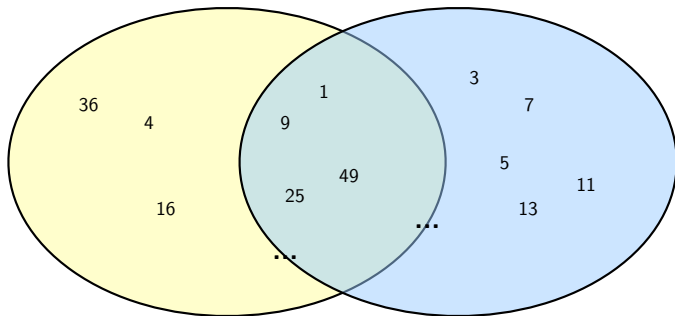


John Venn
(1834–1923)

- Grafische Mengendarstellung
 - Mengen sind zusammenhängende Flächen
 - Überlappungen visualisieren gemeinsame Elemente
- Zeigen alle möglichen Beziehungen

Quadratzahlen

Ungerade Zahlen



John Venn
(1834–1923)

Annahme: Alle betrachteten Mengen sind Teilmengen einer gemeinsamen **Trägermenge** T .

Wichtige Mengenoperationen sind:

- Vereinigung

Annahme: Alle betrachteten Mengen sind Teilmengen einer gemeinsamen **Trägermenge** T .

Wichtige Mengenoperationen sind:

- Vereinigung
- Schnitt

Annahme: Alle betrachteten Mengen sind Teilmengen einer gemeinsamen **Trägermenge** T .

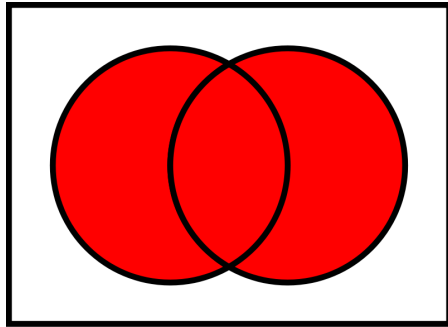
Wichtige Mengenoperationen sind:

- Vereinigung
- Schnitt
- Differenz

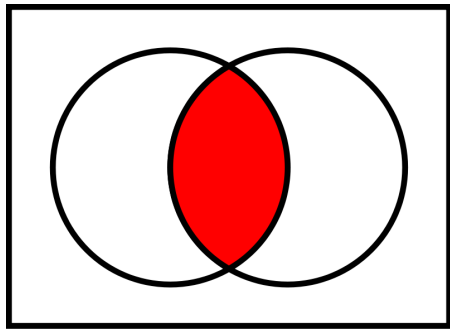
Annahme: Alle betrachteten Mengen sind Teilmengen einer gemeinsamen **Trägermenge** T .

Wichtige Mengenoperationen sind:

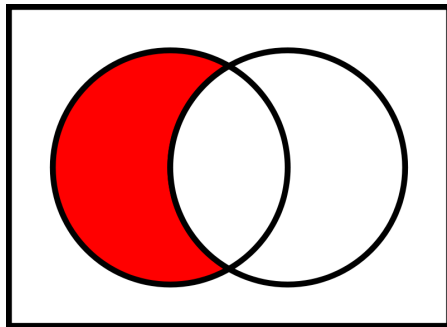
- Vereinigung
- Schnitt
- Differenz
- Komplement



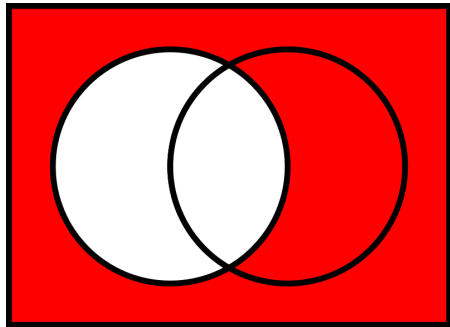
- $M_1 \cup M_2 = \{x \mid x \in M_1 \text{ oder } x \in M_2\}$
- $x \in M_1 \cup M_2$ gdw. $x \in M_1$ oder $x \in M_2$



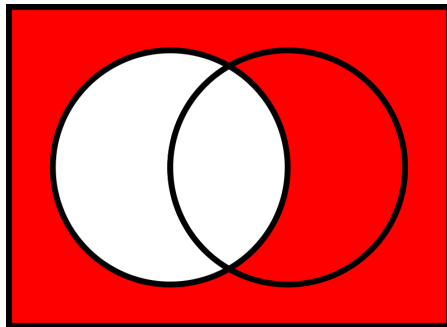
- $M_1 \cap M_2 = \{x \mid x \in M_1 \text{ und } x \in M_2\}$
- $x \in M_1 \cap M_2$ gdw. $x \in M_1$ und $x \in M_2$



- $M_1 \setminus M_2 = \{x \mid x \in M_1 \text{ und } x \notin M_2\}$
- $x \in M_1 \setminus M_2$ gdw. $x \in M_1$ und $x \notin M_2$



- $\overline{M_1} = \{x \mid x \notin M_1\}$
- $x \in \overline{M_1}$ gdw. $x \notin M_1$



- $\overline{M_1} = \{x \mid x \notin M_1\}$
- $x \in \overline{M_1}$ gdw. $x \notin M_1$

Achtung

Hier ist die implizite Annahme der Trägermenge T besonders wichtig.

Übung 2.7

- 1 Sei $T = \{1, 2, \dots, 12\}$,
 $M_1 = \{1, 2, 3, 4, 5, 6, 7, 8\}$, $M_2 = \{2, 4, 6, 8, 10, 12\}$.

Berechnen Sie die folgenden Mengen und visualisieren Sie diese in einem Venn-Diagramm.

- a $M_1 \cup M_2$
- b $M_1 \cap M_2$
- c $M_1 \setminus M_2$
- d $\overline{M_1}$
- e $\overline{M_2}$

- 2 Sei $T = \mathbb{N}$, $M_1 = \{3i \mid i \in \mathbb{N}\}$, $M_2 = \{2i + 1 \mid i \in \mathbb{N}\}$.

Geben Sie für die folgenden Mengen jeweils eine mathematische und eine umgangssprachliche Charakterisierung an.

- a $M_1 \cap M_2$
- b $M_1 \setminus M_2$
- c $\overline{M_1 \setminus M_2}$

Lösungen 2.7

- 2 Sei $T = \mathbb{N}$, $M_1 = \{3i \mid i \in \mathbb{N}\}$, $M_2 = \{2i + 1, i \in \mathbb{N}\}$. Berechnen Sie die folgenden Mengen. Geben Sie jeweils eine mathematische und eine umgangssprachliche Charakterisierung des Ergebnisses an.

a $M_1 \cap M_2$

Lösungen 2.7

- 2 Sei $T = \mathbb{N}$, $M_1 = \{3i \mid i \in \mathbb{N}\}$, $M_2 = \{2i + 1, i \in \mathbb{N}\}$. Berechnen Sie die folgenden Mengen. Geben Sie jeweils eine mathematische und eine umgangssprachliche Charakterisierung des Ergebnisses an.

a $M_1 \cap M_2$

- Die Menge der ungeraden Vielfachen von 3

Lösungen 2.7

2 Sei $T = \mathbb{N}$, $M_1 = \{3i \mid i \in \mathbb{N}\}$, $M_2 = \{2i + 1, i \in \mathbb{N}\}$. Berechnen Sie die folgenden Mengen. Geben Sie jeweils eine mathematische und eine umgangssprachliche Charakterisierung des Ergebnisses an.

a $M_1 \cap M_2$

- Die Menge der ungeraden Vielfachen von 3
- $M_1 \cap M_2 = \{6i + 3 \mid i \in \mathbb{N}\}$

Lösungen 2.7

2 Sei $T = \mathbb{N}$, $M_1 = \{3i \mid i \in \mathbb{N}\}$, $M_2 = \{2i + 1, i \in \mathbb{N}\}$. Berechnen Sie die folgenden Mengen. Geben Sie jeweils eine mathematische und eine umgangssprachliche Charakterisierung des Ergebnisses an.

a $M_1 \cap M_2$

■ Die Menge der ungeraden Vielfachen von 3

■ $M_1 \cap M_2 = \{6i + 3 \mid i \in \mathbb{N}\}$

b $M_1 \setminus M_2$

Lösungen 2.7

2 Sei $T = \mathbb{N}$, $M_1 = \{3i \mid i \in \mathbb{N}\}$, $M_2 = \{2i + 1, i \in \mathbb{N}\}$. Berechnen Sie die folgenden Mengen. Geben Sie jeweils eine mathematische und eine umgangssprachliche Charakterisierung des Ergebnisses an.

a $M_1 \cap M_2$

- Die Menge der ungeraden Vielfachen von 3

- $M_1 \cap M_2 = \{6i + 3 \mid i \in \mathbb{N}\}$

b $M_1 \setminus M_2$

- Die Menge der geraden Vielfachen von 3

Lösungen 2.7

2 Sei $T = \mathbb{N}$, $M_1 = \{3i \mid i \in \mathbb{N}\}$, $M_2 = \{2i + 1, i \in \mathbb{N}\}$. Berechnen Sie die folgenden Mengen. Geben Sie jeweils eine mathematische und eine umgangssprachliche Charakterisierung des Ergebnisses an.

a $M_1 \cap M_2$

- Die Menge der ungeraden Vielfachen von 3
- $M_1 \cap M_2 = \{6i + 3 \mid i \in \mathbb{N}\}$

b $M_1 \setminus M_2$

- Die Menge der geraden Vielfachen von 3
- $M_1 \setminus M_2 = \{6i \mid i \in \mathbb{N}\}$

Lösungen 2.7

2 Sei $T = \mathbb{N}$, $M_1 = \{3i \mid i \in \mathbb{N}\}$, $M_2 = \{2i + 1, i \in \mathbb{N}\}$. Berechnen Sie die folgenden Mengen. Geben Sie jeweils eine mathematische und eine umgangssprachliche Charakterisierung des Ergebnisses an.

a $M_1 \cap M_2$

■ Die Menge der ungeraden Vielfachen von 3

■ $M_1 \cap M_2 = \{6i + 3 \mid i \in \mathbb{N}\}$

b $M_1 \setminus M_2$

■ Die Menge der geraden Vielfachen von 3

■ $M_1 \setminus M_2 = \{6i \mid i \in \mathbb{N}\}$

c $M_1 \setminus \overline{M_2}$

Lösungen 2.7

2 Sei $T = \mathbb{N}$, $M_1 = \{3i \mid i \in \mathbb{N}\}$, $M_2 = \{2i + 1, i \in \mathbb{N}\}$. Berechnen Sie die folgenden Mengen. Geben Sie jeweils eine mathematische und eine umgangssprachliche Charakterisierung des Ergebnisses an.

a $M_1 \cap M_2$

■ Die Menge der ungeraden Vielfachen von 3

■ $M_1 \cap M_2 = \{6i + 3 \mid i \in \mathbb{N}\}$

b $M_1 \setminus M_2$

■ Die Menge der geraden Vielfachen von 3

■ $M_1 \setminus M_2 = \{6i \mid i \in \mathbb{N}\}$

c $M_1 \setminus \overline{M_2}$

■ Siehe $M_1 \cap M_2$

Satz 2.8

Es seien M , P und S Teilmengen einer Trägermenge T . Dann gelten die folgenden Gleichungen:

- Kommutativgesetze

Satz 2.8

Es seien M , P und S Teilmengen einer Trägermenge T . Dann gelten die folgenden Gleichungen:

- Kommutativgesetze
 - $M \cup P = P \cup M$

Satz 2.8

Es seien M , P und S Teilmengen einer Trägermenge T . Dann gelten die folgenden Gleichungen:

- Kommutativgesetze

- $M \cup P = P \cup M$

- $M \cap P = P \cap M$

Satz 2.8

Es seien M , P und S Teilmengen einer Trägermenge T . Dann gelten die folgenden Gleichungen:

- Kommutativgesetze
 - $M \cup P = P \cup M$
 - $M \cap P = P \cap M$
- Assoziativgesetze

Satz 2.8

Es seien M , P und S Teilmengen einer Trägermenge T . Dann gelten die folgenden Gleichungen:

- Kommutativgesetze

- $M \cup P = P \cup M$

- $M \cap P = P \cap M$

- Assoziativgesetze

- $M \cup (P \cup S) = (M \cup P) \cup S$

Satz 2.8

Es seien M , P und S Teilmengen einer Trägermenge T . Dann gelten die folgenden Gleichungen:

- Kommutativgesetze

- $M \cup P = P \cup M$

- $M \cap P = P \cap M$

- Assoziativgesetze

- $M \cup (P \cup S) = (M \cup P) \cup S$

- $M \cap (P \cap S) = (M \cap P) \cap S$

Satz 2.8

Es seien M , P und S Teilmengen einer Trägermenge T . Dann gelten die folgenden Gleichungen:

- Kommutativgesetze

- $M \cup P = P \cup M$

- $M \cap P = P \cap M$

- Assoziativgesetze

- $M \cup (P \cup S) = (M \cup P) \cup S$

- $M \cap (P \cap S) = (M \cap P) \cap S$

- Distributivgesetze

Satz 2.8

Es seien M , P und S Teilmengen einer Trägermenge T . Dann gelten die folgenden Gleichungen:

- Kommutativgesetze

- $M \cup P = P \cup M$

- $M \cap P = P \cap M$

- Assoziativgesetze

- $M \cup (P \cup S) = (M \cup P) \cup S$

- $M \cap (P \cap S) = (M \cap P) \cap S$

- Distributivgesetze

- $M \cup (P \cap S) = (M \cup P) \cap (M \cup S)$

Satz 2.8

Es seien M , P und S Teilmengen einer Trägermenge T . Dann gelten die folgenden Gleichungen:

- Kommutativgesetze

- $M \cup P = P \cup M$

- $M \cap P = P \cap M$

- Assoziativgesetze

- $M \cup (P \cup S) = (M \cup P) \cup S$

- $M \cap (P \cap S) = (M \cap P) \cap S$

- Distributivgesetze

- $M \cup (P \cap S) = (M \cup P) \cap (M \cup S)$

- $M \cap (P \cup S) = (M \cap P) \cup (M \cap S)$

Satz 2.8

Es seien M , P und S Teilmengen einer Trägermenge T . Dann gelten die folgenden Gleichungen:

- Kommutativgesetze

- $M \cup P = P \cup M$

- $M \cap P = P \cap M$

- Assoziativgesetze

- $M \cup (P \cup S) = (M \cup P) \cup S$

- $M \cap (P \cap S) = (M \cap P) \cap S$

- Distributivgesetze

- $M \cup (P \cap S) = (M \cup P) \cap (M \cup S)$

- $M \cap (P \cup S) = (M \cap P) \cup (M \cap S)$

- Gesetze von De Morgan



Augustus
De Morgan
(1806-1871)

Satz 2.8

Es seien M , P und S Teilmengen einer Trägermenge T . Dann gelten die folgenden Gleichungen:

- Kommutativgesetze

- $M \cup P = P \cup M$

- $M \cap P = P \cap M$

- Assoziativgesetze

- $M \cup (P \cup S) = (M \cup P) \cup S$

- $M \cap (P \cap S) = (M \cap P) \cap S$

- Distributivgesetze

- $M \cup (P \cap S) = (M \cup P) \cap (M \cup S)$

- $M \cap (P \cup S) = (M \cap P) \cup (M \cap S)$

- Gesetze von De Morgan

- $\overline{M \cup P} = \overline{M} \cap \overline{P}$



Augustus
De Morgan
(1806-1871)

Satz 2.8

Es seien M , P und S Teilmengen einer Trägermenge T . Dann gelten die folgenden Gleichungen:

- Kommutativgesetze

- $M \cup P = P \cup M$

- $M \cap P = P \cap M$

- Assoziativgesetze

- $M \cup (P \cup S) = (M \cup P) \cup S$

- $M \cap (P \cap S) = (M \cap P) \cap S$

- Distributivgesetze

- $M \cup (P \cap S) = (M \cup P) \cap (M \cup S)$

- $M \cap (P \cup S) = (M \cap P) \cup (M \cap S)$

- Gesetze von De Morgan

- $\overline{M \cup P} = \overline{M} \cap \overline{P}$

- $\overline{M \cap P} = \overline{M} \cup \overline{P}$



Augustus
De Morgan
(1806-1871)

Satz 2.8

Es seien M , P und S Teilmengen einer Trägermenge T . Dann gelten die folgenden Gleichungen:

■ Kommutativgesetze

$$\blacksquare M \cup P = P \cup M$$

$$\blacksquare M \cap P = P \cap M$$

■ Assoziativgesetze

$$\blacksquare M \cup (P \cup S) = (M \cup P) \cup S$$

$$\blacksquare M \cap (P \cap S) = (M \cap P) \cap S$$

■ Distributivgesetze

$$\blacksquare M \cup (P \cap S) = (M \cup P) \cap (M \cup S)$$

$$\blacksquare M \cap (P \cup S) = (M \cap P) \cup (M \cap S)$$

■ Gesetze von De Morgan

$$\blacksquare \overline{M \cup P} = \overline{M} \cap \overline{P}$$

$$\blacksquare \overline{M \cap P} = \overline{M} \cup \overline{P}$$

■ Neutrale Elemente



Augustus
De Morgan
(1806-1871)

Satz 2.8

Es seien M , P und S Teilmengen einer Trägermenge T . Dann gelten die folgenden Gleichungen:

■ Kommutativgesetze

- $M \cup P = P \cup M$

- $M \cap P = P \cap M$

■ Assoziativgesetze

- $M \cup (P \cup S) = (M \cup P) \cup S$

- $M \cap (P \cap S) = (M \cap P) \cap S$

■ Distributivgesetze

- $M \cup (P \cap S) = (M \cup P) \cap (M \cup S)$

- $M \cap (P \cup S) = (M \cap P) \cup (M \cap S)$

■ Gesetze von De Morgan

- $\overline{M \cup P} = \overline{M} \cap \overline{P}$

- $\overline{M \cap P} = \overline{M} \cup \overline{P}$

■ Neutrale Elemente

- $M \cup \emptyset = M$



Augustus
De Morgan
(1806-1871)

Satz 2.8

Es seien M , P und S Teilmengen einer Trägermenge T . Dann gelten die folgenden Gleichungen:

■ Kommutativgesetze

$$\blacksquare M \cup P = P \cup M$$

$$\blacksquare M \cap P = P \cap M$$

■ Assoziativgesetze

$$\blacksquare M \cup (P \cup S) = (M \cup P) \cup S$$

$$\blacksquare M \cap (P \cap S) = (M \cap P) \cap S$$

■ Distributivgesetze

$$\blacksquare M \cup (P \cap S) = (M \cup P) \cap (M \cup S)$$

$$\blacksquare M \cap (P \cup S) = (M \cap P) \cup (M \cap S)$$

■ Gesetze von De Morgan

$$\blacksquare \overline{M \cup P} = \overline{M} \cap \overline{P}$$

$$\blacksquare \overline{M \cap P} = \overline{M} \cup \overline{P}$$

■ Neutrale Elemente

$$\blacksquare M \cup \emptyset = M$$

$$\blacksquare M \cap T = M$$



Augustus
De Morgan
(1806-1871)

Satz 2.8

Es seien M , P und S Teilmengen einer Trägermenge T . Dann gelten die folgenden Gleichungen:

■ Kommutativgesetze

- $M \cup P = P \cup M$

- $M \cap P = P \cap M$

■ Assoziativgesetze

- $M \cup (P \cup S) = (M \cup P) \cup S$

- $M \cap (P \cap S) = (M \cap P) \cap S$

■ Distributivgesetze

- $M \cup (P \cap S) = (M \cup P) \cap (M \cup S)$

- $M \cap (P \cup S) = (M \cap P) \cup (M \cap S)$

■ Gesetze von De Morgan

- $\overline{M \cup P} = \overline{M} \cap \overline{P}$

- $\overline{M \cap P} = \overline{M} \cup \overline{P}$

■ Neutrale Elemente

- $M \cup \emptyset = M$

- $M \cap T = M$

■ Inverse Elemente



Augustus
De Morgan
(1806-1871)

Satz 2.8

Es seien M , P und S Teilmengen einer Trägermenge T . Dann gelten die folgenden Gleichungen:

■ Kommutativgesetze

- $M \cup P = P \cup M$

- $M \cap P = P \cap M$

■ Assoziativgesetze

- $M \cup (P \cup S) = (M \cup P) \cup S$

- $M \cap (P \cap S) = (M \cap P) \cap S$

■ Distributivgesetze

- $M \cup (P \cap S) = (M \cup P) \cap (M \cup S)$

- $M \cap (P \cup S) = (M \cap P) \cup (M \cap S)$

■ Gesetze von De Morgan

- $\overline{M \cup P} = \overline{M} \cap \overline{P}$

- $\overline{M \cap P} = \overline{M} \cup \overline{P}$

■ Neutrale Elemente

- $M \cup \emptyset = M$

- $M \cap T = M$

■ Inverse Elemente

- $M \cup \overline{M} = T$



Augustus
De Morgan
(1806-1871)

Satz 2.8

Es seien M , P und S Teilmengen einer Trägermenge T . Dann gelten die folgenden Gleichungen:

■ Kommutativgesetze

- $M \cup P = P \cup M$
- $M \cap P = P \cap M$

■ Assoziativgesetze

- $M \cup (P \cup S) = (M \cup P) \cup S$
- $M \cap (P \cap S) = (M \cap P) \cap S$

■ Distributivgesetze

- $M \cup (P \cap S) = (M \cup P) \cap (M \cup S)$
- $M \cap (P \cup S) = (M \cap P) \cup (M \cap S)$

■ Gesetze von De Morgan

- $\overline{M \cup P} = \overline{M} \cap \overline{P}$
- $\overline{M \cap P} = \overline{M} \cup \overline{P}$

■ Neutrale Elemente

- $M \cup \emptyset = M$
- $M \cap T = M$

■ Inverse Elemente

- $M \cup \overline{M} = T$
- $M \cap \overline{M} = \emptyset$



Augustus
De Morgan
(1806-1871)

Satz 2.8

Es seien M , P und S Teilmengen einer Trägermenge T . Dann gelten die folgenden Gleichungen:

■ Kommutativgesetze

- $M \cup P = P \cup M$
- $M \cap P = P \cap M$

■ Assoziativgesetze

- $M \cup (P \cup S) = (M \cup P) \cup S$
- $M \cap (P \cap S) = (M \cap P) \cap S$

■ Distributivgesetze

- $M \cup (P \cap S) = (M \cup P) \cap (M \cup S)$
- $M \cap (P \cup S) = (M \cap P) \cup (M \cap S)$

■ Gesetze von De Morgan

- $\overline{M \cup P} = \overline{M} \cap \overline{P}$
- $\overline{M \cap P} = \overline{M} \cup \overline{P}$

■ Neutrale Elemente

- $M \cup \emptyset = M$
- $M \cap T = M$

■ Inverse Elemente

- $M \cup \overline{M} = T$
- $M \cap \overline{M} = \emptyset$

■ Absorbierende Elemente



Augustus
De Morgan
(1806-1871)

Satz 2.8

Es seien M , P und S Teilmengen einer Trägermenge T . Dann gelten die folgenden Gleichungen:

■ Kommutativgesetze

$$\blacksquare M \cup P = P \cup M$$

$$\blacksquare M \cap P = P \cap M$$

■ Assoziativgesetze

$$\blacksquare M \cup (P \cup S) = (M \cup P) \cup S$$

$$\blacksquare M \cap (P \cap S) = (M \cap P) \cap S$$

■ Distributivgesetze

$$\blacksquare M \cup (P \cap S) = (M \cup P) \cap (M \cup S)$$

$$\blacksquare M \cap (P \cup S) = (M \cap P) \cup (M \cap S)$$

■ Gesetze von De Morgan

$$\blacksquare \overline{M \cup P} = \overline{M} \cap \overline{P}$$

$$\blacksquare \overline{M \cap P} = \overline{M} \cup \overline{P}$$

■ Neutrale Elemente

$$\blacksquare M \cup \emptyset = M$$

$$\blacksquare M \cap T = M$$

■ Inverse Elemente

$$\blacksquare M \cup \overline{M} = T$$

$$\blacksquare M \cap \overline{M} = \emptyset$$

■ Absorbierende Elemente

$$\blacksquare M \cup T = T$$



Augustus
De Morgan
(1806-1871)

Satz 2.8

Es seien M , P und S Teilmengen einer Trägermenge T . Dann gelten die folgenden Gleichungen:

■ Kommutativgesetze

- $M \cup P = P \cup M$
- $M \cap P = P \cap M$

■ Assoziativgesetze

- $M \cup (P \cup S) = (M \cup P) \cup S$
- $M \cap (P \cap S) = (M \cap P) \cap S$

■ Distributivgesetze

- $M \cup (P \cap S) = (M \cup P) \cap (M \cup S)$
- $M \cap (P \cup S) = (M \cap P) \cup (M \cap S)$

■ Gesetze von De Morgan

- $\overline{M \cup P} = \overline{M} \cap \overline{P}$
- $\overline{M \cap P} = \overline{M} \cup \overline{P}$

■ Neutrale Elemente

- $M \cup \emptyset = M$
- $M \cap T = M$

■ Inverse Elemente

- $M \cup \overline{M} = T$
- $M \cap \overline{M} = \emptyset$

■ Absorbierende Elemente

- $M \cup T = T$
- $M \cap \emptyset = \emptyset$



Augustus
De Morgan
(1806-1871)

Satz 2.8

Es seien M , P und S Teilmengen einer Trägermenge T . Dann gelten die folgenden Gleichungen:

■ Kommutativgesetze

- $M \cup P = P \cup M$
- $M \cap P = P \cap M$

■ Assoziativgesetze

- $M \cup (P \cup S) = (M \cup P) \cup S$
- $M \cap (P \cap S) = (M \cap P) \cap S$

■ Distributivgesetze

- $M \cup (P \cap S) = (M \cup P) \cap (M \cup S)$
- $M \cap (P \cup S) = (M \cap P) \cup (M \cap S)$

■ Gesetze von De Morgan

- $\overline{M \cup P} = \overline{M} \cap \overline{P}$
- $\overline{M \cap P} = \overline{M} \cup \overline{P}$

■ Neutrale Elemente

- $M \cup \emptyset = M$
- $M \cap T = M$

■ Inverse Elemente

- $M \cup \overline{M} = T$
- $M \cap \overline{M} = \emptyset$

■ Absorbierende Elemente

- $M \cup T = T$
- $M \cap \emptyset = \emptyset$

■ Idempotenz



Augustus
De Morgan
(1806-1871)

Satz 2.8

Es seien M, P und S Teilmengen einer Trägermenge T . Dann gelten die folgenden Gleichungen:

■ Kommutativgesetze

- $M \cup P = P \cup M$
- $M \cap P = P \cap M$

■ Assoziativgesetze

- $M \cup (P \cup S) = (M \cup P) \cup S$
- $M \cap (P \cap S) = (M \cap P) \cap S$

■ Distributivgesetze

- $M \cup (P \cap S) = (M \cup P) \cap (M \cup S)$
- $M \cap (P \cup S) = (M \cap P) \cup (M \cap S)$

■ Gesetze von De Morgan

- $\overline{M \cup P} = \overline{M} \cap \overline{P}$
- $\overline{M \cap P} = \overline{M} \cup \overline{P}$

■ Neutrale Elemente

- $M \cup \emptyset = M$
- $M \cap T = M$

■ Inverse Elemente

- $M \cup \overline{M} = T$
- $M \cap \overline{M} = \emptyset$

■ Absorbierende Elemente

- $M \cup T = T$
- $M \cap \emptyset = \emptyset$

■ Idempotenz

- $M \cup M = M$



Augustus
De Morgan
(1806-1871)

Satz 2.8

Es seien M, P und S Teilmengen einer Trägermenge T . Dann gelten die folgenden Gleichungen:

■ Kommutativgesetze

- $M \cup P = P \cup M$
- $M \cap P = P \cap M$

■ Assoziativgesetze

- $M \cup (P \cup S) = (M \cup P) \cup S$
- $M \cap (P \cap S) = (M \cap P) \cap S$

■ Distributivgesetze

- $M \cup (P \cap S) = (M \cup P) \cap (M \cup S)$
- $M \cap (P \cup S) = (M \cap P) \cup (M \cap S)$

■ Gesetze von De Morgan

- $\overline{M \cup P} = \overline{M} \cap \overline{P}$
- $\overline{M \cap P} = \overline{M} \cup \overline{P}$

■ Neutrale Elemente

- $M \cup \emptyset = M$
- $M \cap T = M$

■ Inverse Elemente

- $M \cup \overline{M} = T$
- $M \cap \overline{M} = \emptyset$

■ Absorbierende Elemente

- $M \cup T = T$
- $M \cap \emptyset = \emptyset$

■ Idempotenz

- $M \cup M = M$
- $M \cap M = M$



Augustus
De Morgan
(1806-1871)

Satz 2.8

Es seien M, P und S Teilmengen einer Trägermenge T . Dann gelten die folgenden Gleichungen:

■ Kommutativgesetze

- $M \cup P = P \cup M$
- $M \cap P = P \cap M$

■ Assoziativgesetze

- $M \cup (P \cup S) = (M \cup P) \cup S$
- $M \cap (P \cap S) = (M \cap P) \cap S$

■ Distributivgesetze

- $M \cup (P \cap S) = (M \cup P) \cap (M \cup S)$
- $M \cap (P \cup S) = (M \cap P) \cup (M \cap S)$

■ Gesetze von De Morgan

- $\overline{M \cup P} = \overline{M} \cap \overline{P}$
- $\overline{M \cap P} = \overline{M} \cup \overline{P}$

■ Neutrale Elemente

- $M \cup \emptyset = M$
- $M \cap T = M$

■ Inverse Elemente

- $M \cup \overline{M} = T$
- $M \cap \overline{M} = \emptyset$

■ Absorbierende Elemente

- $M \cup T = T$
- $M \cap \emptyset = \emptyset$

■ Idempotenz

- $M \cup M = M$
- $M \cap M = M$

■ Doppeltes Komplement



Augustus
De Morgan
(1806-1871)

Satz 2.8

Es seien M, P und S Teilmengen einer Trägermenge T . Dann gelten die folgenden Gleichungen:

■ Kommutativgesetze

- $M \cup P = P \cup M$

- $M \cap P = P \cap M$

■ Assoziativgesetze

- $M \cup (P \cup S) = (M \cup P) \cup S$

- $M \cap (P \cap S) = (M \cap P) \cap S$

■ Distributivgesetze

- $M \cup (P \cap S) = (M \cup P) \cap (M \cup S)$

- $M \cap (P \cup S) = (M \cap P) \cup (M \cap S)$

■ Gesetze von De Morgan

- $\overline{M \cup P} = \overline{M} \cap \overline{P}$

- $\overline{M \cap P} = \overline{M} \cup \overline{P}$

■ Neutrale Elemente

- $M \cup \emptyset = M$

- $M \cap T = M$

■ Inverse Elemente

- $M \cup \overline{M} = T$

- $M \cap \overline{M} = \emptyset$

■ Absorbierende Elemente

- $M \cup T = T$

- $M \cap \emptyset = \emptyset$

■ Idempotenz

- $M \cup M = M$

- $M \cap M = M$

■ Doppeltes Komplement

- $\overline{\overline{M}} = M$



Augustus
De Morgan
(1806-1871)

Menge Zusammenfassung von Elementen zu einem Ganzen

Menge Zusammenfassung von Elementen zu einem Ganzen

Operationen Vereinigung, Schnitt, Differenz, Komplement

Menge Zusammenfassung von Elementen zu einem Ganzen

Operationen Vereinigung, Schnitt, Differenz, Komplement

Visualisierung Venn-Diagramme

Menge Zusammenfassung von Elementen zu einem Ganzen

Operationen Vereinigung, Schnitt, Differenz, Komplement

Visualisierung Venn-Diagramme

Rechenregeln De Morgan, neutrale/inverse/absorbierende Elemente

1. Einführung

2. Mengen

3. Aussagenlogik

3.1 Syntax

3.2 Semantik

3.3 Schlussfolgerungsverfahren

4. Prädikatenlogik

5. Prolog

Einfachste in dieser Vorlesung betrachtete Logik

- begrenzte Ausdruckstärke

Einfachste in dieser Vorlesung betrachtete Logik

- begrenzte Ausdrucksstärke
- einfach zu verstehen und anzuwenden

Einfachste in dieser Vorlesung betrachtete Logik

- begrenzte Ausdruckstärke
- einfach zu verstehen und anzuwenden
- enthält verschiedene Konzepte, die auch in ausdrucksstärkeren Logiken vorkommen

Einfachste in dieser Vorlesung betrachtete Logik

- begrenzte Ausdruckstärke
- einfach zu verstehen und anzuwenden
- enthält verschiedene Konzepte, die auch in ausdrucksstärkeren Logiken vorkommen

Einfachste in dieser Vorlesung betrachtete Logik

- begrenzte Ausdruckstärke
- einfach zu verstehen und anzuwenden
- enthält verschiedene Konzepte, die auch in ausdrucksstärkeren Logiken vorkommen

Aussagenvariablen repräsentieren atomare Aussagen

Beispiel 3.1

$A \rightsquigarrow$ „Es regnet.“

Einfachste in dieser Vorlesung betrachtete Logik

- begrenzte Ausdrucksstärke
- einfach zu verstehen und anzuwenden
- enthält verschiedene Konzepte, die auch in ausdrucksstärkeren Logiken vorkommen

Aussagenvariablen repräsentieren atomare Aussagen

Beispiel 3.1

$A \rightsquigarrow$ „Es regnet.“

Junktoren repräsentieren Zusammenhänge zwischen Aussagen

Beispiel 3.2

$A \rightarrow B \rightsquigarrow$ „Wenn es regnet, wird die Erde nass.“

- werden durch Großbuchstaben repräsentiert $A, B, X, Y, \dots$

- werden durch Großbuchstaben repräsentiert $A, B, X, Y, \dots$
- drücken Aussagen aus, die **wahr** oder **falsch** sein können

- werden durch Großbuchstaben repräsentiert $A, B, X, Y, \dots$
- drücken Aussagen aus, die **wahr** oder **falsch** sein können
 - „Es regnet.“

- werden durch Großbuchstaben repräsentiert $A, B, X, Y, \dots$
- drücken Aussagen aus, die **wahr** oder **falsch** sein können
 - „Es regnet.“
 - „Paul liebt Paula.“

- werden durch Großbuchstaben repräsentiert $A, B, X, Y, \dots$
- drücken Aussagen aus, die **wahr** oder **falsch** sein können
 - „Es regnet.“
 - „Paul liebt Paula.“
 - „Sokrates ist sterblich.“

1. Einführung

2. Mengen

3. Aussagenlogik

3.1 Syntax

3.2 Semantik

3.3 Schlussfolgerungsverfahren

4. Prädikatenlogik

5. Prolog

Lehre von **wohlgeformten** Ausdrücken

- Linguistik: Satzbau; Regeln zum Erzeugen von **Sätzen** aus **Wörtern**
- Logik: Regeln zum Erzeugen von **Formeln** aus vorgegebenen Symbolen

Lehre von **wohlgeformten** Ausdrücken

- Linguistik: Satzbau; Regeln zum Erzeugen von **Sätzen** aus **Wörtern**
- Logik: Regeln zum Erzeugen von **Formeln** aus vorgegebenen Symbolen

Beispiel 3.3

	syntaktisch falsch	syntaktisch richtig
Deutsch	geben . hat Mensch ? die	Heute ist schönes Wetter.

Lehre von wohlgeformten Ausdrücken

- Linguistik: Satzbau; Regeln zum Erzeugen von Sätzen aus Wörtern
- Logik: Regeln zum Erzeugen von Formeln aus vorgegebenen Symbolen

Beispiel 3.3

	syntaktisch falsch	syntaktisch richtig
Deutsch	geben . hat Mensch ? die	Heute ist schönes Wetter.
Mathematik	$+) 2 - \cdot ((xy - 3 <$	$3 < (4 + 2) \cdot 5$

Definition 3.4 (Aussagenlogische Formel)

Sei V eine Menge von Aussagenvariablen.

- Jedes $X \in V$ ist eine aussagenlogische Formel.

Definition 3.4 (Aussagenlogische Formel)

Sei V eine Menge von Aussagenvariablen.

- Jedes $X \in V$ ist eine aussagenlogische Formel.
- **W** und **F** sind aussagenlogische Formeln.

Definition 3.4 (Aussagenlogische Formel)

Sei V eine Menge von Aussagenvariablen.

- Jedes $X \in V$ ist eine aussagenlogische Formel.
- **W** und **F** sind aussagenlogische Formeln.
- Wenn φ und ψ aussagenlogische Formeln sind, dann auch

Definition 3.4 (Aussagenlogische Formel)

Sei V eine Menge von Aussagenvariablen.

- Jedes $X \in V$ ist eine aussagenlogische Formel.
- **W** und **F** sind aussagenlogische Formeln.
- Wenn φ und ψ aussagenlogische Formeln sind, dann auch
 - $(\neg\varphi)$ (Negation)

Definition 3.4 (Aussagenlogische Formel)

Sei V eine Menge von Aussagenvariablen.

- Jedes $X \in V$ ist eine aussagenlogische Formel.
- **W** und **F** sind aussagenlogische Formeln.
- Wenn φ und ψ aussagenlogische Formeln sind, dann auch
 - $(\neg\varphi)$ (Negation)
 - $(\varphi \wedge \psi)$ (Konjunktion)

Definition 3.4 (Aussagenlogische Formel)

Sei V eine Menge von Aussagenvariablen.

- Jedes $X \in V$ ist eine aussagenlogische Formel.
- **W** und **F** sind aussagenlogische Formeln.
- Wenn φ und ψ aussagenlogische Formeln sind, dann auch
 - $(\neg\varphi)$ (Negation)
 - $(\varphi \wedge \psi)$ (Konjunktion)
 - $(\varphi \vee \psi)$ (Disjunktion)

Definition 3.4 (Aussagenlogische Formel)

Sei V eine Menge von Aussagenvariablen.

- Jedes $X \in V$ ist eine aussagenlogische Formel.
- **W** und **F** sind aussagenlogische Formeln.
- Wenn φ und ψ aussagenlogische Formeln sind, dann auch
 - $(\neg\varphi)$ (Negation)
 - $(\varphi \wedge \psi)$ (Konjunktion)
 - $(\varphi \vee \psi)$ (Disjunktion)
 - $(\varphi \rightarrow \psi)$ (materiale Implikation)

Definition 3.4 (Aussagenlogische Formel)

Sei V eine Menge von Aussagenvariablen.

- Jedes $X \in V$ ist eine aussagenlogische Formel.
- **W** und **F** sind aussagenlogische Formeln.
- Wenn φ und ψ aussagenlogische Formeln sind, dann auch
 - $(\neg\varphi)$ (Negation)
 - $(\varphi \wedge \psi)$ (Konjunktion)
 - $(\varphi \vee \psi)$ (Disjunktion)
 - $(\varphi \rightarrow \psi)$ (materiale Implikation)
 - $(\varphi \leftrightarrow \psi)$ (materiale Äquivalenz)

Definition 3.4 (Aussagenlogische Formel)

Sei V eine Menge von Aussagenvariablen.

- Jedes $X \in V$ ist eine aussagenlogische Formel.
- **W** und **F** sind aussagenlogische Formeln.
- Wenn φ und ψ aussagenlogische Formeln sind, dann auch
 - $(\neg\varphi)$ (Negation)
 - $(\varphi \wedge \psi)$ (Konjunktion)
 - $(\varphi \vee \psi)$ (Disjunktion)
 - $(\varphi \rightarrow \psi)$ (materiale Implikation)
 - $(\varphi \leftrightarrow \psi)$ (materiale Äquivalenz)

Beispiel 3.5 (Formeln über der Variablenmenge $\{A, B, C, D, E\}$)

■ $(A \wedge B)$	■ $(A \wedge (B \vee C))$	■ $(A \rightarrow ((B \wedge (C \leftrightarrow D)) \vee (D \rightarrow E)))$
■ $(\mathbf{F} \vee A)$	■ $(\neg(A \vee B))$	■ $(A \leftrightarrow (A \leftrightarrow D))$
■ $(\neg\mathbf{W})$	■ $(C \rightarrow (B \vee D))$	■ $((\neg(A \leftrightarrow B) \leftrightarrow (C \leftrightarrow D)) \rightarrow E)$

Übung 3.6

Seien A , B , C und D Aussagenvariablen.

Welche der folgenden Zeichenketten sind aussagenlogische Formeln?

1 $(A \rightarrow \mathbf{F})$

2 $(A \wedge (B \vee C))$

3 $(A \neg B)$

4 $((A \rightarrow C) \wedge (((\neg A) \rightarrow C) \rightarrow C))$

5 $((\forall B) \vee (C \wedge D))$

6 $(A \rightarrow (B \vee (\neg B))(C \vee (\neg C)))$

7 $A \wedge B \vee \neg C$

8 $(A \rightarrow A)$

9 $(A \wedge (\neg A))$

10 $(A \neg \wedge B)$

11 $((\neg A) \wedge B)$

12 $((\neg A) \wedge B)$

13 $(\neg(A \wedge B))$

14 $((A \leftrightarrow B) \leftrightarrow ((\neg A) \leftrightarrow (\neg B)))$

Um Klammern zu sparen, wird vereinbart:

- Äußerste Klammern um eine Formel können weggelassen werden.

Um Klammern zu sparen, wird vereinbart:

- Äußerste Klammern um eine Formel können weggelassen werden.
- Gleiche Junktoren werden links-assoziativ gelesen.

Um Klammern zu sparen, wird vereinbart:

- Äußerste Klammern um eine Formel können weggelassen werden.
- Gleiche Junktoren werden links-assoziativ gelesen.
- Vorrang unterschiedlicher Junktoren: $\neg$ $\gg$ $\wedge$ $\gg$ $\vee$ $\gg$ $\rightarrow$ $\gg$ $\leftrightarrow$

Um Klammern zu sparen, wird vereinbart:

- Äußerste Klammern um eine Formel können weggelassen werden.
- Gleiche Junktoren werden links-assoziativ gelesen.
- Vorrang unterschiedlicher Junktoren: $\neg$ $\gg$ $\wedge$ $\gg$ $\vee$ $\gg$ $\rightarrow$ $\gg$ $\leftrightarrow$

Um Klammern zu sparen, wird vereinbart:

- Äußerste Klammern um eine Formel können weggelassen werden.
- Gleiche Junktoren werden links-assoziativ gelesen.
- Vorrang unterschiedlicher Junktoren: $\neg$ $\gg$ $\wedge$ $\gg$ $\vee$ $\gg$ $\rightarrow$ $\gg$ $\leftrightarrow$

Beispiel 3.7

- $A \wedge B$ statt $(A \wedge B)$
- $A \rightarrow B \rightarrow C$ statt $((A \rightarrow B) \rightarrow C)$
- $A \wedge \neg B \rightarrow B \vee C$ statt $((A \wedge (\neg B)) \rightarrow (B \vee C))$

Übung 3.8

Entfernen Sie so viele Klammern wie möglich, ohne den Vorrang zu verändern.

1 $((A \wedge B) \vee ((C \wedge D) \rightarrow (A \vee C)))$

2 $((A \wedge ((B \vee C) \wedge D)) \rightarrow A) \vee C$

3 $(A \wedge (B \vee (C \wedge (D \rightarrow (A \vee C)))))$

- 1. Einführung
- 2. Mengen
- 3. Aussagenlogik
 - 3.1 Syntax
 - 3.2 Semantik**
 - 3.3 Schlussfolgerungsverfahren
- 4. Prädikatenlogik
- 5. Prolog

Lehre von der **Bedeutung** von Ausdrücken

- Voraussetzung: Ausdruck ist wohlgeformt (syntaktisch korrekt)

Lehre von der **Bedeutung** von Ausdrücken

- Voraussetzung: Ausdruck ist wohlgeformt (syntaktisch korrekt)
- Linguistik: Was bedeutet ein Begriff oder Satz?

Lehre von der **Bedeutung** von Ausdrücken

- Voraussetzung: Ausdruck ist wohlgeformt (syntaktisch korrekt)
- Linguistik: Was bedeutet ein Begriff oder Satz?
- Logik: Ist ein Satz wahr oder falsch?

Lehre von der **Bedeutung** von Ausdrücken

- Voraussetzung: Ausdruck ist wohlgeformt (syntaktisch korrekt)
- Linguistik: Was bedeutet ein Begriff oder Satz?
- Logik: Ist ein Satz wahr oder falsch?

Lehre von der **Bedeutung** von Ausdrücken

- Voraussetzung: Ausdruck ist wohlgeformt (syntaktisch korrekt)
- Linguistik: Was bedeutet ein Begriff oder Satz?
- Logik: Ist ein Satz wahr oder falsch?

Beispiel 3.9

	immer wahr	immer falsch	kommt drauf an
Deutsch	Die Erde kreist um die Sonne.	Der Mensch x ist unsterblich.	Heute ist schönes Wetter.

Lehre von der **Bedeutung** von Ausdrücken

- Voraussetzung: Ausdruck ist wohlgeformt (syntaktisch korrekt)
- Linguistik: Was bedeutet ein Begriff oder Satz?
- Logik: Ist ein Satz wahr oder falsch?

Beispiel 3.9

	immer wahr	immer falsch	kommt drauf an
Deutsch	Die Erde kreist um die Sonne.	Der Mensch x ist unsterblich.	Heute ist schönes Wetter.
Arithmetik	$4 + 2 = 6$	$x \cdot x < 0$	$x + 3 = 5$

- Aussagenvariablen erhalten Wert 0 oder 1
- Formel wird wahr oder falsch

- Aussagenvariablen erhalten Wert 0 oder 1
- Formel wird wahr oder falsch

Definition 3.10 (Interpretation in der Aussagenlogik)

Eine **Interpretation** ist eine Funktion $\mathcal{I} : V \rightarrow \mathbb{B}$ mit

- einer Menge von Aussagenvariablen V
- der Boole'schen Menge $\mathbb{B} = \{0, 1\}$

- Aussagenvariablen erhalten Wert 0 oder 1
- Formel wird wahr oder falsch

Definition 3.10 (Interpretation in der Aussagenlogik)

Eine **Interpretation** ist eine Funktion $\mathcal{I} : V \rightarrow \mathbb{B}$ mit

- einer Menge von Aussagenvariablen V
- der Boole'schen Menge $\mathbb{B} = \{0, 1\}$

Beispiel 3.11

$\{A \mapsto 0, B \mapsto 1\} \rightsquigarrow$ „es regnet **nicht**, und die Erde wird nass“

- Aussagenvariablen erhalten Wert 0 oder 1
- Formel wird wahr oder falsch

Definition 3.10 (Interpretation in der Aussagenlogik)

Eine **Interpretation** ist eine Funktion $\mathcal{I} : V \rightarrow \mathbb{B}$ mit

- einer Menge von Aussagenvariablen V
- der Boole'schen Menge $\mathbb{B} = \{0, 1\}$

Beispiel 3.11

$\{A \mapsto 0, B \mapsto 1\} \rightsquigarrow$ „es regnet **nicht**, und die Erde wird nass“

Notation: Statt $\mathcal{I}(A) = 0$ schreibt man auch $A^{\mathcal{I}} = 0$.

$\mathbf{F}^I$	$\mathbf{W}^I$
0	1

$\mathbf{F}^I$	$\mathbf{W}^I$
0	1

φ^I	0	1
$(\neg\varphi)^I$	1	0

$\mathbf{F}^I$	$\mathbf{W}^I$
0	1

$$(\varphi \wedge \psi)^I$$

ψ^I	φ^I	0	1
0		0	0
1		0	1

φ^I	0	1
$(\neg\varphi)^I$	1	0

$\mathbf{F}^I$	$\mathbf{W}^I$
0	1

$$(\varphi \wedge \psi)^I$$

ψ^I	φ^I	0	1
0		0	0
1		0	1

φ^I	0	1
$(\neg\varphi)^I$	1	0

$$(\varphi \vee \psi)^I$$

ψ^I	φ^I	0	1
0		0	1
1		1	1

F^I	W^I
0	1

$$(\varphi \wedge \psi)^I$$

ψ^I	φ^I	0	1
0		0	0
1		0	1

$$(\varphi \rightarrow \psi)^I$$

ψ^I	φ^I	0	1
0		1	0
1		1	1

φ^I	0	1
$(\neg\varphi)^I$	1	0

$$(\varphi \vee \psi)^I$$

ψ^I	φ^I	0	1
0		0	1
1		1	1

$\mathbf{F}^I$	$\mathbf{W}^I$
0	1

$$(\varphi \wedge \psi)^I$$

ψ^I	φ^I	0	1
0		0	0
1		0	1

$$(\varphi \rightarrow \psi)^I$$

ψ^I	φ^I	0	1
0		1	0
1		1	1

φ^I	0	1
$(\neg\varphi)^I$	1	0

$$(\varphi \vee \psi)^I$$

ψ^I	φ^I	0	1
0		0	1
1		1	1

$$(\varphi \leftrightarrow \psi)^I$$

ψ^I	φ^I	0	1
0		1	0
1		0	1

Beispiel 3.12

Die Interpretation $\mathcal{I} = \{A \mapsto 1, B \mapsto 1, C \mapsto 0\}$ macht die Formel ...

- B wahr;

Beispiel 3.12

Die Interpretation $\mathcal{I} = \{A \mapsto 1, B \mapsto 1, C \mapsto 0\}$ macht die Formel ...

- B wahr;
- $A \wedge B$ wahr;

Beispiel 3.12

Die Interpretation $\mathcal{I} = \{A \mapsto 1, B \mapsto 1, C \mapsto 0\}$ macht die Formel ...

- B wahr;
- $A \wedge B$ wahr;
- $A \wedge C$ falsch;

Beispiel 3.12

Die Interpretation $\mathcal{I} = \{A \mapsto 1, B \mapsto 1, C \mapsto 0\}$ macht die Formel ...

- B wahr;
- $A \wedge B$ wahr;
- $A \wedge C$ falsch;
- $(A \wedge B) \vee (A \wedge C)$ wahr;

Beispiel 3.12

Die Interpretation $\mathcal{I} = \{A \mapsto 1, B \mapsto 1, C \mapsto 0\}$ macht die Formel ...

- B wahr;
- $A \wedge B$ wahr;
- $A \wedge C$ falsch;
- $(A \wedge B) \vee (A \wedge C)$ wahr;
- $((A \wedge B) \vee (A \wedge C)) \rightarrow C$ falsch.

Übung 3.13

Finden Sie für die folgenden Formeln je zwei Interpretationen:

- eine, die die Formel wahr macht
- eine, die die Formel falsch macht

1 $A \wedge B \rightarrow C$

2 $(A \vee B) \wedge (A \vee C) \rightarrow (B \wedge C)$

3 $A \rightarrow B \leftrightarrow \neg B \rightarrow \neg A$

Definition 3.14 (Tautologie)

Eine **Tautologie** ist eine Formel, die in jeder Interpretation wahr ist.

Definition 3.14 (Tautologie)

Eine **Tautologie** ist eine Formel, die in jeder Interpretation wahr ist.

Beispiel 3.15

- $\alpha \leftrightarrow \alpha$ (Identität)

Definition 3.14 (Tautologie)

Eine **Tautologie** ist eine Formel, die in jeder Interpretation wahr ist.

Beispiel 3.15

- $\alpha \leftrightarrow \alpha$ (Identität)
- $\neg(\alpha \wedge \neg\alpha)$ (Ausgeschlossener Widerspruch)

Definition 3.14 (Tautologie)

Eine **Tautologie** ist eine Formel, die in jeder Interpretation wahr ist.

Beispiel 3.15

- $\alpha \leftrightarrow \alpha$ (Identität)
- $\neg(\alpha \wedge \neg\alpha)$ (Ausgeschlossener Widerspruch)
- $\alpha \vee \neg\alpha$ (Ausgeschlossenes Drittes)

Definition 3.14 (Tautologie)

Eine **Tautologie** ist eine Formel, die in jeder Interpretation wahr ist.

Beispiel 3.15

- $\alpha \leftrightarrow \alpha$ (Identität)
- $\neg(\alpha \wedge \neg\alpha)$ (Ausgeschlossener Widerspruch)
- $\alpha \vee \neg\alpha$ (Ausgeschlossenes Drittes)
- $\alpha \rightarrow \beta \leftrightarrow \neg\beta \rightarrow \neg\alpha$ (Kontraposition, Umkehrschluss)

Definition 3.14 (Tautologie)

Eine **Tautologie** ist eine Formel, die in jeder Interpretation wahr ist.

Beispiel 3.15

- $\alpha \leftrightarrow \alpha$ (Identität)
- $\neg(\alpha \wedge \neg\alpha)$ (Ausgeschlossener Widerspruch)
- $\alpha \vee \neg\alpha$ (Ausgeschlossenes Drittes)
- $\alpha \rightarrow \beta \leftrightarrow \neg\beta \rightarrow \neg\alpha$ (Kontraposition, Umkehrschluss)
- $(\alpha \rightarrow \beta) \wedge (\alpha \rightarrow \neg\beta) \rightarrow \neg\alpha$ (Reductio ad Absurdum, Widerspruchsbeweis)

Definition 3.14 (Tautologie)

Eine **Tautologie** ist eine Formel, die in jeder Interpretation wahr ist.

Beispiel 3.15

- $\alpha \leftrightarrow \alpha$ (Identität)
- $\neg(\alpha \wedge \neg\alpha)$ (Ausgeschlossener Widerspruch)
- $\alpha \vee \neg\alpha$ (Ausgeschlossenes Drittes)
- $\alpha \rightarrow \beta \leftrightarrow \neg\beta \rightarrow \neg\alpha$ (Kontraposition, Umkehrschluss)
- $(\alpha \rightarrow \beta) \wedge (\alpha \rightarrow \neg\beta) \rightarrow \neg\alpha$ (Reductio ad Absurdum, Widerspruchsbeweis)
- $(\alpha \rightarrow \beta) \wedge \alpha \rightarrow \beta$ (Modus Ponens)

Definition 3.14 (Tautologie)

Eine **Tautologie** ist eine Formel, die in jeder Interpretation wahr ist.

Beispiel 3.15

- $\alpha \leftrightarrow \alpha$ (Identität)
- $\neg(\alpha \wedge \neg\alpha)$ (Ausgeschlossener Widerspruch)
- $\alpha \vee \neg\alpha$ (Ausgeschlossenes Drittes)
- $\alpha \rightarrow \beta \leftrightarrow \neg\beta \rightarrow \neg\alpha$ (Kontraposition, Umkehrschluss)
- $(\alpha \rightarrow \beta) \wedge (\alpha \rightarrow \neg\beta) \rightarrow \neg\alpha$ (Reductio ad Absurdum, Widerspruchsbeweis)
- $(\alpha \rightarrow \beta) \wedge \alpha \rightarrow \beta$ (Modus Ponens)
- $(\alpha \rightarrow \beta) \wedge \neg\beta \rightarrow \neg\alpha$ (Modus Tollens)

Definition 3.14 (Tautologie)

Eine **Tautologie** ist eine Formel, die in jeder Interpretation wahr ist.

Beispiel 3.15

- $\alpha \leftrightarrow \alpha$ (Identität)
- $\neg(\alpha \wedge \neg\alpha)$ (Ausgeschlossener Widerspruch)
- $\alpha \vee \neg\alpha$ (Ausgeschlossenes Drittes)
- $\alpha \rightarrow \beta \leftrightarrow \neg\beta \rightarrow \neg\alpha$ (Kontraposition, Umkehrschluss)
- $(\alpha \rightarrow \beta) \wedge (\alpha \rightarrow \neg\beta) \rightarrow \neg\alpha$ (Reductio ad Absurdum, Widerspruchsbeweis)
- $(\alpha \rightarrow \beta) \wedge \alpha \rightarrow \beta$ (Modus Ponens)
- $(\alpha \rightarrow \beta) \wedge \neg\beta \rightarrow \neg\alpha$ (Modus Tollens)
- $(\alpha \rightarrow \beta) \wedge (\neg\alpha \rightarrow \beta) \rightarrow \beta$ (Fallunterscheidung)

Definition 3.16 (Modell)

Eine Interpretation $\mathcal{I}$ ist ein **Modell** für eine Formel φ ($\mathcal{I} \models \varphi$), wenn $\varphi^{\mathcal{I}} = 1$ gilt.

Definition 3.16 (Modell)

Eine Interpretation $\mathcal{I}$ ist ein **Modell** für eine Formel φ ($\mathcal{I} \models \varphi$), wenn $\varphi^{\mathcal{I}} = 1$ gilt.

Beispiel 3.17

$\mathcal{I} = \{A \mapsto 1, B \mapsto 0\}$ ist Modell für $A \vee B$, aber nicht für $A \wedge B$.

Definition 3.16 (Modell)

Eine Interpretation $\mathcal{I}$ ist ein **Modell** für eine Formel φ ($\mathcal{I} \models \varphi$), wenn $\varphi^{\mathcal{I}} = 1$ gilt.

Beispiel 3.17

$\mathcal{I} = \{A \mapsto 1, B \mapsto 0\}$ ist Modell für $A \vee B$, aber nicht für $A \wedge B$.

Definition 3.18 (Erfüllbarkeit, Gültigkeit)

Eine Formel φ heißt **erfüllbar**, wenn sie ein Modell hat, sonst **unerfüllbar**.

Definition 3.16 (Modell)

Eine Interpretation $\mathcal{I}$ ist ein **Modell** für eine Formel φ ($\mathcal{I} \models \varphi$), wenn $\varphi^{\mathcal{I}} = 1$ gilt.

Beispiel 3.17

$\mathcal{I} = \{A \mapsto 1, B \mapsto 0\}$ ist Modell für $A \vee B$, aber nicht für $A \wedge B$.

Definition 3.18 (Erfüllbarkeit, Gültigkeit)

Eine Formel φ heißt **erfüllbar**, wenn sie ein Modell hat, sonst **unerfüllbar**.

Eine Formel φ heißt **gültig** ($\models \varphi$) wenn jede Interpretation ein Modell ist, sonst **falsifizierbar**.

Definition 3.16 (Modell)

Eine Interpretation $\mathcal{I}$ ist ein **Modell** für eine Formel φ ($\mathcal{I} \models \varphi$), wenn $\varphi^{\mathcal{I}} = 1$ gilt.

Beispiel 3.17

$\mathcal{I} = \{A \mapsto 1, B \mapsto 0\}$ ist Modell für $A \vee B$, aber nicht für $A \wedge B$.

Definition 3.18 (Erfüllbarkeit, Gültigkeit)

Eine Formel φ heißt **erfüllbar**, wenn sie ein Modell hat, sonst **unerfüllbar**.

Eine Formel φ heißt **gültig** ($\models \varphi$) wenn jede Interpretation ein Modell ist, sonst **falsifizierbar**.

Beispiel 3.19

erfüllbar A , $A \vee B$, $A \wedge \neg B$, $A \leftrightarrow B$

Definition 3.16 (Modell)

Eine Interpretation $\mathcal{I}$ ist ein **Modell** für eine Formel φ ($\mathcal{I} \models \varphi$), wenn $\varphi^{\mathcal{I}} = 1$ gilt.

Beispiel 3.17

$\mathcal{I} = \{A \mapsto 1, B \mapsto 0\}$ ist Modell für $A \vee B$, aber nicht für $A \wedge B$.

Definition 3.18 (Erfüllbarkeit, Gültigkeit)

Eine Formel φ heißt **erfüllbar**, wenn sie ein Modell hat, sonst **unerfüllbar**.

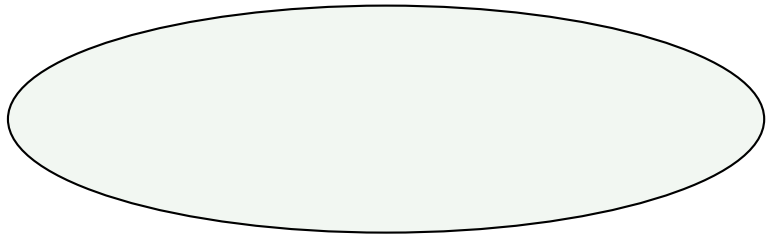
Eine Formel φ heißt **gültig** ($\models \varphi$) wenn jede Interpretation ein Modell ist, sonst **falsifizierbar**.

Beispiel 3.19

erfüllbar A , $A \vee B$, $A \wedge \neg B$, $A \leftrightarrow B$

gültig $A \vee \neg A$, $A \rightarrow A$, $A \leftrightarrow \neg(\neg A)$ (Tautologien)

erfüllbar
Es gibt $\mathcal{I}$ mit $\mathcal{I} \models \varphi$

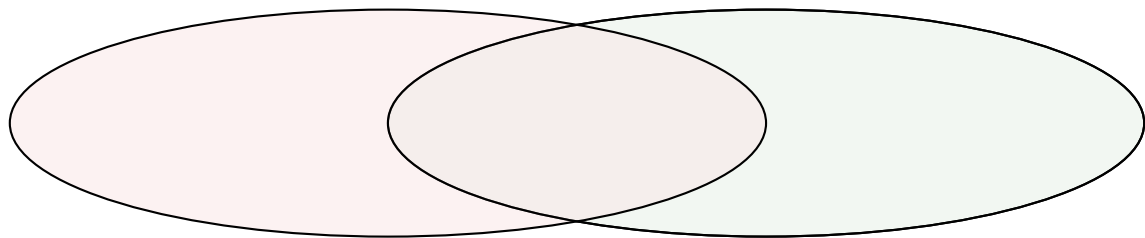


falsifizierbar

Es gibt $\mathcal{I}$ mit $\mathcal{I} \not\models \varphi$

erfüllbar

Es gibt $\mathcal{I}$ mit $\mathcal{I} \models \varphi$

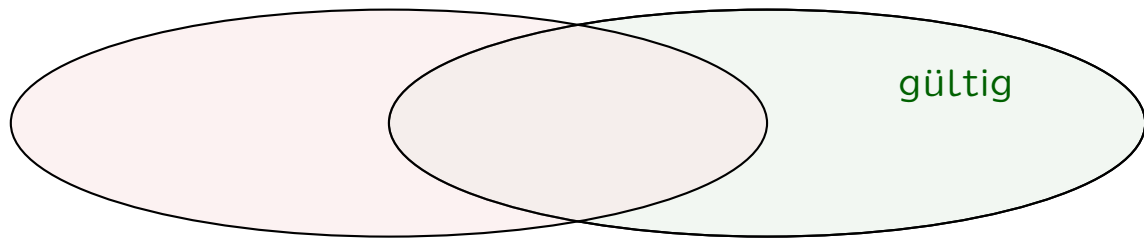


falsifizierbar

Es gibt $\mathcal{I}$ mit $\mathcal{I} \not\models \varphi$

erfüllbar

Es gibt $\mathcal{I}$ mit $\mathcal{I} \models \varphi$

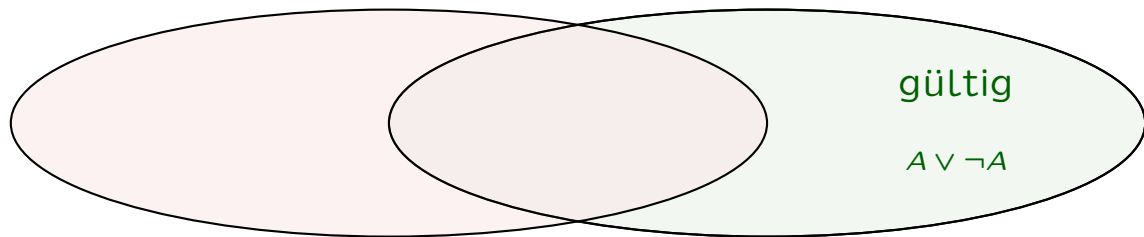


falsifizierbar

Es gibt $\mathcal{I}$ mit $\mathcal{I} \not\models \varphi$

erfüllbar

Es gibt $\mathcal{I}$ mit $\mathcal{I} \models \varphi$

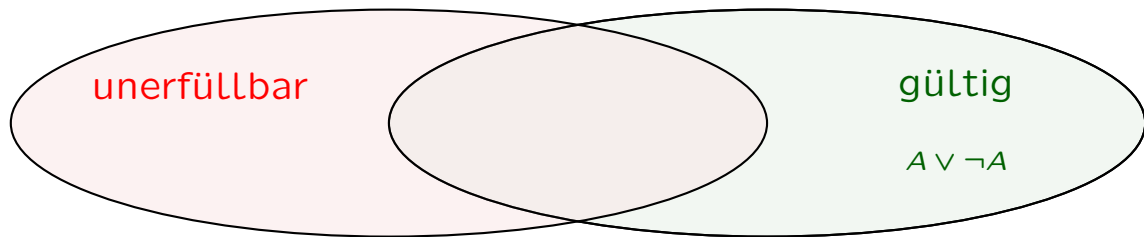


falsifizierbar

Es gibt $\mathcal{I}$ mit $\mathcal{I} \not\models \varphi$

erfüllbar

Es gibt $\mathcal{I}$ mit $\mathcal{I} \models \varphi$

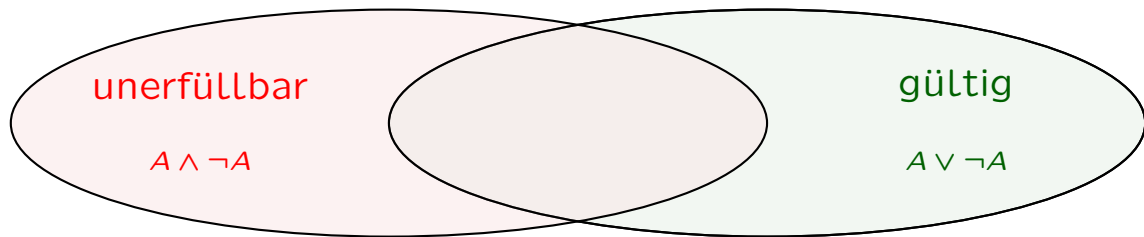


falsifizierbar

Es gibt $\mathcal{I}$ mit $\mathcal{I} \not\models \varphi$

erfüllbar

Es gibt $\mathcal{I}$ mit $\mathcal{I} \models \varphi$

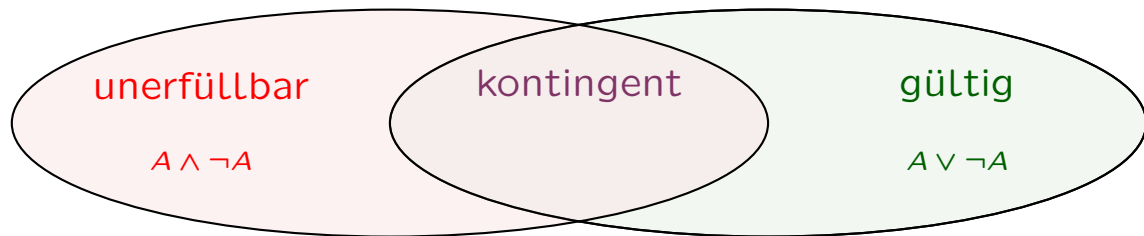


falsifizierbar

Es gibt $\mathcal{I}$ mit $\mathcal{I} \not\models \varphi$

erfüllbar

Es gibt $\mathcal{I}$ mit $\mathcal{I} \models \varphi$

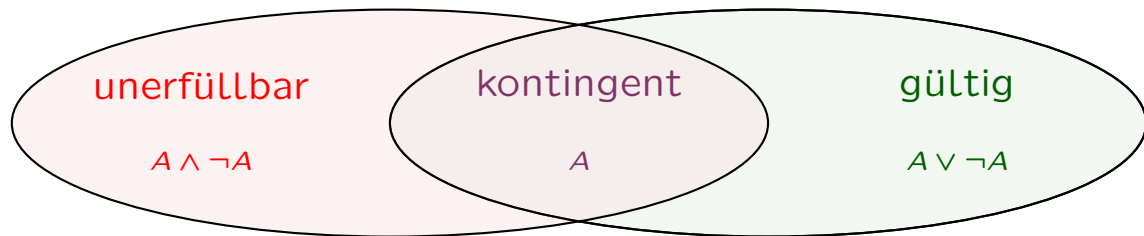


falsifizierbar

Es gibt $\mathcal{I}$ mit $\mathcal{I} \not\models \varphi$

erfüllbar

Es gibt $\mathcal{I}$ mit $\mathcal{I} \models \varphi$



Definition 3.20 (Logische Implikation)

Eine aussagenlogische Formel φ **impliziert** (logisch) eine Formel ψ ($\varphi \models \psi$), wenn jedes Modell von φ auch ein Modell von ψ ist.

Definition 3.20 (Logische Implikation)

Eine aussagenlogische Formel φ **impliziert** (logisch) eine Formel ψ ($\varphi \models \psi$), wenn jedes Modell von φ auch ein Modell von ψ ist.

Beispiel 3.21

$$\alpha \models \alpha$$

Definition 3.20 (Logische Implikation)

Eine aussagenlogische Formel φ **impliziert** (logisch) eine Formel ψ ($\varphi \models \psi$), wenn jedes Modell von φ auch ein Modell von ψ ist.

Beispiel 3.21

$$\alpha \models \alpha$$

$$\alpha \models \alpha \vee \beta$$

Definition 3.20 (Logische Implikation)

Eine aussagenlogische Formel φ **impliziert** (logisch) eine Formel ψ ($\varphi \models \psi$), wenn jedes Modell von φ auch ein Modell von ψ ist.

Beispiel 3.21

$$\begin{aligned}\alpha &\models \alpha \\ \alpha &\models \alpha \vee \beta \\ \alpha \wedge \beta &\models \alpha\end{aligned}$$

Definition 3.20 (Logische Implikation)

Eine aussagenlogische Formel φ **impliziert** (logisch) eine Formel ψ ($\varphi \models \psi$), wenn jedes Modell von φ auch ein Modell von ψ ist.

Beispiel 3.21

$$\begin{aligned}\alpha &\models \alpha \\ \alpha &\models \alpha \vee \beta \\ \alpha \wedge \beta &\models \alpha \\ \mathbf{F} &\models \alpha \quad \text{ex falso quodlibet}\end{aligned}$$

Definition 3.20 (Logische Implikation)

Eine aussagenlogische Formel φ **impliziert** (logisch) eine Formel ψ ($\varphi \models \psi$), wenn jedes Modell von φ auch ein Modell von ψ ist.

Beispiel 3.21

$$\begin{aligned}\alpha &\models \alpha \\ \alpha &\models \alpha \vee \beta \\ \alpha \wedge \beta &\models \alpha \\ \mathbf{F} &\models \alpha \quad \text{ex falso quodlibet} \\ \alpha &\models \mathbf{W}\end{aligned}$$

Definition 3.22 (Äquivalenz)

Zwei aussagenlogische Formeln φ und ψ sind (logisch) **äquivalent** ($\varphi \equiv \psi$), wenn jede Interpretation beiden Formeln denselben Wahrheitswert zuordnet, d. h. wenn für jedes $\mathcal{I}$ gilt: $\varphi^{\mathcal{I}} = \psi^{\mathcal{I}}$.

Definition 3.22 (Äquivalenz)

Zwei aussagenlogische Formeln φ und ψ sind (logisch) **äquivalent** ($\varphi \equiv \psi$), wenn jede Interpretation beiden Formeln denselben Wahrheitswert zuordnet, d. h. wenn für jedes $\mathcal{I}$ gilt: $\varphi^{\mathcal{I}} = \psi^{\mathcal{I}}$.

Beispiel 3.23 (Ersetzung von Operatoren durch äquivalente Ausdrücke)

$$\alpha \leftrightarrow \beta \equiv (\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$$

Definition 3.22 (Äquivalenz)

Zwei aussagenlogische Formeln φ und ψ sind (logisch) **äquivalent** ($\varphi \equiv \psi$), wenn jede Interpretation beiden Formeln denselben Wahrheitswert zuordnet, d. h. wenn für jedes $\mathcal{I}$ gilt: $\varphi^{\mathcal{I}} = \psi^{\mathcal{I}}$.

Beispiel 3.23 (Ersetzung von Operatoren durch äquivalente Ausdrücke)

$$\begin{aligned}\alpha \leftrightarrow \beta &\equiv (\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha) \\ &\equiv \alpha \wedge \beta \vee \neg\alpha \wedge \neg\beta\end{aligned}$$

Definition 3.22 (Äquivalenz)

Zwei aussagenlogische Formeln φ und ψ sind (logisch) **äquivalent** ($\varphi \equiv \psi$), wenn jede Interpretation beiden Formeln denselben Wahrheitswert zuordnet, d. h. wenn für jedes $\mathcal{I}$ gilt: $\varphi^{\mathcal{I}} = \psi^{\mathcal{I}}$.

Beispiel 3.23 (Ersetzung von Operatoren durch äquivalente Ausdrücke)

$$\begin{aligned}\alpha \leftrightarrow \beta &\equiv (\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha) \\ &\equiv \alpha \wedge \beta \vee \neg \alpha \wedge \neg \beta \\ \alpha \rightarrow \beta &\equiv \neg \alpha \vee \beta\end{aligned}$$

Definition 3.22 (Äquivalenz)

Zwei aussagenlogische Formeln φ und ψ sind (logisch) **äquivalent** ($\varphi \equiv \psi$), wenn jede Interpretation beiden Formeln denselben Wahrheitswert zuordnet, d. h. wenn für jedes $\mathcal{I}$ gilt: $\varphi^{\mathcal{I}} = \psi^{\mathcal{I}}$.

Beispiel 3.23 (Ersetzung von Operatoren durch äquivalente Ausdrücke)

$$\begin{aligned}\alpha \leftrightarrow \beta &\equiv (\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha) \\ &\equiv \alpha \wedge \beta \vee \neg \alpha \wedge \neg \beta \\ \alpha \rightarrow \beta &\equiv \neg \alpha \vee \beta \\ \alpha \wedge \beta &\equiv \neg(\neg \alpha \vee \neg \beta)\end{aligned}$$

Definition 3.22 (Äquivalenz)

Zwei aussagenlogische Formeln φ und ψ sind (logisch) **äquivalent** ($\varphi \equiv \psi$), wenn jede Interpretation beiden Formeln denselben Wahrheitswert zuordnet, d. h. wenn für jedes $\mathcal{I}$ gilt: $\varphi^{\mathcal{I}} = \psi^{\mathcal{I}}$.

Beispiel 3.23 (Ersetzung von Operatoren durch äquivalente Ausdrücke)

$$\begin{aligned}\alpha \leftrightarrow \beta &\equiv (\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha) \\ &\equiv \alpha \wedge \beta \vee \neg \alpha \wedge \neg \beta \\ \alpha \rightarrow \beta &\equiv \neg \alpha \vee \beta \\ \alpha \wedge \beta &\equiv \neg(\neg \alpha \vee \neg \beta) \\ \alpha \vee \beta &\equiv \neg(\neg \alpha \wedge \neg \beta)\end{aligned}$$

Satz 3.24

Es seien α , β und γ aussagenlogische Formeln. Dann gelten die folgenden Äquivalenzen:

- Kommutativgesetze

Satz 3.24

Es seien α , β und γ aussagenlogische Formeln. Dann gelten die folgenden Äquivalenzen:

- Kommutativgesetze

- $\alpha \vee \beta \equiv \beta \vee \alpha$

Satz 3.24

Es seien α , β und γ aussagenlogische Formeln. Dann gelten die folgenden Äquivalenzen:

- Kommutativgesetze

- $\alpha \vee \beta \equiv \beta \vee \alpha$

- $\alpha \wedge \beta \equiv \beta \wedge \alpha$

Satz 3.24

Es seien α , β und γ aussagenlogische Formeln. Dann gelten die folgenden Äquivalenzen:

- Kommutativgesetze
 - $\alpha \vee \beta \equiv \beta \vee \alpha$
 - $\alpha \wedge \beta \equiv \beta \wedge \alpha$
- Assoziativgesetze

Satz 3.24

Es seien α , β und γ aussagenlogische Formeln. Dann gelten die folgenden Äquivalenzen:

- Kommutativgesetze

- $\alpha \vee \beta \equiv \beta \vee \alpha$

- $\alpha \wedge \beta \equiv \beta \wedge \alpha$

- Assoziativgesetze

- $\alpha \vee (\beta \vee \gamma) \equiv (\alpha \vee \beta) \vee \gamma$

Satz 3.24

Es seien α , β und γ aussagenlogische Formeln. Dann gelten die folgenden Äquivalenzen:

- Kommutativgesetze

- $\alpha \vee \beta \equiv \beta \vee \alpha$

- $\alpha \wedge \beta \equiv \beta \wedge \alpha$

- Assoziativgesetze

- $\alpha \vee (\beta \vee \gamma) \equiv (\alpha \vee \beta) \vee \gamma$

- $\alpha \wedge (\beta \wedge \gamma) \equiv (\alpha \wedge \beta) \wedge \gamma$

Satz 3.24

Es seien α , β und γ aussagenlogische Formeln. Dann gelten die folgenden Äquivalenzen:

- Kommutativgesetze

- $\alpha \vee \beta \equiv \beta \vee \alpha$

- $\alpha \wedge \beta \equiv \beta \wedge \alpha$

- Assoziativgesetze

- $\alpha \vee (\beta \vee \gamma) \equiv (\alpha \vee \beta) \vee \gamma$

- $\alpha \wedge (\beta \wedge \gamma) \equiv (\alpha \wedge \beta) \wedge \gamma$

- Distributivgesetze

Satz 3.24

Es seien α , β und γ aussagenlogische Formeln. Dann gelten die folgenden Äquivalenzen:

- Kommutativgesetze

- $\alpha \vee \beta \equiv \beta \vee \alpha$
- $\alpha \wedge \beta \equiv \beta \wedge \alpha$

- Assoziativgesetze

- $\alpha \vee (\beta \vee \gamma) \equiv (\alpha \vee \beta) \vee \gamma$
- $\alpha \wedge (\beta \wedge \gamma) \equiv (\alpha \wedge \beta) \wedge \gamma$

- Distributivgesetze

- $\alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$

Satz 3.24

Es seien α , β und γ aussagenlogische Formeln. Dann gelten die folgenden Äquivalenzen:

- Kommutativgesetze

- $\alpha \vee \beta \equiv \beta \vee \alpha$
- $\alpha \wedge \beta \equiv \beta \wedge \alpha$

- Assoziativgesetze

- $\alpha \vee (\beta \vee \gamma) \equiv (\alpha \vee \beta) \vee \gamma$
- $\alpha \wedge (\beta \wedge \gamma) \equiv (\alpha \wedge \beta) \wedge \gamma$

- Distributivgesetze

- $\alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$
- $\alpha \wedge (\beta \vee \gamma) \equiv (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$

Satz 3.24

Es seien α , β und γ aussagenlogische Formeln. Dann gelten die folgenden Äquivalenzen:

- Kommutativgesetze

- $\alpha \vee \beta \equiv \beta \vee \alpha$

- $\alpha \wedge \beta \equiv \beta \wedge \alpha$

- Assoziativgesetze

- $\alpha \vee (\beta \vee \gamma) \equiv (\alpha \vee \beta) \vee \gamma$

- $\alpha \wedge (\beta \wedge \gamma) \equiv (\alpha \wedge \beta) \wedge \gamma$

- Distributivgesetze

- $\alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$

- $\alpha \wedge (\beta \vee \gamma) \equiv (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$

- Gesetze von De Morgan

Satz 3.24

Es seien α , β und γ aussagenlogische Formeln. Dann gelten die folgenden Äquivalenzen:

- Kommutativgesetze

- $\alpha \vee \beta \equiv \beta \vee \alpha$

- $\alpha \wedge \beta \equiv \beta \wedge \alpha$

- Assoziativgesetze

- $\alpha \vee (\beta \vee \gamma) \equiv (\alpha \vee \beta) \vee \gamma$

- $\alpha \wedge (\beta \wedge \gamma) \equiv (\alpha \wedge \beta) \wedge \gamma$

- Distributivgesetze

- $\alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$

- $\alpha \wedge (\beta \vee \gamma) \equiv (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$

- Gesetze von De Morgan

- $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$

Satz 3.24

Es seien α , β und γ aussagenlogische Formeln. Dann gelten die folgenden Äquivalenzen:

- Kommutativgesetze

- $\alpha \vee \beta \equiv \beta \vee \alpha$
- $\alpha \wedge \beta \equiv \beta \wedge \alpha$

- Assoziativgesetze

- $\alpha \vee (\beta \vee \gamma) \equiv (\alpha \vee \beta) \vee \gamma$
- $\alpha \wedge (\beta \wedge \gamma) \equiv (\alpha \wedge \beta) \wedge \gamma$

- Distributivgesetze

- $\alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$
- $\alpha \wedge (\beta \vee \gamma) \equiv (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$

- Gesetze von De Morgan

- $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$
- $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$

Satz 3.24

Es seien α , β und γ aussagenlogische Formeln. Dann gelten die folgenden Äquivalenzen:

■ Kommutativgesetze

- $\alpha \vee \beta \equiv \beta \vee \alpha$
- $\alpha \wedge \beta \equiv \beta \wedge \alpha$

■ Assoziativgesetze

- $\alpha \vee (\beta \vee \gamma) \equiv (\alpha \vee \beta) \vee \gamma$
- $\alpha \wedge (\beta \wedge \gamma) \equiv (\alpha \wedge \beta) \wedge \gamma$

■ Distributivgesetze

- $\alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$
- $\alpha \wedge (\beta \vee \gamma) \equiv (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$

■ Gesetze von De Morgan

- $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$
- $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$

■ Neutrale Elemente

Satz 3.24

Es seien α , β und γ aussagenlogische Formeln. Dann gelten die folgenden Äquivalenzen:

■ Kommutativgesetze

- $\alpha \vee \beta \equiv \beta \vee \alpha$
- $\alpha \wedge \beta \equiv \beta \wedge \alpha$

■ Assoziativgesetze

- $\alpha \vee (\beta \vee \gamma) \equiv (\alpha \vee \beta) \vee \gamma$
- $\alpha \wedge (\beta \wedge \gamma) \equiv (\alpha \wedge \beta) \wedge \gamma$

■ Distributivgesetze

- $\alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$
- $\alpha \wedge (\beta \vee \gamma) \equiv (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$

■ Gesetze von De Morgan

- $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$
- $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$

■ Neutrale Elemente

- $\alpha \vee \mathbf{F} \equiv \alpha$

Satz 3.24

Es seien α , β und γ aussagenlogische Formeln. Dann gelten die folgenden Äquivalenzen:

■ Kommutativgesetze

- $\alpha \vee \beta \equiv \beta \vee \alpha$
- $\alpha \wedge \beta \equiv \beta \wedge \alpha$

■ Assoziativgesetze

- $\alpha \vee (\beta \vee \gamma) \equiv (\alpha \vee \beta) \vee \gamma$
- $\alpha \wedge (\beta \wedge \gamma) \equiv (\alpha \wedge \beta) \wedge \gamma$

■ Distributivgesetze

- $\alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$
- $\alpha \wedge (\beta \vee \gamma) \equiv (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$

■ Gesetze von De Morgan

- $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$
- $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$

■ Neutrale Elemente

- $\alpha \vee \mathbf{F} \equiv \alpha$
- $\alpha \wedge \mathbf{W} \equiv \alpha$

Satz 3.24

Es seien α , β und γ aussagenlogische Formeln. Dann gelten die folgenden Äquivalenzen:

■ Kommutativgesetze

- $\alpha \vee \beta \equiv \beta \vee \alpha$
- $\alpha \wedge \beta \equiv \beta \wedge \alpha$

■ Assoziativgesetze

- $\alpha \vee (\beta \vee \gamma) \equiv (\alpha \vee \beta) \vee \gamma$
- $\alpha \wedge (\beta \wedge \gamma) \equiv (\alpha \wedge \beta) \wedge \gamma$

■ Distributivgesetze

- $\alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$
- $\alpha \wedge (\beta \vee \gamma) \equiv (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$

■ Gesetze von De Morgan

- $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$
- $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$

■ Neutrale Elemente

- $\alpha \vee \mathbf{F} \equiv \alpha$
- $\alpha \wedge \mathbf{W} \equiv \alpha$

■ Inverse Elemente

Satz 3.24

Es seien α , β und γ aussagenlogische Formeln. Dann gelten die folgenden Äquivalenzen:

■ Kommutativgesetze

- $\alpha \vee \beta \equiv \beta \vee \alpha$
- $\alpha \wedge \beta \equiv \beta \wedge \alpha$

■ Assoziativgesetze

- $\alpha \vee (\beta \vee \gamma) \equiv (\alpha \vee \beta) \vee \gamma$
- $\alpha \wedge (\beta \wedge \gamma) \equiv (\alpha \wedge \beta) \wedge \gamma$

■ Distributivgesetze

- $\alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$
- $\alpha \wedge (\beta \vee \gamma) \equiv (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$

■ Gesetze von De Morgan

- $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$
- $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$

■ Neutrale Elemente

- $\alpha \vee \mathbf{F} \equiv \alpha$
- $\alpha \wedge \mathbf{W} \equiv \alpha$

■ Inverse Elemente

- $\alpha \vee \neg\alpha \equiv \mathbf{W}$

Satz 3.24

Es seien α , β und γ aussagenlogische Formeln. Dann gelten die folgenden Äquivalenzen:

■ Kommutativgesetze

- $\alpha \vee \beta \equiv \beta \vee \alpha$
- $\alpha \wedge \beta \equiv \beta \wedge \alpha$

■ Assoziativgesetze

- $\alpha \vee (\beta \vee \gamma) \equiv (\alpha \vee \beta) \vee \gamma$
- $\alpha \wedge (\beta \wedge \gamma) \equiv (\alpha \wedge \beta) \wedge \gamma$

■ Distributivgesetze

- $\alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$
- $\alpha \wedge (\beta \vee \gamma) \equiv (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$

■ Gesetze von De Morgan

- $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$
- $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$

■ Neutrale Elemente

- $\alpha \vee \mathbf{F} \equiv \alpha$
- $\alpha \wedge \mathbf{W} \equiv \alpha$

■ Inverse Elemente

- $\alpha \vee \neg\alpha \equiv \mathbf{W}$
- $\alpha \wedge \neg\alpha \equiv \mathbf{F}$

Satz 3.24

Es seien α , β und γ aussagenlogische Formeln. Dann gelten die folgenden Äquivalenzen:

■ Kommutativgesetze

- $\alpha \vee \beta \equiv \beta \vee \alpha$
- $\alpha \wedge \beta \equiv \beta \wedge \alpha$

■ Assoziativgesetze

- $\alpha \vee (\beta \vee \gamma) \equiv (\alpha \vee \beta) \vee \gamma$
- $\alpha \wedge (\beta \wedge \gamma) \equiv (\alpha \wedge \beta) \wedge \gamma$

■ Distributivgesetze

- $\alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$
- $\alpha \wedge (\beta \vee \gamma) \equiv (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$

■ Gesetze von De Morgan

- $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$
- $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$

■ Neutrale Elemente

- $\alpha \vee \mathbf{F} \equiv \alpha$
- $\alpha \wedge \mathbf{W} \equiv \alpha$

■ Inverse Elemente

- $\alpha \vee \neg\alpha \equiv \mathbf{W}$
- $\alpha \wedge \neg\alpha \equiv \mathbf{F}$

■ Absorbierende Elemente

Satz 3.24

Es seien α , β und γ aussagenlogische Formeln. Dann gelten die folgenden Äquivalenzen:

■ Kommutativgesetze

- $\alpha \vee \beta \equiv \beta \vee \alpha$
- $\alpha \wedge \beta \equiv \beta \wedge \alpha$

■ Assoziativgesetze

- $\alpha \vee (\beta \vee \gamma) \equiv (\alpha \vee \beta) \vee \gamma$
- $\alpha \wedge (\beta \wedge \gamma) \equiv (\alpha \wedge \beta) \wedge \gamma$

■ Distributivgesetze

- $\alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$
- $\alpha \wedge (\beta \vee \gamma) \equiv (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$

■ Gesetze von De Morgan

- $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$
- $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$

■ Neutrale Elemente

- $\alpha \vee \mathbf{F} \equiv \alpha$
- $\alpha \wedge \mathbf{W} \equiv \alpha$

■ Inverse Elemente

- $\alpha \vee \neg\alpha \equiv \mathbf{W}$
- $\alpha \wedge \neg\alpha \equiv \mathbf{F}$

■ Absorbierende Elemente

- $\alpha \vee \mathbf{W} \equiv \mathbf{W}$

Satz 3.24

Es seien α , β und γ aussagenlogische Formeln. Dann gelten die folgenden Äquivalenzen:

■ Kommutativgesetze

- $\alpha \vee \beta \equiv \beta \vee \alpha$
- $\alpha \wedge \beta \equiv \beta \wedge \alpha$

■ Assoziativgesetze

- $\alpha \vee (\beta \vee \gamma) \equiv (\alpha \vee \beta) \vee \gamma$
- $\alpha \wedge (\beta \wedge \gamma) \equiv (\alpha \wedge \beta) \wedge \gamma$

■ Distributivgesetze

- $\alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$
- $\alpha \wedge (\beta \vee \gamma) \equiv (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$

■ Gesetze von De Morgan

- $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$
- $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$

■ Neutrale Elemente

- $\alpha \vee \mathbf{F} \equiv \alpha$
- $\alpha \wedge \mathbf{W} \equiv \alpha$

■ Inverse Elemente

- $\alpha \vee \neg\alpha \equiv \mathbf{W}$
- $\alpha \wedge \neg\alpha \equiv \mathbf{F}$

■ Absorbierende Elemente

- $\alpha \vee \mathbf{W} \equiv \mathbf{W}$
- $\alpha \wedge \mathbf{F} \equiv \mathbf{F}$

Satz 3.24

Es seien α , β und γ aussagenlogische Formeln. Dann gelten die folgenden Äquivalenzen:

■ Kommutativgesetze

- $\alpha \vee \beta \equiv \beta \vee \alpha$
- $\alpha \wedge \beta \equiv \beta \wedge \alpha$

■ Assoziativgesetze

- $\alpha \vee (\beta \vee \gamma) \equiv (\alpha \vee \beta) \vee \gamma$
- $\alpha \wedge (\beta \wedge \gamma) \equiv (\alpha \wedge \beta) \wedge \gamma$

■ Distributivgesetze

- $\alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$
- $\alpha \wedge (\beta \vee \gamma) \equiv (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$

■ Gesetze von De Morgan

- $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$
- $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$

■ Neutrale Elemente

- $\alpha \vee \mathbf{F} \equiv \alpha$
- $\alpha \wedge \mathbf{W} \equiv \alpha$

■ Inverse Elemente

- $\alpha \vee \neg\alpha \equiv \mathbf{W}$
- $\alpha \wedge \neg\alpha \equiv \mathbf{F}$

■ Absorbierende Elemente

- $\alpha \vee \mathbf{W} \equiv \mathbf{W}$
- $\alpha \wedge \mathbf{F} \equiv \mathbf{F}$

■ Idempotenz

Satz 3.24

Es seien α , β und γ aussagenlogische Formeln. Dann gelten die folgenden Äquivalenzen:

■ Kommutativgesetze

- $\alpha \vee \beta \equiv \beta \vee \alpha$
- $\alpha \wedge \beta \equiv \beta \wedge \alpha$

■ Assoziativgesetze

- $\alpha \vee (\beta \vee \gamma) \equiv (\alpha \vee \beta) \vee \gamma$
- $\alpha \wedge (\beta \wedge \gamma) \equiv (\alpha \wedge \beta) \wedge \gamma$

■ Distributivgesetze

- $\alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$
- $\alpha \wedge (\beta \vee \gamma) \equiv (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$

■ Gesetze von De Morgan

- $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$
- $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$

■ Neutrale Elemente

- $\alpha \vee \mathbf{F} \equiv \alpha$
- $\alpha \wedge \mathbf{W} \equiv \alpha$

■ Inverse Elemente

- $\alpha \vee \neg\alpha \equiv \mathbf{W}$
- $\alpha \wedge \neg\alpha \equiv \mathbf{F}$

■ Absorbierende Elemente

- $\alpha \vee \mathbf{W} \equiv \mathbf{W}$
- $\alpha \wedge \mathbf{F} \equiv \mathbf{F}$

■ Idempotenz

- $\alpha \vee \alpha \equiv \alpha$

Satz 3.24

Es seien α , β und γ aussagenlogische Formeln. Dann gelten die folgenden Äquivalenzen:

■ Kommutativgesetze

- $\alpha \vee \beta \equiv \beta \vee \alpha$
- $\alpha \wedge \beta \equiv \beta \wedge \alpha$

■ Assoziativgesetze

- $\alpha \vee (\beta \vee \gamma) \equiv (\alpha \vee \beta) \vee \gamma$
- $\alpha \wedge (\beta \wedge \gamma) \equiv (\alpha \wedge \beta) \wedge \gamma$

■ Distributivgesetze

- $\alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$
- $\alpha \wedge (\beta \vee \gamma) \equiv (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$

■ Gesetze von De Morgan

- $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$
- $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$

■ Neutrale Elemente

- $\alpha \vee \mathbf{F} \equiv \alpha$
- $\alpha \wedge \mathbf{W} \equiv \alpha$

■ Inverse Elemente

- $\alpha \vee \neg\alpha \equiv \mathbf{W}$
- $\alpha \wedge \neg\alpha \equiv \mathbf{F}$

■ Absorbierende Elemente

- $\alpha \vee \mathbf{W} \equiv \mathbf{W}$
- $\alpha \wedge \mathbf{F} \equiv \mathbf{F}$

■ Idempotenz

- $\alpha \vee \alpha \equiv \alpha$
- $\alpha \wedge \alpha \equiv \alpha$

Satz 3.24

Es seien α , β und γ aussagenlogische Formeln. Dann gelten die folgenden Äquivalenzen:

■ Kommutativgesetze

- $\alpha \vee \beta \equiv \beta \vee \alpha$
- $\alpha \wedge \beta \equiv \beta \wedge \alpha$

■ Assoziativgesetze

- $\alpha \vee (\beta \vee \gamma) \equiv (\alpha \vee \beta) \vee \gamma$
- $\alpha \wedge (\beta \wedge \gamma) \equiv (\alpha \wedge \beta) \wedge \gamma$

■ Distributivgesetze

- $\alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$
- $\alpha \wedge (\beta \vee \gamma) \equiv (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$

■ Gesetze von De Morgan

- $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$
- $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$

■ Neutrale Elemente

- $\alpha \vee \mathbf{F} \equiv \alpha$
- $\alpha \wedge \mathbf{W} \equiv \alpha$

■ Inverse Elemente

- $\alpha \vee \neg\alpha \equiv \mathbf{W}$
- $\alpha \wedge \neg\alpha \equiv \mathbf{F}$

■ Absorbierende Elemente

- $\alpha \vee \mathbf{W} \equiv \mathbf{W}$
- $\alpha \wedge \mathbf{F} \equiv \mathbf{F}$

■ Idempotenz

- $\alpha \vee \alpha \equiv \alpha$
- $\alpha \wedge \alpha \equiv \alpha$

■ Doppelte Negation

Satz 3.24

Es seien α , β und γ aussagenlogische Formeln. Dann gelten die folgenden Äquivalenzen:

■ Kommutativgesetze

- $\alpha \vee \beta \equiv \beta \vee \alpha$
- $\alpha \wedge \beta \equiv \beta \wedge \alpha$

■ Assoziativgesetze

- $\alpha \vee (\beta \vee \gamma) \equiv (\alpha \vee \beta) \vee \gamma$
- $\alpha \wedge (\beta \wedge \gamma) \equiv (\alpha \wedge \beta) \wedge \gamma$

■ Distributivgesetze

- $\alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$
- $\alpha \wedge (\beta \vee \gamma) \equiv (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$

■ Gesetze von De Morgan

- $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$
- $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$

■ Neutrale Elemente

- $\alpha \vee \mathbf{F} \equiv \alpha$
- $\alpha \wedge \mathbf{W} \equiv \alpha$

■ Inverse Elemente

- $\alpha \vee \neg\alpha \equiv \mathbf{W}$
- $\alpha \wedge \neg\alpha \equiv \mathbf{F}$

■ Absorbierende Elemente

- $\alpha \vee \mathbf{W} \equiv \mathbf{W}$
- $\alpha \wedge \mathbf{F} \equiv \mathbf{F}$

■ Idempotenz

- $\alpha \vee \alpha \equiv \alpha$
- $\alpha \wedge \alpha \equiv \alpha$

■ Doppelte Negation

- $\neg\neg\alpha \equiv \alpha$

1. Einführung

2. Mengen

3. Aussagenlogik

3.1 Syntax

3.2 Semantik

3.3 Schlussfolgerungsverfahren

4. Prädikatenlogik

5. Prolog

- Bestimmung der Eigenschaften einer Formel φ

- Bestimmung der Eigenschaften einer Formel φ
 - **gültig** (φ ist eine Tautologie; jede Interpretation ist ein Modell)

- Bestimmung der Eigenschaften einer Formel φ
 - **gültig** (φ ist eine Tautologie; jede Interpretation ist ein Modell)
 - **erfüllbar** (Es gibt ein Modell für φ)

- Bestimmung der Eigenschaften einer Formel φ
 - **gültig** (φ ist eine Tautologie; jede Interpretation ist ein Modell)
 - **erfüllbar** (Es gibt ein Modell für φ)
 - **falsifizierbar** (Es gibt eine Interpretation, die kein Modell für φ ist)

- Bestimmung der Eigenschaften einer Formel φ
 - **gültig** (φ ist eine Tautologie; jede Interpretation ist ein Modell)
 - **erfüllbar** (Es gibt ein Modell für φ)
 - **falsifizierbar** (Es gibt eine Interpretation, die kein Modell für φ ist)
 - **unerfüllbar** (Es gibt keine Modelle für φ)

- Bestimmung der Eigenschaften einer Formel φ
 - **gültig** (φ ist eine Tautologie; jede Interpretation ist ein Modell)
 - **erfüllbar** (Es gibt ein Modell für φ)
 - **falsifizierbar** (Es gibt eine Interpretation, die kein Modell für φ ist)
 - **unerfüllbar** (Es gibt keine Modelle für φ)
- Bestimmung, ob eine Formel φ eine Formel ψ (logisch) **impliziert**

Beispiel 3.25 (Reductio ad Absurdum)

$$(A \rightarrow B) \wedge (A \rightarrow \neg B) \rightarrow \neg A \quad ||$$

Beispiel 3.25 (Reductio ad Absurdum)

$$\begin{array}{l} (A \rightarrow B) \wedge (A \rightarrow \neg B) \rightarrow \neg A \\ (\neg A \vee B) \wedge (\neg A \vee \neg B) \rightarrow \neg A \end{array} \quad \parallel \quad \text{Ersetzung von } \rightarrow \text{ (zweimal)}$$

Beispiel 3.25 (Reductio ad Absurdum)

$$\begin{array}{l|l} (A \rightarrow B) \wedge (A \rightarrow \neg B) \rightarrow \neg A & \text{Ersetzung von } \rightarrow \text{ (zweimal)} \\ (\neg A \vee B) \wedge (\neg A \vee \neg B) \rightarrow \neg A & \text{Ersetzung von } \rightarrow \\ \neg((\neg A \vee B) \wedge (\neg A \vee \neg B)) \vee \neg A & \end{array}$$

Beispiel 3.25 (Reductio ad Absurdum)

$(A \rightarrow B) \wedge (A \rightarrow \neg B) \rightarrow \neg A$	Ersetzung von $\rightarrow$ (zweimal)
$(\neg A \vee B) \wedge (\neg A \vee \neg B) \rightarrow \neg A$	Ersetzung von $\rightarrow$
$\neg((\neg A \vee B) \wedge (\neg A \vee \neg B)) \vee \neg A$	De Morgan
$\neg(\neg A \vee B) \vee \neg(\neg A \vee \neg B) \vee \neg A$	

Beispiel 3.25 (Reductio ad Absurdum)

$(A \rightarrow B) \wedge (A \rightarrow \neg B) \rightarrow \neg A$	Ersetzung von $\rightarrow$ (zweimal)
$(\neg A \vee B) \wedge (\neg A \vee \neg B) \rightarrow \neg A$	Ersetzung von $\rightarrow$
$\neg((\neg A \vee B) \wedge (\neg A \vee \neg B)) \vee \neg A$	De Morgan
$\neg(\neg A \vee B) \vee \neg(\neg A \vee \neg B) \vee \neg A$	De Morgan (zweimal)
$(A \wedge \neg B) \vee (A \wedge B) \vee \neg A$	

Beispiel 3.25 (Reductio ad Absurdum)

$(A \rightarrow B) \wedge (A \rightarrow \neg B) \rightarrow \neg A$	Ersetzung von $\rightarrow$ (zweimal)
$(\neg A \vee B) \wedge (\neg A \vee \neg B) \rightarrow \neg A$	Ersetzung von $\rightarrow$
$\neg((\neg A \vee B) \wedge (\neg A \vee \neg B)) \vee \neg A$	De Morgan
$\neg(\neg A \vee B) \vee \neg(\neg A \vee \neg B) \vee \neg A$	De Morgan (zweimal)
$(A \wedge \neg B) \vee (A \wedge B) \vee \neg A$	Distributivität
$(A \wedge (\neg B \vee B)) \vee \neg A$	

Beispiel 3.25 (Reductio ad Absurdum)

$(A \rightarrow B) \wedge (A \rightarrow \neg B) \rightarrow \neg A$	Ersetzung von $\rightarrow$ (zweimal)
$(\neg A \vee B) \wedge (\neg A \vee \neg B) \rightarrow \neg A$	Ersetzung von $\rightarrow$
$\neg((\neg A \vee B) \wedge (\neg A \vee \neg B)) \vee \neg A$	De Morgan
$\neg(\neg A \vee B) \vee \neg(\neg A \vee \neg B) \vee \neg A$	De Morgan (zweimal)
$(A \wedge \neg B) \vee (A \wedge B) \vee \neg A$	Distributivität
$(A \wedge (\neg B \vee B)) \vee \neg A$	Ausgeschlossenes Drittes
$(A \wedge \mathbf{W}) \vee \neg A$	

Beispiel 3.25 (Reductio ad Absurdum)

$$\begin{aligned} & (A \rightarrow B) \wedge (A \rightarrow \neg B) \rightarrow \neg A \\ & (\neg A \vee B) \wedge (\neg A \vee \neg B) \rightarrow \neg A \\ & \neg((\neg A \vee B) \wedge (\neg A \vee \neg B)) \vee \neg A \\ & \neg(\neg A \vee B) \vee \neg(\neg A \vee \neg B) \vee \neg A \\ & (A \wedge \neg B) \vee (A \wedge B) \vee \neg A \\ & (A \wedge (\neg B \vee B)) \vee \neg A \\ & (A \wedge \mathbf{W}) \vee \neg A \\ & A \vee \neg A \end{aligned}$$

Ersetzung von $\rightarrow$ (zweimal)

Ersetzung von $\rightarrow$

De Morgan

De Morgan (zweimal)

Distributivität

Ausgeschlossenes Drittes

$\mathbf{W}$ ist neutrales Element für $\wedge$

Beispiel 3.25 (Reductio ad Absurdum)

$(A \rightarrow B) \wedge (A \rightarrow \neg B) \rightarrow \neg A$	Ersetzung von $\rightarrow$ (zweimal)
$(\neg A \vee B) \wedge (\neg A \vee \neg B) \rightarrow \neg A$	Ersetzung von $\rightarrow$
$\neg((\neg A \vee B) \wedge (\neg A \vee \neg B)) \vee \neg A$	De Morgan
$\neg(\neg A \vee B) \vee \neg(\neg A \vee \neg B) \vee \neg A$	De Morgan (zweimal)
$(A \wedge \neg B) \vee (A \wedge B) \vee \neg A$	Distributivität
$(A \wedge (\neg B \vee B)) \vee \neg A$	Ausgeschlossenes Drittes
$(A \wedge \mathbf{W}) \vee \neg A$	$\mathbf{W}$ ist neutrales Element für $\wedge$
$A \vee \neg A$	Ausgeschlossenes Drittes
$\mathbf{W}$	

Beispiel 3.25 (Reductio ad Absurdum)

$(A \rightarrow B) \wedge (A \rightarrow \neg B) \rightarrow \neg A$	Ersetzung von $\rightarrow$ (zweimal)
$(\neg A \vee B) \wedge (\neg A \vee \neg B) \rightarrow \neg A$	Ersetzung von $\rightarrow$
$\neg((\neg A \vee B) \wedge (\neg A \vee \neg B)) \vee \neg A$	De Morgan
$\neg(\neg A \vee B) \vee \neg(\neg A \vee \neg B) \vee \neg A$	De Morgan (zweimal)
$(A \wedge \neg B) \vee (A \wedge B) \vee \neg A$	Distributivität
$(A \wedge (\neg B \vee B)) \vee \neg A$	Ausgeschlossenes Drittes
$(A \wedge \mathbf{W}) \vee \neg A$	$\mathbf{W}$ ist neutrales Element für $\wedge$
$A \vee \neg A$	Ausgeschlossenes Drittes
$\mathbf{W}$	

Nachteile:

- Transformationen müssen geraten werden.
- Wenn φ nicht gezeigt werden kann, folgt nicht, dass φ nicht gültig ist.

Beispiel 3.26 (Reductio ad Absurdum)

A	B	$A \rightarrow B$	$A \rightarrow \neg B$	$(A \rightarrow B) \wedge (A \rightarrow \neg B)$	$\neg A$	$(A \rightarrow B) \wedge (A \rightarrow \neg B) \rightarrow \neg A$
0	0	1	1	1	1	1

Beispiel 3.26 (Reductio ad Absurdum)

A	B	$A \rightarrow B$	$A \rightarrow \neg B$	$(A \rightarrow B) \wedge (A \rightarrow \neg B)$	$\neg A$	$(A \rightarrow B) \wedge (A \rightarrow \neg B) \rightarrow \neg A$
0	0	1	1	1	1	1
0	1	1	1	1	1	1

Beispiel 3.26 (Reductio ad Absurdum)

A	B	$A \rightarrow B$	$A \rightarrow \neg B$	$(A \rightarrow B) \wedge (A \rightarrow \neg B)$	$\neg A$	$(A \rightarrow B) \wedge (A \rightarrow \neg B) \rightarrow \neg A$
0	0	1	1	1	1	1
0	1	1	1	1	1	1
1	0	0	1	0	0	1

Beispiel 3.26 (Reductio ad Absurdum)

A	B	$A \rightarrow B$	$A \rightarrow \neg B$	$(A \rightarrow B) \wedge (A \rightarrow \neg B)$	$\neg A$	$(A \rightarrow B) \wedge (A \rightarrow \neg B) \rightarrow \neg A$
0	0	1	1	1	1	1
0	1	1	1	1	1	1
1	0	0	1	0	0	1
1	1	1	0	0	0	1

Beispiel 3.26 (Reductio ad Absurdum)

A	B	$A \rightarrow B$	$A \rightarrow \neg B$	$(A \rightarrow B) \wedge (A \rightarrow \neg B)$	$\neg A$	$(A \rightarrow B) \wedge (A \rightarrow \neg B) \rightarrow \neg A$
0	0	1	1	1	1	1
0	1	1	1	1	1	1
1	0	0	1	0	0	1
1	1	1	0	0	0	1

Nachteile:

- ineffizient, besonders bei vielen Variablen („exponentielle Komplexität“)
- redundant, da viele Varianten keinen Einfluss auf das Ergebnis haben

Übung 3.27

Zeigen Sie, dass die Formel φ eine Tautologie ist:

$$\varphi = A \wedge B \rightarrow C \leftrightarrow A \rightarrow (B \rightarrow C)$$

Verwenden Sie

- 1 Äquivalenzumformungen,
- 2 eine Wahrheitstabelle.

1. Einführung

2. Mengen

3. Aussagenlogik

3.1 Syntax

3.2 Semantik

3.3 Schlussfolgerungsverfahren

3.3.1 Resolution

3.3.2 Tableaus

4. Prädikatenlogik

5. Prolog

- John Alan Robinson: *A machine-oriented logic based on the resolution principle* (1965)



J. A. Robinson
(1930–2016)

- John Alan Robinson: *A machine-oriented logic based on the resolution principle* (1965)
- **Erfüllbarkeitstest** für Formel φ



J. A. Robinson
(1930–2016)

- John Alan Robinson: *A machine-oriented logic based on the resolution principle* (1965)
- **Erfüllbarkeitstest** für Formel φ
- Erfüllbarkeitstest entscheidet auch



J. A. Robinson
(1930–2016)

- John Alan Robinson: *A machine-oriented logic based on the resolution principle* (1965)
- **Erfüllbarkeitstest** für Formel φ
- Erfüllbarkeitstest entscheidet auch
 - **Gültigkeit**
 $\models \varphi$ gilt gdw. $\neg\varphi$ unerfüllbar ist.



J. A. Robinson
(1930–2016)

- John Alan Robinson: *A machine-oriented logic based on the resolution principle* (1965)
- **Erfüllbarkeitstest** für Formel φ
- Erfüllbarkeitstest entscheidet auch
 - **Gültigkeit**
 $\models \varphi$ gilt gdw. $\neg\varphi$ unerfüllbar ist.
 - (logische) **Implikation**
 $\varphi \models \psi$ gilt gdw. $\varphi \wedge \neg\psi$ unerfüllbar ist.



J. A. Robinson
(1930–2016)

Definition 3.28 (Literal)

Ein **Literal** ist eine (möglicherweise negierte) Aussagenvariable.

Definition 3.28 (Literal)

Ein **Literal** ist eine (möglicherweise negierte) Aussagenvariable.

Beispiel 3.29

Über der Variablenmenge $\{A, B\}$ ist die Menge der möglichen Literale $\{A, \neg A, B, \neg B\}$.

Finde Widerspruch in φ

Finde Widerspruch in φ

- betrachte φ als Konjunktion von Disjunktionen (**Klauseln**) von Literalen

Beispiel 3.30 ($\varphi = (X \vee Y) \wedge (\neg Y \vee Z) \wedge \neg Z \wedge \neg X$)

$$X \vee Y$$

$$\neg Y \vee Z$$

$$\neg Z$$

$$\neg X$$

Finde Widerspruch in φ

- betrachte φ als Konjunktion von Disjunktionen (Klauseln) von Literalen
- suche Paare von Klauseln C_1, C_2 mit gegensätzlichen Literalen $X, \neg X$

Beispiel 3.30 ($\varphi = (X \vee Y) \wedge (\neg Y \vee Z) \wedge \neg Z \wedge \neg X$)

$X \vee Y$

$\neg Y \vee Z$

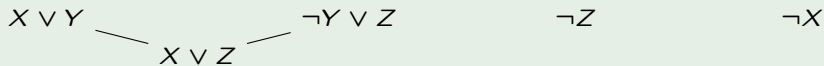
$\neg Z$

$\neg X$

Finde Widerspruch in φ

- betrachte φ als Konjunktion von Disjunktionen (**Klauseln**) von Literalen
- suche Paare von Klauseln C_1, C_2 mit gegensätzlichen Literalen $X, \neg X$
- erzeuge **Resolvente** C_3 durch Vereinigen der **Elternklauseln** C_1 und C_2 und Entfernen von X und $\neg X$

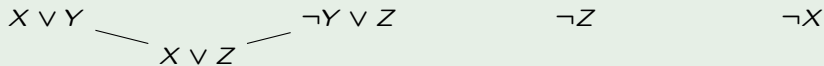
Beispiel 3.30 ($\varphi = (X \vee Y) \wedge (\neg Y \vee Z) \wedge \neg Z \wedge \neg X$)



Finde Widerspruch in φ

- betrachte φ als Konjunktion von Disjunktionen (**Klauseln**) von Literalen
- suche Paare von Klauseln C_1, C_2 mit gegensätzlichen Literalen $X, \neg X$
- erzeuge **Resolvente** C_3 durch Vereinigen der **Elternklauseln** C_1 und C_2 und Entfernen von X und $\neg X$
 - C_3 ist **nicht äquivalent** zu $C_1 \wedge C_2$!

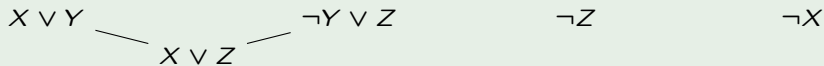
Beispiel 3.30 ($\varphi = (X \vee Y) \wedge (\neg Y \vee Z) \wedge \neg Z \wedge \neg X$)



Finde Widerspruch in φ

- betrachte φ als Konjunktion von Disjunktionen (**Klauseln**) von Literalen
- suche Paare von Klauseln C_1, C_2 mit gegensätzlichen Literalen $X, \neg X$
- erzeuge **Resolvente** C_3 durch Vereinigen der **Elternklauseln** C_1 und C_2 und Entfernen von X und $\neg X$
 - C_3 ist **nicht äquivalent** zu $C_1 \wedge C_2$!
 - $C_1 \wedge C_2$ erfüllbar $\rightsquigarrow$ C_3 erfüllbar

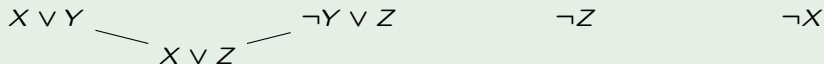
Beispiel 3.30 ($\varphi = (X \vee Y) \wedge (\neg Y \vee Z) \wedge \neg Z \wedge \neg X$)



Finde Widerspruch in φ

- betrachte φ als Konjunktion von Disjunktionen (**Klauseln**) von Literalen
- suche Paare von Klauseln C_1, C_2 mit gegensätzlichen Literalen $X, \neg X$
- erzeuge **Resolvente** C_3 durch Vereinigen der **Elternklauseln** C_1 und C_2 und Entfernen von X und $\neg X$
 - C_3 ist **nicht äquivalent** zu $C_1 \wedge C_2$!
 - $C_1 \wedge C_2$ erfüllbar $\rightsquigarrow$ C_3 erfüllbar
 - C_3 unerfüllbar $\rightsquigarrow$ $C_1 \wedge C_2$ unerfüllbar

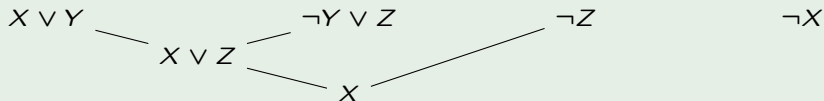
Beispiel 3.30 ($\varphi = (X \vee Y) \wedge (\neg Y \vee Z) \wedge \neg Z \wedge \neg X$)



Finde Widerspruch in φ

- betrachte φ als Konjunktion von Disjunktionen (**Klauseln**) von Literalen
- suche Paare von Klauseln C_1, C_2 mit gegensätzlichen Literalen $X, \neg X$
- erzeuge **Resolvente** C_3 durch Vereinigen der **Elternklauseln** C_1 und C_2 und Entfernen von X und $\neg X$
 - C_3 ist **nicht äquivalent** zu $C_1 \wedge C_2$!
 - $C_1 \wedge C_2$ erfüllbar $\rightsquigarrow$ C_3 erfüllbar
 - C_3 unerfüllbar $\rightsquigarrow$ $C_1 \wedge C_2$ unerfüllbar

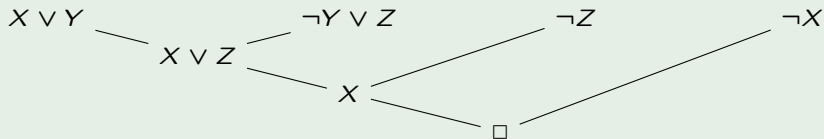
Beispiel 3.30 ($\varphi = (X \vee Y) \wedge (\neg Y \vee Z) \wedge \neg Z \wedge \neg X$)



Finde Widerspruch in φ

- betrachte φ als Konjunktion von Disjunktionen (**Klauseln**) von Literalen
- suche Paare von Klauseln C_1, C_2 mit gegensätzlichen Literalen $X, \neg X$
- erzeuge **Resolvente** C_3 durch Vereinigen der **Elternklauseln** C_1 und C_2 und Entfernen von X und $\neg X$
 - C_3 ist **nicht äquivalent** zu $C_1 \wedge C_2$!
 - $C_1 \wedge C_2$ erfüllbar $\rightsquigarrow$ C_3 erfüllbar
 - C_3 unerfüllbar $\rightsquigarrow$ $C_1 \wedge C_2$ unerfüllbar

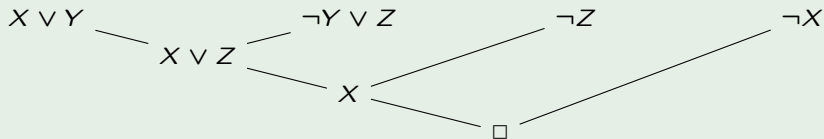
Beispiel 3.30 ($\varphi = (X \vee Y) \wedge (\neg Y \vee Z) \wedge \neg Z \wedge \neg X$)



Finde Widerspruch in φ

- betrachte φ als Konjunktion von Disjunktionen (**Klauseln**) von Literalen
- suche Paare von Klauseln C_1, C_2 mit gegensätzlichen Literalen $X, \neg X$
- erzeuge **Resolvente** C_3 durch Vereinigen der **Elternklauseln** C_1 und C_2 und Entfernen von X und $\neg X$
 - C_3 ist **nicht äquivalent** zu $C_1 \wedge C_2$!
 - $C_1 \wedge C_2$ erfüllbar $\rightsquigarrow$ C_3 erfüllbar
 - C_3 unerfüllbar $\rightsquigarrow$ $C_1 \wedge C_2$ unerfüllbar
- leere Klausel $\square$: Widerspruch

Beispiel 3.30 ($\varphi = (X \vee Y) \wedge (\neg Y \vee Z) \wedge \neg Z \wedge \neg X$)



Warum funktioniert das Resolutionsprinzip?

Gegeben: zwei Klauseln $C_1 = X \vee Y$ und $C_2 = \neg Y \vee Z$.

Gesucht: Modell für $C_1 \wedge C_2$.

Warum funktioniert das Resolutionsprinzip?

Gegeben: zwei Klauseln $C_1 = X \vee Y$ und $C_2 = \neg Y \vee Z$.

Gesucht: Modell für $C_1 \wedge C_2$.

In einer Interpretation $\mathcal{I}$ ist $Y^{\mathcal{I}}$ entweder 0 oder 1.

- wenn $Y^{\mathcal{I}} = 1$ gilt, folgt $C_1^{\mathcal{I}} = 1$, und $C_2^{\mathcal{I}} = 1$ gilt gdw. $Z^{\mathcal{I}} = 1$;
- wenn $Y^{\mathcal{I}} = 0$ gilt, folgt $C_2^{\mathcal{I}} = 1$, und $C_1^{\mathcal{I}} = 1$ gilt gdw. $X^{\mathcal{I}} = 1$.

Warum funktioniert das Resolutionsprinzip?

Gegeben: zwei Klauseln $C_1 = X \vee Y$ und $C_2 = \neg Y \vee Z$.

Gesucht: Modell für $C_1 \wedge C_2$.

In einer Interpretation $\mathcal{I}$ ist $Y^{\mathcal{I}}$ entweder 0 oder 1.

- wenn $Y^{\mathcal{I}} = 1$ gilt, folgt $C_1^{\mathcal{I}} = 1$, und $C_2^{\mathcal{I}} = 1$ gilt gdw. $Z^{\mathcal{I}} = 1$;
- wenn $Y^{\mathcal{I}} = 0$ gilt, folgt $C_2^{\mathcal{I}} = 1$, und $C_1^{\mathcal{I}} = 1$ gilt gdw. $X^{\mathcal{I}} = 1$.

Es folgt:

- Jedes Modell für $C_1 \wedge C_2$ ist auch Modell für $C_3 = X \vee Z$.

Warum funktioniert das Resolutionsprinzip?

Gegeben: zwei Klauseln $C_1 = X \vee Y$ und $C_2 = \neg Y \vee Z$.

Gesucht: Modell für $C_1 \wedge C_2$.

In einer Interpretation $\mathcal{I}$ ist $Y^{\mathcal{I}}$ entweder 0 oder 1.

- wenn $Y^{\mathcal{I}} = 1$ gilt, folgt $C_1^{\mathcal{I}} = 1$, und $C_2^{\mathcal{I}} = 1$ gilt gdw. $Z^{\mathcal{I}} = 1$;
- wenn $Y^{\mathcal{I}} = 0$ gilt, folgt $C_2^{\mathcal{I}} = 1$, und $C_1^{\mathcal{I}} = 1$ gilt gdw. $X^{\mathcal{I}} = 1$.

Es folgt:

- Jedes Modell für $C_1 \wedge C_2$ ist auch Modell für $C_3 = X \vee Z$.
- Wenn $C_1 \wedge C_2$ erfüllbar ist, dann auch $C_1 \wedge C_2 \wedge C_3$.

Warum funktioniert das Resolutionsprinzip?

Gegeben: zwei Klauseln $C_1 = X \vee Y$ und $C_2 = \neg Y \vee Z$.

Gesucht: Modell für $C_1 \wedge C_2$.

In einer Interpretation $\mathcal{I}$ ist $Y^{\mathcal{I}}$ entweder 0 oder 1.

- wenn $Y^{\mathcal{I}} = 1$ gilt, folgt $C_1^{\mathcal{I}} = 1$, und $C_2^{\mathcal{I}} = 1$ gilt gdw. $Z^{\mathcal{I}} = 1$;
- wenn $Y^{\mathcal{I}} = 0$ gilt, folgt $C_2^{\mathcal{I}} = 1$, und $C_1^{\mathcal{I}} = 1$ gilt gdw. $X^{\mathcal{I}} = 1$.

Es folgt:

- Jedes Modell für $C_1 \wedge C_2$ ist auch Modell für $C_3 = X \vee Z$.
- Wenn $C_1 \wedge C_2$ erfüllbar ist, dann auch $C_1 \wedge C_2 \wedge C_3$.
- Wenn $C_1 \wedge C_2 \wedge C_3$ unerfüllbar ist, dann auch $C_1 \wedge C_2$.

Definition 3.31 (Konjunktive Normalform)

Eine aussagenlogische Formel φ ist in **konjunktiver Normalform (KNF)**, wenn φ eine Konjunktion von Disjunktionen von Literalen ist.

Definition 3.31 (Konjunktive Normalform)

Eine aussagenlogische Formel φ ist in **konjunktiver Normalform (KNF)**, wenn φ eine Konjunktion von Disjunktionen von Literalen ist.

Beispiel 3.32

$(X \vee Y) \wedge (\neg Y \vee Z) \wedge \neg Z \wedge \neg X$ ist in KNF;

Definition 3.31 (Konjunktive Normalform)

Eine aussagenlogische Formel φ ist in **konjunktiver Normalform (KNF)**, wenn φ eine Konjunktion von Disjunktionen von Literalen ist.

Beispiel 3.32

$(X \vee Y) \wedge (\neg Y \vee Z) \wedge \neg Z \wedge \neg X$ ist in KNF;
 $(X \wedge Y) \vee \neg(\neg Y \vee Z)$ ist **nicht** in KNF.

Satz 3.33

Jede aussagenlogische Formel kann in eine äquivalente Formel in KNF transformiert werden.

Satz 3.33

Jede aussagenlogische Formel kann in eine äquivalente Formel in KNF transformiert werden.

Verfahren:

- 1.a) eliminiere materiale Äquivalenz
- b) eliminiere materiale Implikation

$$\varphi \leftrightarrow \psi \rightsquigarrow (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$$

$$\varphi \rightarrow \psi \rightsquigarrow \neg\varphi \vee \psi$$

Satz 3.33

Jede aussagenlogische Formel kann in eine äquivalente Formel in KNF transformiert werden.

Verfahren:

- 1.a) eliminiere materiale Äquivalenz
- b) eliminiere materiale Implikation
2. Gesetze von De Morgan

$$\begin{array}{lll} \varphi \leftrightarrow \psi & \rightsquigarrow & (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi) \\ \varphi \rightarrow \psi & \rightsquigarrow & \neg\varphi \vee \psi \\ \neg(\varphi \vee \psi) & \rightsquigarrow & \neg\varphi \wedge \neg\psi \\ \neg(\varphi \wedge \psi) & \rightsquigarrow & \neg\varphi \vee \neg\psi \end{array}$$

Satz 3.33

Jede aussagenlogische Formel kann in eine äquivalente Formel in KNF transformiert werden.

Verfahren:

- 1.a) eliminiere materiale Äquivalenz
- b) eliminiere materiale Implikation
2. Gesetze von De Morgan
- 3.a) eliminiere doppelte Negation
- b) eliminiere negierte Boole-Konstanten

$\varphi \leftrightarrow \psi$	$\rightsquigarrow$	$(\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$
$\varphi \rightarrow \psi$	$\rightsquigarrow$	$\neg \varphi \vee \psi$
$\neg(\varphi \vee \psi)$	$\rightsquigarrow$	$\neg \varphi \wedge \neg \psi$
$\neg(\varphi \wedge \psi)$	$\rightsquigarrow$	$\neg \varphi \vee \neg \psi$
$\neg \neg \varphi$	$\rightsquigarrow$	φ
$\neg \mathbf{W}$	$\rightsquigarrow$	$\mathbf{F}$
$\neg \mathbf{F}$	$\rightsquigarrow$	$\mathbf{W}$

Satz 3.33

Jede aussagenlogische Formel kann in eine äquivalente Formel in KNF transformiert werden.

Verfahren:

- 1.a) eliminiere materiale Äquivalenz
- b) eliminiere materiale Implikation
2. Gesetze von De Morgan
- 3.a) eliminiere doppelte Negation
- b) eliminiere negierte Boole-Konstanten
4. Distributivgesetz

$\varphi \leftrightarrow \psi$	$\rightsquigarrow$	$(\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$
$\varphi \rightarrow \psi$	$\rightsquigarrow$	$\neg \varphi \vee \psi$
$\neg(\varphi \vee \psi)$	$\rightsquigarrow$	$\neg \varphi \wedge \neg \psi$
$\neg(\varphi \wedge \psi)$	$\rightsquigarrow$	$\neg \varphi \vee \neg \psi$
$\neg \neg \varphi$	$\rightsquigarrow$	φ
$\neg \mathbf{W}$	$\rightsquigarrow$	$\mathbf{F}$
$\neg \mathbf{F}$	$\rightsquigarrow$	$\mathbf{W}$
$\varphi \vee (\psi \wedge \chi)$	$\rightsquigarrow$	$(\varphi \vee \psi) \wedge (\varphi \vee \chi)$

Satz 3.33

Jede aussagenlogische Formel kann in eine äquivalente Formel in KNF transformiert werden.

Verfahren:

- 1.a) eliminiere materiale Äquivalenz
- b) eliminiere materiale Implikation
2. Gesetze von De Morgan
- 3.a) eliminiere doppelte Negation
- b) eliminiere negierte Boole-Konstanten
4. Distributivgesetz
5. Assoziativgesetze

$\varphi \leftrightarrow \psi$	$\rightsquigarrow$	$(\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$
$\varphi \rightarrow \psi$	$\rightsquigarrow$	$\neg\varphi \vee \psi$
$\neg(\varphi \vee \psi)$	$\rightsquigarrow$	$\neg\varphi \wedge \neg\psi$
$\neg(\varphi \wedge \psi)$	$\rightsquigarrow$	$\neg\varphi \vee \neg\psi$
$\neg\neg\varphi$	$\rightsquigarrow$	φ
$\neg\mathbf{W}$	$\rightsquigarrow$	$\mathbf{F}$
$\neg\mathbf{F}$	$\rightsquigarrow$	$\mathbf{W}$
$\varphi \vee (\psi \wedge \chi)$	$\rightsquigarrow$	$(\varphi \vee \psi) \wedge (\varphi \vee \chi)$
$\varphi \vee (\psi \vee \chi)$	$\rightsquigarrow$	$\varphi \vee \psi \vee \chi$
$\varphi \wedge (\psi \wedge \chi)$	$\rightsquigarrow$	$\varphi \wedge \psi \wedge \chi$

Satz 3.33

Jede aussagenlogische Formel kann in eine äquivalente Formel in KNF transformiert werden.

Verfahren:

- 1.a) eliminiere materiale Äquivalenz
- b) eliminiere materiale Implikation
2. Gesetze von De Morgan
- 3.a) eliminiere doppelte Negation
- b) eliminiere negierte Boole-Konstanten
4. Distributivgesetz
5. Assoziativgesetze

$$\begin{array}{lll} \varphi \leftrightarrow \psi & \rightsquigarrow & (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi) \\ \varphi \rightarrow \psi & \rightsquigarrow & \neg \varphi \vee \psi \\ \neg(\varphi \vee \psi) & \rightsquigarrow & \neg \varphi \wedge \neg \psi \\ \neg(\varphi \wedge \psi) & \rightsquigarrow & \neg \varphi \vee \neg \psi \\ \neg \neg \varphi & \rightsquigarrow & \varphi \\ \neg \mathbf{W} & \rightsquigarrow & \mathbf{F} \\ \neg \mathbf{F} & \rightsquigarrow & \mathbf{W} \\ \varphi \vee (\psi \wedge \chi) & \rightsquigarrow & (\varphi \vee \psi) \wedge (\varphi \vee \chi) \\ \varphi \vee (\psi \vee \chi) & \rightsquigarrow & \varphi \vee \psi \vee \chi \\ \varphi \wedge (\psi \wedge \chi) & \rightsquigarrow & \varphi \wedge \psi \wedge \chi \end{array}$$

Beispiel 3.34

$$(X \wedge Y) \vee \neg(\neg Y \vee Z) \rightsquigarrow (X \vee Y) \wedge (Y \vee Y) \wedge (X \vee \neg Z) \wedge (Y \vee \neg Z)$$

Übung 3.35

Transformieren Sie die folgenden Formeln in KNF:

1 $(X \vee Y) \wedge (A \wedge B)$

2 $(X \vee Y) \rightarrow (A \vee B)$

3 $X \vee (\neg A \wedge \neg(B \wedge \neg C))$

Definition 3.36 (Klausel)

Eine **Klausel** ist eine als Menge geschriebene Disjunktion von Literalen.

Aufgrund von Idempotenz und Kommutativität ist es möglich und sinnvoll, Disjunktionen als Mengen zu betrachten.

Definition 3.36 (Klausel)

Eine **Klausel** ist eine als Menge geschriebene Disjunktion von Literalen.

Aufgrund von Idempotenz und Kommutativität ist es möglich und sinnvoll, Disjunktionen als Mengen zu betrachten.

Beispiel 3.37

$$(X \vee \neg Y \vee Z) \rightsquigarrow \{X, \neg Y, Z\}$$

Definition 3.36 (Klausel)

Eine **Klausel** ist eine als Menge geschriebene Disjunktion von Literalen.

Aufgrund von Idempotenz und Kommutativität ist es möglich und sinnvoll, Disjunktionen als Mengen zu betrachten.

Beispiel 3.37

$$(X \vee \neg Y \vee Z) \rightsquigarrow \{X, \neg Y, Z\}$$

Jede aussagenlogische Formel (in KNF) kann als Menge von Klauseln betrachtet werden.

Definition 3.36 (Klausel)

Eine **Klausel** ist eine als Menge geschriebene Disjunktion von Literalen.

Aufgrund von Idempotenz und Kommutativität ist es möglich und sinnvoll, Disjunktionen als Mengen zu betrachten.

Beispiel 3.37

$$(X \vee \neg Y \vee Z) \rightsquigarrow \{X, \neg Y, Z\}$$

Jede aussagenlogische Formel (in KNF) kann als Menge von Klauseln betrachtet werden.

Beispiel 3.38

$$\varphi = (X \vee Y) \wedge (\neg Y \vee Z) \wedge \neg Z \wedge \neg X$$

Definition 3.36 (Klausel)

Eine **Klausel** ist eine als Menge geschriebene Disjunktion von Literalen.

Aufgrund von Idempotenz und Kommutativität ist es möglich und sinnvoll, Disjunktionen als Mengen zu betrachten.

Beispiel 3.37

$$(X \vee \neg Y \vee Z) \rightsquigarrow \{X, \neg Y, Z\}$$

Jede aussagenlogische Formel (in KNF) kann als Menge von Klauseln betrachtet werden.

Beispiel 3.38

$$\begin{aligned}\varphi &= (X \vee Y) \wedge (\neg Y \vee Z) \wedge \neg Z \wedge \neg X \\ \mathcal{K}(\varphi) &= \{\{X, Y\}, \{\neg Y, Z\}, \{\neg Z\}, \{\neg X\}\}\end{aligned}$$

Definition 3.36 (Klausel)

Eine **Klausel** ist eine als Menge geschriebene Disjunktion von Literalen.

Aufgrund von Idempotenz und Kommutativität ist es möglich und sinnvoll, Disjunktionen als Mengen zu betrachten.

Beispiel 3.37

$$(X \vee \neg Y \vee Z) \rightsquigarrow \{X, \neg Y, Z\}$$

Jede aussagenlogische Formel (in KNF) kann als Menge von Klauseln betrachtet werden.

Beispiel 3.38

$$\begin{aligned}\varphi &= (X \vee Y) \wedge (\neg Y \vee Z) \wedge \neg Z \wedge \neg X \\ \mathcal{K}(\varphi) &= \{\{X, Y\}, \{\neg Y, Z\}, \{\neg Z\}, \{\neg X\}\}\end{aligned}$$

Intuitiv: $\wedge$ zwischen Klauseln; $\vee$ zwischen Literalen

Definition 3.39 (Resolvente)

Sei V eine Menge von Aussagenvariablen und $X \in V$.

Seien $C_1 = \{X, L_{11}, L_{12}, L_{13}, \dots\}$ und $C_2 = \{\neg X, L_{21}, L_{22}, L_{23}, \dots\}$ Klauseln, wobei alle L_i Literale über V sind.

Dann ist $C_3 = \{L_{11}, L_{12}, L_{13}, \dots, L_{21}, L_{22}, L_{23}, \dots\}$ eine **Resolvente** von C_1 und C_2 ($\{C_1, C_2\} \vdash C_3$).

Definition 3.39 (Resolvente)

Sei V eine Menge von Aussagenvariablen und $X \in V$.

Seien $C_1 = \{X, L_{11}, L_{12}, L_{13}, \dots\}$ und $C_2 = \{\neg X, L_{21}, L_{22}, L_{23}, \dots\}$ Klauseln, wobei alle L_i Literale über V sind.

Dann ist $C_3 = \{L_{11}, L_{12}, L_{13}, \dots, L_{21}, L_{22}, L_{23}, \dots\}$ eine **Resolvente** von C_1 und C_2 ($\{C_1, C_2\} \vdash C_3$).

Kann eine Klausel C aus einer Klauselmenge S durch (beliebig viele, einschließlich 0) Resolutionsschritte hergeleitet werden, schreiben wir $S \vdash^* C$.

Übung 3.40

Finden Sie für die Klauselmengen S möglichst viele Resolventen.

$$S = \{\{A, C\}, \{A, \neg B\}, \{\neg A, \neg B\}, \{A, \neg C\}\}$$

Eingabe: AL-Formel φ

Ausgabe: „erfüllbar“ oder „unerfüllbar“

Eingabe: AL-Formel φ

Ausgabe: „erfüllbar“ oder „unerfüllbar“

1: $\varphi' := \text{knf}(\varphi)$

Eingabe: AL-Formel φ

Ausgabe: „erfüllbar“ oder „unerfüllbar“

- 1: $\varphi' := \text{knf}(\varphi)$
- 2: initialisiere S mit Disjunktionen von φ'

Eingabe: AL-Formel φ

Ausgabe: „erfüllbar“ oder „unerfüllbar“

- 1: $\varphi' := \text{knf}(\varphi)$
- 2: initialisiere S mit Disjunktionen von φ'
- 3: **while** es existieren $\{C_1, C_2\} \subseteq S$ mit $\{C_1, C_2\} \vdash C_{\text{neu}}$ und $C_{\text{neu}} \notin S$ **do**

Eingabe: AL-Formel φ

Ausgabe: „erfüllbar“ oder „unerfüllbar“

- 1: $\varphi' := \text{knf}(\varphi)$
- 2: initialisiere S mit Disjunktionen von φ'
- 3: **while** es existieren $\{C_1, C_2\} \subseteq S$ mit $\{C_1, C_2\} \vdash C_{\text{neu}}$ und $C_{\text{neu}} \notin S$ **do**
- 4: **if** $C_{\text{neu}} = \square$ **then**
- 5: **return** „unerfüllbar“

Eingabe: AL-Formel φ

Ausgabe: „erfüllbar“ oder „unerfüllbar“

```
1:  $\varphi' := \text{knf}(\varphi)$ 
2: initialisiere  $S$  mit Disjunktionen von  $\varphi'$ 
3: while es existieren  $\{C_1, C_2\} \subseteq S$  mit  $\{C_1, C_2\} \vdash C_{\text{neu}}$  und  $C_{\text{neu}} \notin S$  do
4:     if  $C_{\text{neu}} = \square$  then
5:         return „unerfüllbar“
6:     else
7:          $S := S \cup \{C_{\text{neu}}\}$ 
```

Eingabe: AL-Formel φ

Ausgabe: „erfüllbar“ oder „unerfüllbar“

```
1:  $\varphi' := \text{knf}(\varphi)$ 
2: initialisiere  $S$  mit Disjunktionen von  $\varphi'$ 
3: while es existieren  $\{C_1, C_2\} \subseteq S$  mit  $\{C_1, C_2\} \vdash C_{\text{neu}}$  und  $C_{\text{neu}} \notin S$  do
4:     if  $C_{\text{neu}} = \square$  then
5:         return „unerfüllbar“
6:     else
7:          $S := S \cup \{C_{\text{neu}}\}$ 
8: return „erfüllbar“
```

Satz 3.41 (Korrektheit der Resolution)

Sei S eine Klauselmenge. Aus $S \vdash^* C$ folgt $S \models C$.

Satz 3.41 (Korrektheit der Resolution)

Sei S eine Klauselmenge. Aus $S \vdash^* C$ folgt $S \models C$.

Beweis.

- Gilt $S \vdash C$, ist jedes Modell für S auch Modell für C .

Satz 3.41 (Korrektheit der Resolution)

Sei S eine Klauselmenge. Aus $S \vdash^* C$ folgt $S \models C$.

Beweis.

- Gilt $S \vdash C$, ist jedes Modell für S auch Modell für C .
- Mit Induktion: $S \vdash^* C$ impliziert $S \models C$.



Satz 3.41 (Korrektheit der Resolution)

Sei S eine Klauselmenge. Aus $S \vdash^* C$ folgt $S \models C$.

Beweis.

- Gilt $S \vdash C$, ist jedes Modell für S auch Modell für C .
- Mit Induktion: $S \vdash^* C$ impliziert $S \models C$.



Satz 3.41 (Korrektheit der Resolution)

Sei S eine Klauselmenge. Aus $S \vdash^* C$ folgt $S \models C$.

Beweis.

- Gilt $S \vdash C$, ist jedes Modell für S auch Modell für C .
- Mit Induktion: $S \vdash^* C$ impliziert $S \models C$. □

Korollar 3.42

Wenn $S \vdash^* \square$ gilt, ist S unerfüllbar.

Satz 3.41 (Korrektheit der Resolution)

Sei S eine Klauselmenge. Aus $S \vdash^* C$ folgt $S \models C$.

Beweis.

- Gilt $S \vdash C$, ist jedes Modell für S auch Modell für C .
- Mit Induktion: $S \vdash^* C$ impliziert $S \models C$. □

Korollar 3.42

Wenn $S \vdash^* \square$ gilt, ist S unerfüllbar.

Beweis.

- Ist S erfüllbar, dann auch $S \cup \{C\}$.

Satz 3.41 (Korrektheit der Resolution)

Sei S eine Klauselmenge. Aus $S \vdash^* C$ folgt $S \models C$.

Beweis.

- Gilt $S \vdash C$, ist jedes Modell für S auch Modell für C .
- Mit Induktion: $S \vdash^* C$ impliziert $S \models C$. □

Korollar 3.42

Wenn $S \vdash^* \square$ gilt, ist S unerfüllbar.

Beweis.

- Ist S erfüllbar, dann auch $S \cup \{C\}$.
- Umgekehrt: Ist $S \cup \{C\}$ unerfüllbar, dann auch S .

Satz 3.41 (Korrektheit der Resolution)

Sei S eine Klauselmenge. Aus $S \vdash^* C$ folgt $S \models C$.

Beweis.

- Gilt $S \vdash C$, ist jedes Modell für S auch Modell für C .
- Mit Induktion: $S \vdash^* C$ impliziert $S \models C$.



Korollar 3.42

Wenn $S \vdash^* \square$ gilt, ist S unerfüllbar.

Beweis.

- Ist S erfüllbar, dann auch $S \cup \{C\}$.
- Umgekehrt: Ist $S \cup \{C\}$ unerfüllbar, dann auch S .
- Die leere Klausel $\square$ ist unerfüllbar.



Satz 3.43 (Widerlegungsvollständigkeit der Resolution)

Ist eine Klauselmeng S unerfüllbar, gilt $S \vdash^* \square$.

Satz 3.43 (Widerlegungsvollständigkeit der Resolution)

Ist eine Klauselmenge S unerfüllbar, gilt $S \vdash^* \square$.

Beweis.

Induktion über Anzahl n der Aussagenvariablen in S .

$n = 0$ Dann gilt $S = \{\square\}$ und damit $S \vdash^* \square$.

Satz 3.43 (Widerlegungsvollständigkeit der Resolution)

Ist eine Klauselmenge S unerfüllbar, gilt $S \vdash^* \square$.

Beweis.

Induktion über Anzahl n der Aussagenvariablen in S .

$n = 0$ Dann gilt $S = \{\square\}$ und damit $S \vdash^* \square$.

Satz 3.43 (Widerlegungsvollständigkeit der Resolution)

Ist eine Klauselmenge S unerfüllbar, gilt $S \vdash^* \square$.

Beweis.

Induktion über Anzahl n der Aussagenvariablen in S .

$n = 0$ Dann gilt $S = \{\square\}$ und damit $S \vdash^* \square$.

$n \rightsquigarrow n + 1$ ■ Sei S eine unerfüllbare Klauselmenge mit den Variablen $A_1, \dots, A_n, A_{n+1}$.

Satz 3.43 (Widerlegungsvollständigkeit der Resolution)

Ist eine Klauselmenge S unerfüllbar, gilt $S \vdash^* \square$.

Beweis.

Induktion über Anzahl n der Aussagenvariablen in S .

$n = 0$ Dann gilt $S = \{\square\}$ und damit $S \vdash^* \square$.

- $n \rightsquigarrow n + 1$
- Sei S eine unerfüllbare Klauselmenge mit den Variablen $A_1, \dots, A_n, A_{n+1}$.
 - Definiere S_1 : Entferne aus S alle Klauseln, die das Literal A_{n+1} enthalten, und aus verbleibenden Klauseln alle Literale $\neg A_{n+1}$. (Intuitiv: $A_{n+1} \mapsto 1$)
 - Definiere S_0 : Entferne aus S alle Klauseln, die das Literal $\neg A_{n+1}$ enthalten, und aus verbleibenden Klauseln alle Literale A_{n+1} . (Intuitiv: $A_{n+1} \mapsto 0$)

Satz 3.43 (Widerlegungsvollständigkeit der Resolution)

Ist eine Klauselmenge S unerfüllbar, gilt $S \vdash^* \square$.

Beweis.

Induktion über Anzahl n der Aussagenvariablen in S .

$n = 0$ Dann gilt $S = \{\square\}$ und damit $S \vdash^* \square$.

- $n \rightsquigarrow n + 1$
- Sei S eine unerfüllbare Klauselmenge mit den Variablen $A_1, \dots, A_n, A_{n+1}$.
 - Definiere S_1 : Entferne aus S alle Klauseln, die das Literal A_{n+1} enthalten, und aus verbleibenden Klauseln alle Literale $\neg A_{n+1}$. (Intuitiv: $A_{n+1} \mapsto 1$)
Definiere S_0 : Entferne aus S alle Klauseln, die das Literal $\neg A_{n+1}$ enthalten, und aus verbleibenden Klauseln alle Literale A_{n+1} . (Intuitiv: $A_{n+1} \mapsto 0$)
 - S_1 und S_0 sind unerfüllbar (sonst wäre auch S erfüllbar) und enthalten n Variablen.

Satz 3.43 (Widerlegungsvollständigkeit der Resolution)

Ist eine Klauselmenge S unerfüllbar, gilt $S \vdash^* \square$.

Beweis.

Induktion über Anzahl n der Aussagenvariablen in S .

$n = 0$ Dann gilt $S = \{\square\}$ und damit $S \vdash^* \square$.

- $n \rightsquigarrow n + 1$
- Sei S eine unerfüllbare Klauselmenge mit den Variablen $A_1, \dots, A_n, A_{n+1}$.
 - Definiere S_1 : Entferne aus S alle Klauseln, die das Literal A_{n+1} enthalten, und aus verbleibenden Klauseln alle Literale $\neg A_{n+1}$. (Intuitiv: $A_{n+1} \mapsto 1$)
Definiere S_0 : Entferne aus S alle Klauseln, die das Literal $\neg A_{n+1}$ enthalten, und aus verbleibenden Klauseln alle Literale A_{n+1} . (Intuitiv: $A_{n+1} \mapsto 0$)
 - S_1 und S_0 sind unerfüllbar (sonst wäre auch S erfüllbar) und enthalten n Variablen.
 - **Induktionsvoraussetzung:** $S_1 \vdash^* \square$ und $S_0 \vdash^* \square$.

Satz 3.43 (Widerlegungsvollständigkeit der Resolution)

Ist eine Klauselmenge S unerfüllbar, gilt $S \vdash^* \square$.

Beweis.

Induktion über Anzahl n der Aussagenvariablen in S .

$n = 0$ Dann gilt $S = \{\square\}$ und damit $S \vdash^* \square$.

- $n \rightsquigarrow n + 1$
- Sei S eine unerfüllbare Klauselmenge mit den Variablen $A_1, \dots, A_n, A_{n+1}$.
 - Definiere S_1 : Entferne aus S alle Klauseln, die das Literal A_{n+1} enthalten, und aus verbleibenden Klauseln alle Literale $\neg A_{n+1}$. (Intuitiv: $A_{n+1} \mapsto 1$)
Definiere S_0 : Entferne aus S alle Klauseln, die das Literal $\neg A_{n+1}$ enthalten, und aus verbleibenden Klauseln alle Literale A_{n+1} . (Intuitiv: $A_{n+1} \mapsto 0$)
 - S_1 und S_0 sind unerfüllbar (sonst wäre auch S erfüllbar) und enthalten n Variablen.
 - **Induktionsvoraussetzung:** $S_1 \vdash^* \square$ und $S_0 \vdash^* \square$.
 - Aus $S_1 \vdash^* \square$ folgt $S \vdash^* \{\neg A_{n+1}\}$; aus $S_0 \vdash^* \square$ folgt $S \vdash^* \{A_{n+1}\}$.

Satz 3.43 (Widerlegungsvollständigkeit der Resolution)

Ist eine Klauselmeng S unerfüllbar, gilt $S \vdash^* \square$.

Beweis.

Induktion über Anzahl n der Aussagenvariablen in S .

$n = 0$ Dann gilt $S = \{\square\}$ und damit $S \vdash^* \square$.

- $n \rightsquigarrow n + 1$
- Sei S eine unerfüllbare Klauselmeng mit den Variablen $A_1, \dots, A_n, A_{n+1}$.
 - Definiere S_1 : Entferne aus S alle Klauseln, die das Literal A_{n+1} enthalten, und aus verbleibenden Klauseln alle Literale $\neg A_{n+1}$. (Intuitiv: $A_{n+1} \mapsto 1$)
Definiere S_0 : Entferne aus S alle Klauseln, die das Literal $\neg A_{n+1}$ enthalten, und aus verbleibenden Klauseln alle Literale A_{n+1} . (Intuitiv: $A_{n+1} \mapsto 0$)
 - S_1 und S_0 sind unerfüllbar (sonst wäre auch S erfüllbar) und enthalten n Variablen.
 - **Induktionsvoraussetzung:** $S_1 \vdash^* \square$ und $S_0 \vdash^* \square$.
 - Aus $S_1 \vdash^* \square$ folgt $S \vdash^* \{\neg A_{n+1}\}$; aus $S_0 \vdash^* \square$ folgt $S \vdash^* \{A_{n+1}\}$.
 - Durch Resolution von $\{A_{n+1}\}$ und $\{\neg A_{n+1}\}$ folgt $S \vdash^* \square$. □

Beispiel 3.44

Sei $S = \{\{A_1, A_2\}, \{\neg A_1\}, \{\neg A_2\}\}$

■ $S_1 = \{\{\neg A_1\}, \square\}$

Beispiel 3.44

Sei $S = \{\{A_1, A_2\}, \{\neg A_1\}, \{\neg A_2\}\}$

- $S_1 = \{\{\neg A_1\}, \Box\}$

- Aus $\Box \in S_1$ folgt $S_1 \vdash^* \Box$

Beispiel 3.44

Sei $S = \{\{A_1, A_2\}, \{\neg A_1\}, \{\neg A_2\}\}$

- $S_1 = \{\{\neg A_1\}, \Box\}$

- Aus $\Box \in S_1$ folgt $S_1 \vdash^* \Box$

- $S_0 = \{\{A_1\}, \{\neg A_1\}\}$

Beispiel 3.44

Sei $S = \{\{A_1, A_2\}, \{\neg A_1\}, \{\neg A_2\}\}$

- $S_1 = \{\{\neg A_1\}, \Box\}$

- Aus $\Box \in S_1$ folgt $S_1 \vdash^* \Box$

- $S_0 = \{\{A_1\}, \{\neg A_1\}\}$

- $S_0 \vdash^* \Box$ folgt per Induktion:

Beispiel 3.44

Sei $S = \{\{A_1, A_2\}, \{\neg A_1\}, \{\neg A_2\}\}$

- $S_1 = \{\{\neg A_1\}, \Box\}$

- Aus $\Box \in S_1$ folgt $S_1 \vdash^* \Box$

- $S_0 = \{\{A_1\}, \{\neg A_1\}\}$

- $S_0 \vdash^* \Box$ folgt per Induktion:

- $S_{01} = \{\Box\}$; es folgt $S_0 \vdash^* \{\neg A_1\}$

Beispiel 3.44

Sei $S = \{\{A_1, A_2\}, \{\neg A_1\}, \{\neg A_2\}\}$

- $S_1 = \{\{\neg A_1\}, \Box\}$

- Aus $\Box \in S_1$ folgt $S_1 \vdash^* \Box$

- $S_0 = \{\{A_1\}, \{\neg A_1\}\}$

- $S_0 \vdash^* \Box$ folgt per Induktion:

- $S_{01} = \{\Box\}$; es folgt $S_0 \vdash^* \{\neg A_1\}$

- $S_{00} = \{\Box\}$; es folgt $S_0 \vdash^* \{A_1\}$

Beispiel 3.44

Sei $S = \{\{A_1, A_2\}, \{\neg A_1\}, \{\neg A_2\}\}$

- $S_1 = \{\{\neg A_1\}, \square\}$

- Aus $\square \in S_1$ folgt $S_1 \vdash^* \square$

- $S_0 = \{\{A_1\}, \{\neg A_1\}\}$

- $S_0 \vdash^* \square$ folgt per Induktion:

- $S_{01} = \{\square\}$; es folgt $S_0 \vdash^* \{\neg A_1\}$

- $S_{00} = \{\square\}$; es folgt $S_0 \vdash^* \{A_1\}$

- Durch Resolution aus $\{A_1\}$ und $\{\neg A_1\}$ folgt $S_0 \vdash^* \square$.

Beispiel 3.44

Sei $S = \{\{A_1, A_2\}, \{\neg A_1\}, \{\neg A_2\}\}$

- $S_1 = \{\{\neg A_1\}, \Box\}$

- Aus $\Box \in S_1$ folgt $S_1 \vdash^* \Box$

- $S_0 = \{\{A_1\}, \{\neg A_1\}\}$

- $S_0 \vdash^* \Box$ folgt per Induktion:

- $S_{01} = \{\Box\}$; es folgt $S_0 \vdash^* \{\neg A_1\}$

- $S_{00} = \{\Box\}$; es folgt $S_0 \vdash^* \{A_1\}$

- Durch Resolution aus $\{A_1\}$ und $\{\neg A_1\}$ folgt $S_0 \vdash^* \Box$.

- $S \vdash^* \Box$ folgt per Induktion:

Beispiel 3.44

Sei $S = \{\{A_1, A_2\}, \{\neg A_1\}, \{\neg A_2\}\}$

- $S_1 = \{\{\neg A_1\}, \square\}$

- Aus $\square \in S_1$ folgt $S_1 \vdash^* \square$

- $S_0 = \{\{A_1\}, \{\neg A_1\}\}$

- $S_0 \vdash^* \square$ folgt per Induktion:

- $S_{01} = \{\square\}$; es folgt $S_0 \vdash^* \{\neg A_1\}$

- $S_{00} = \{\square\}$; es folgt $S_0 \vdash^* \{A_1\}$

- Durch Resolution aus $\{A_1\}$ und $\{\neg A_1\}$ folgt $S_0 \vdash^* \square$.

- $S \vdash^* \square$ folgt per Induktion:

- Aus $S_0 \vdash^* \square$ folgt $S \vdash^* \{A_2\}$

Beispiel 3.44

Sei $S = \{\{A_1, A_2\}, \{\neg A_1\}, \{\neg A_2\}\}$

- $S_1 = \{\{\neg A_1\}, \square\}$

- Aus $\square \in S_1$ folgt $S_1 \vdash^* \square$

- $S_0 = \{\{A_1\}, \{\neg A_1\}\}$

- $S_0 \vdash^* \square$ folgt per Induktion:

- $S_{01} = \{\square\}$; es folgt $S_0 \vdash^* \{\neg A_1\}$

- $S_{00} = \{\square\}$; es folgt $S_0 \vdash^* \{A_1\}$

- Durch Resolution aus $\{A_1\}$ und $\{\neg A_1\}$ folgt $S_0 \vdash^* \square$.

- $S \vdash^* \square$ folgt per Induktion:

- Aus $S_0 \vdash^* \square$ folgt $S \vdash^* \{A_2\}$

- Aus $S_1 \vdash^* \square$ folgt $S \vdash^* \{\neg A_2\}$

Beispiel 3.44

Sei $S = \{\{A_1, A_2\}, \{\neg A_1\}, \{\neg A_2\}\}$

- $S_1 = \{\{\neg A_1\}, \square\}$

- Aus $\square \in S_1$ folgt $S_1 \vdash^* \square$

- $S_0 = \{\{A_1\}, \{\neg A_1\}\}$

- $S_0 \vdash^* \square$ folgt per Induktion:

- $S_{01} = \{\square\}$; es folgt $S_0 \vdash^* \{\neg A_1\}$

- $S_{00} = \{\square\}$; es folgt $S_0 \vdash^* \{A_1\}$

- Durch Resolution aus $\{A_1\}$ und $\{\neg A_1\}$ folgt $S_0 \vdash^* \square$.

- $S \vdash^* \square$ folgt per Induktion:

- Aus $S_0 \vdash^* \square$ folgt $S \vdash^* \{A_2\}$

- Aus $S_1 \vdash^* \square$ folgt $S \vdash^* \{\neg A_2\}$

- Durch Resolution aus $\{A_2\}$ und $\{\neg A_2\}$ folgt $S \vdash^* \square$.

Satz 3.45 (Terminierung der Resolution)

Der Resolutions-Algorithmus terminiert für jede Eingabe.

Satz 3.45 (Terminierung der Resolution)

Der Resolutions-Algorithmus terminiert für jede Eingabe.

Beweis.

- Neue Klauseln enthalten nur Literale der Eingabe. (Endliches Vokabular)

Satz 3.45 (Terminierung der Resolution)

Der Resolutions-Algorithmus terminiert für jede Eingabe.

Beweis.

- Neue Klauseln enthalten nur Literale der Eingabe. (Endliches Vokabular)
- Nur **neue** Klauseln werden hinzugefügt.

Satz 3.45 (Terminierung der Resolution)

Der Resolutions-Algorithmus terminiert für jede Eingabe.

Beweis.

- Neue Klauseln enthalten nur Literale der Eingabe. (Endliches Vokabular)
- Nur **neue** Klauseln werden hinzugefügt.
- Klauseln werden nie entfernt.



Satz 3.45 (Terminierung der Resolution)

Der Resolutions-Algorithmus terminiert für jede Eingabe.

Beweis.

- Neue Klauseln enthalten nur Literale der Eingabe. (Endliches Vokabular)
- Nur **neue** Klauseln werden hinzugefügt.
- Klauseln werden nie entfernt. □

Laufzeit: worst-case exponentiell (Formel der Länge $n \rightsquigarrow 2^n$ Schritte)

Prämissen:

- 1 Wenn Hans zum Meeting eingeladen wird und nicht auf Dienstreise ist, nimmt er am Meeting teil.

$$\neg R \wedge E \rightarrow M \quad \rightsquigarrow \quad R \vee \neg E \vee M$$

Prämissen:

- 1 Wenn Hans zum Meeting eingeladen wird und nicht auf Dienstreise ist, nimmt er am Meeting teil.
$$\neg R \wedge E \rightarrow M \quad \rightsquigarrow \quad R \vee \neg E \vee M$$
- 2 Wenn der Chef Hans beim Meeting sehen will, lädt er ihn auch ein.
$$C \rightarrow E \quad \rightsquigarrow \quad \neg C \vee E$$

Prämissen:

- 1 Wenn Hans zum Meeting eingeladen wird und nicht auf Dienstreise ist, nimmt er am Meeting teil.
$$\neg R \wedge E \rightarrow M \quad \rightsquigarrow \quad R \vee \neg E \vee M$$
- 2 Wenn der Chef Hans beim Meeting sehen will, lädt er ihn auch ein.
$$C \rightarrow E \quad \rightsquigarrow \quad \neg C \vee E$$
- 3 Wenn der Chef Hans nicht beim Meeting sehen will, wird Hans bald gekündigt.
$$\neg C \rightarrow K \quad \rightsquigarrow \quad C \vee K$$

Prämissen:

- 1 Wenn Hans zum Meeting eingeladen wird und nicht auf Dienstreise ist, nimmt er am Meeting teil.
$$\neg R \wedge E \rightarrow M \quad \rightsquigarrow \quad R \vee \neg E \vee M$$
- 2 Wenn der Chef Hans beim Meeting sehen will, lädt er ihn auch ein.
$$C \rightarrow E \quad \rightsquigarrow \quad \neg C \vee E$$
- 3 Wenn der Chef Hans nicht beim Meeting sehen will, wird Hans bald gekündigt.
$$\neg C \rightarrow K \quad \rightsquigarrow \quad C \vee K$$
- 4 Hans hat nicht am Meeting teilgenommen.
$$\neg M$$

Prämissen:

- 1 Wenn Hans zum Meeting eingeladen wird und nicht auf Dienstreise ist, nimmt er am Meeting teil.
$$\neg R \wedge E \rightarrow M \quad \rightsquigarrow \quad R \vee \neg E \vee M$$
- 2 Wenn der Chef Hans beim Meeting sehen will, lädt er ihn auch ein.
$$C \rightarrow E \quad \rightsquigarrow \quad \neg C \vee E$$
- 3 Wenn der Chef Hans nicht beim Meeting sehen will, wird Hans bald gekündigt.
$$\neg C \rightarrow K \quad \rightsquigarrow \quad C \vee K$$
- 4 Hans hat nicht am Meeting teilgenommen.
$$\neg M$$
- 5 Hans ist nicht auf einer Dienstreise.
$$\neg R$$

Prämissen:

- 1 Wenn Hans zum Meeting eingeladen wird und nicht auf Dienstreise ist, nimmt er am Meeting teil. $\neg R \wedge E \rightarrow M \rightsquigarrow R \vee \neg E \vee M$
- 2 Wenn der Chef Hans beim Meeting sehen will, lädt er ihn auch ein. $C \rightarrow E \rightsquigarrow \neg C \vee E$
- 3 Wenn der Chef Hans nicht beim Meeting sehen will, wird Hans bald gekündigt. $\neg C \rightarrow K \rightsquigarrow C \vee K$
- 4 Hans hat nicht am Meeting teilgenommen. $\neg M$
- 5 Hans ist nicht auf einer Dienstreise. $\neg R$

Konklusion:

- 6 Hans wird bald gekündigt. K

- Implizieren die **Prämissen** 1–5 die **Konklusion** 6?

- Implizieren die **Prämissen** 1–5 die **Konklusion** 6?
 - Logisch: Ist **jedes** Modell von 1–5 auch ein Modell von 6?
 $1 \wedge 2 \wedge 3 \wedge 4 \wedge 5 \models 6$?

- Implizieren die **Prämissen** 1–5 die **Konklusion** 6?
 - Logisch: Ist **jedes** Modell von 1–5 auch ein Modell von 6?
 $1 \wedge 2 \wedge 3 \wedge 4 \wedge 5 \models 6$?
- Umgekehrt: **Existiert** ein Modell von 1–5, in dem 6 **nicht** gilt?

- Implizieren die **Prämissen** 1–5 die **Konklusion** 6?
 - Logisch: Ist **jedes** Modell von 1–5 auch ein Modell von 6?
 $1 \wedge 2 \wedge 3 \wedge 4 \wedge 5 \models 6$?
- Umgekehrt: **Existiert** ein Modell von 1–5, im dem 6 **nicht** gilt?
 - Logisch: Ist $1 \wedge 2 \wedge 3 \wedge 4 \wedge 5 \wedge \neg 6$ erfüllbar?
Wenn ja, gilt die Implikation **nicht**.

- Implizieren die **Prämissen** 1–5 die **Konklusion** 6?
 - Logisch: Ist **jedes** Modell von 1–5 auch ein Modell von 6?
 $1 \wedge 2 \wedge 3 \wedge 4 \wedge 5 \models 6$?
- Umgekehrt: **Existiert** ein Modell von 1–5, im dem 6 **nicht** gilt?
 - Logisch: Ist $1 \wedge 2 \wedge 3 \wedge 4 \wedge 5 \wedge \neg 6$ erfüllbar?
Wenn ja, gilt die Implikation **nicht**.
 - Ist $1 \wedge 2 \wedge 3 \wedge 4 \wedge 5 \wedge \neg 6$ **unerfüllbar**, gilt die Implikation.

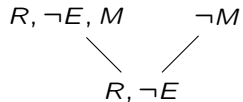
- Implizieren die **Prämissen** 1–5 die **Konklusion** 6?
 - Logisch: Ist **jedes** Modell von 1–5 auch ein Modell von 6?
 $1 \wedge 2 \wedge 3 \wedge 4 \wedge 5 \models 6$
- Umgekehrt: **Existiert** ein Modell von 1–5, im dem 6 **nicht** gilt?
 - Logisch: Ist $1 \wedge 2 \wedge 3 \wedge 4 \wedge 5 \wedge \neg 6$ erfüllbar?
Wenn ja, gilt die Implikation **nicht**.
 - Ist $1 \wedge 2 \wedge 3 \wedge 4 \wedge 5 \wedge \neg 6$ **unerfüllbar**, gilt die Implikation.
- Teste mittels Resolution die Erfüllbarkeit von

- Implizieren die **Prämissen** 1–5 die **Konklusion** 6?
 - Logisch: Ist **jedes** Modell von 1–5 auch ein Modell von 6?
 $1 \wedge 2 \wedge 3 \wedge 4 \wedge 5 \models 6$
- Umgekehrt: **Existiert** ein Modell von 1–5, im dem 6 **nicht** gilt?
 - Logisch: Ist $1 \wedge 2 \wedge 3 \wedge 4 \wedge 5 \wedge \neg 6$ erfüllbar?
Wenn ja, gilt die Implikation **nicht**.
 - Ist $1 \wedge 2 \wedge 3 \wedge 4 \wedge 5 \wedge \neg 6$ **unerfüllbar**, gilt die Implikation.
- Teste mittels Resolution die Erfüllbarkeit von

- Implizieren die **Prämissen** 1–5 die **Konklusion** 6?
 - Logisch: Ist **jedes** Modell von 1–5 auch ein Modell von 6?
 $1 \wedge 2 \wedge 3 \wedge 4 \wedge 5 \models 6$?
- Umgekehrt: **Existiert** ein Modell von 1–5, im dem 6 **nicht** gilt?
 - Logisch: Ist $1 \wedge 2 \wedge 3 \wedge 4 \wedge 5 \wedge \neg 6$ erfüllbar?
Wenn ja, gilt die Implikation **nicht**.
 - Ist $1 \wedge 2 \wedge 3 \wedge 4 \wedge 5 \wedge \neg 6$ **unerfüllbar**, gilt die Implikation.
- Teste mittels Resolution die Erfüllbarkeit von

$$\{\{R, \neg E, M\}, \{\neg C, E\}, \{C, K\}, \{\neg M\}, \{\neg R\}, \{\neg K\}\}$$

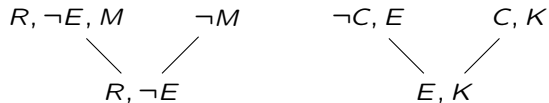
$\{\{R, \neg E, M\}, \{\neg C, E\}, \{C, K\}, \{\neg M\}, \{\neg R\}, \{\neg K\}\}$



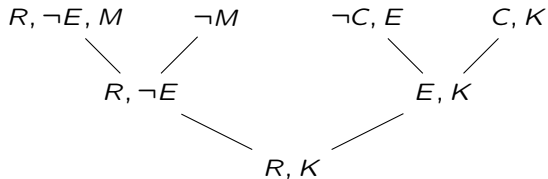
$\{\{R, \neg E, M\}, \{\neg C, E\}, \{C, K\}, \{\neg M\}, \{\neg R\}, \{\neg K\}\}$



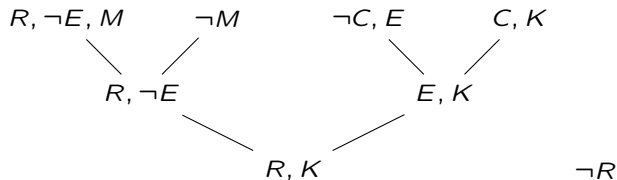
$\{\{R, \neg E, M\}, \{\neg C, E\}, \{C, K\}, \{\neg M\}, \{\neg R\}, \{\neg K\}\}$



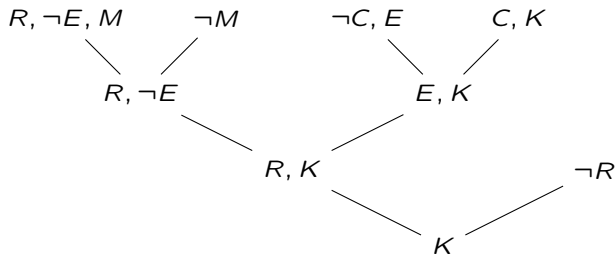
$\{\{R, \neg E, M\}, \{\neg C, E\}, \{C, K\}, \{\neg M\}, \{\neg R\}, \{\neg K\}\}$



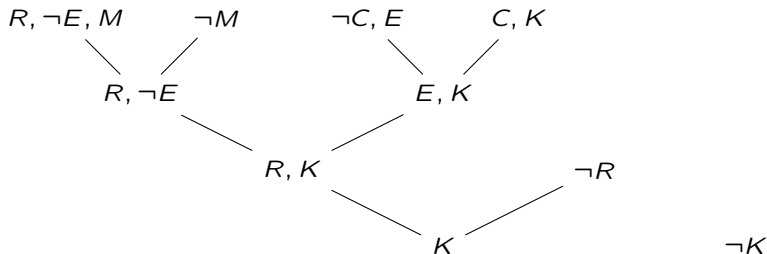
$\{\{R, \neg E, M\}, \{\neg C, E\}, \{C, K\}, \{\neg M\}, \{\neg R\}, \{\neg K\}\}$



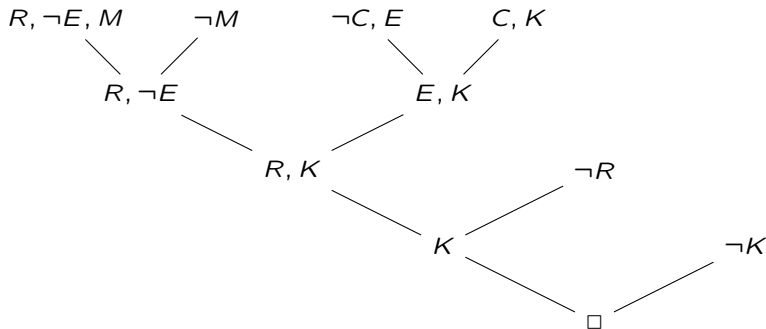
$\{\{R, \neg E, M\}, \{\neg C, E\}, \{C, K\}, \{\neg M\}, \{\neg R\}, \{\neg K\}\}$



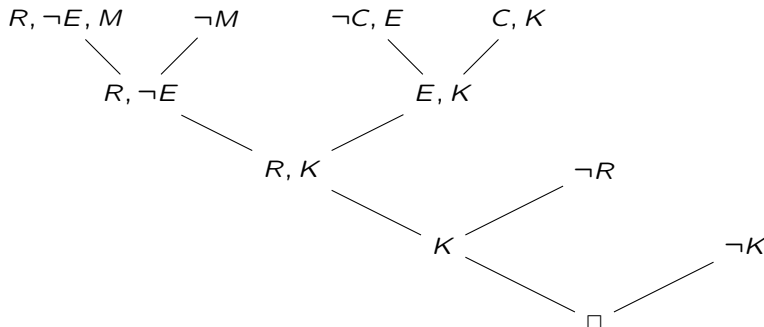
$\{\{R, \neg E, M\}, \{\neg C, E\}, \{C, K\}, \{\neg M\}, \{\neg R\}, \{\neg K\}\}$



$\{\{R, \neg E, M\}, \{\neg C, E\}, \{C, K\}, \{\neg M\}, \{\neg R\}, \{\neg K\}\}$



$\{\{R, \neg E, M\}, \{\neg C, E\}, \{C, K\}, \{\neg M\}, \{\neg R\}, \{\neg K\}\}$



Die Folgerung gilt, d. h. Hans wird leider entlassen. . .

Test mit Wahrheitstabelle

R	E	M	C	K	1 ($R, \neg E, M$)	2 ($\neg C, E$)	3 (C, K)	4 ($\neg M$)	5 ($\neg R$)	6 (K)	$1 \wedge \dots \wedge 5 \wedge \neg 6$

Test mit Wahrheitstabelle

R	E	M	C	K	1 ($R, \neg E, M$)	2 ($\neg C, E$)	3 (C, K)	4 ($\neg M$)	5 ($\neg R$)	6 (K)	$1 \wedge \dots \wedge 5 \wedge \neg 6$
0	0	0	0	0	1	1	0	1	1	0	0

Test mit Wahrheitstabelle

R	E	M	C	K	1 ($R, \neg E, M$)	2 ($\neg C, E$)	3 (C, K)	4 ($\neg M$)	5 ($\neg R$)	6 (K)	$1 \wedge \dots \wedge 5 \wedge \neg 6$
0	0	0	0	0	1	1	0	1	1	0	0
0	0	0	0	1	1	1	1	1	1	1	0
		...					...				...
1	0	1	0	0	1	1	0	0	0	0	0
		...					...				...
1	1	1	1	1	1	1	1	0	0	1	0

Test mit Wahrheitstabelle

R	E	M	C	K	1 ($R, \neg E, M$)	2 ($\neg C, E$)	3 (C, K)	4 ($\neg M$)	5 ($\neg R$)	6 (K)	$1 \wedge \dots \wedge 5 \wedge \neg 6$
0	0	0	0	0	1	1	0	1	1	0	0
0	0	0	0	1	1	1	1	1	1	1	0
		...					...				...
1	0	1	0	0	1	1	0	0	0	0	0
		...					...				...
1	1	1	1	1	1	1	1	0	0	1	0

- Die einzige Interpretation, die die Klauseln 1–5 wahr macht (Zeile 2), macht auch Klausel 6 wahr.
- Die letzte Spalte ist immer 0.
- Die Konklusion folgt aus den Prämissen.

Wenn mehrere Resolventen möglich sind:

- Nicht jeder Weg ist gleich schnell

Wenn mehrere Resolventen möglich sind:

- Nicht jeder Weg ist gleich schnell
- ... aber alle Wege führen zum Ziel.

Wenn mehrere Resolventen möglich sind:

- Nicht jeder Weg ist gleich schnell
- ... aber alle Wege führen zum Ziel.

Definition 3.46 (Don't-care-Nichtdeterminismus)

Ein Algorithmus heißt **don't-care-nichtdeterministisch**, wenn jede von mehreren möglichen nichtdeterministischen Entscheidungen zum korrekten Ergebnis führt.

Wenn mehrere Resolventen möglich sind:

- Nicht jeder Weg ist gleich schnell
- ... aber alle Wege führen zum Ziel.

Definition 3.46 (Don't-care-Nichtdeterminismus)

Ein Algorithmus heißt **don't-care-nichtdeterministisch**, wenn jede von mehreren möglichen nichtdeterministischen Entscheidungen zum korrekten Ergebnis führt.

In don't-care-nichtdeterministischen Algorithmen müssen einmal getroffene Entscheidungen nie rückgängig gemacht werden (Backtracking).

Auswahl der Elternklauseln ist entscheidend für Effizienz

Auswahl der Elternklauseln ist entscheidend für Effizienz

Optimierungen:

- Ignoriere **Tautologien** $\{X, \neg X, \dots\}$

Auswahl der Elternklauseln ist entscheidend für Effizienz

Optimierungen:

- Ignoriere **Tautologien** $\{X, \neg X, \dots\}$
- Ignoriere C_1 , wenn ein $C_2 \subset C_1$ existiert

Auswahl der Elternklauseln ist entscheidend für Effizienz

Optimierungen:

- Ignoriere **Tautologien** $\{X, \neg X, \dots\}$
- Ignoriere C_1 , wenn ein $C_2 \subset C_1$ existiert

Heuristiken:

- Priorisiere kleine Klauseln

Auswahl der Elternklauseln ist entscheidend für Effizienz

Optimierungen:

- Ignoriere **Tautologien** $\{X, \neg X, \dots\}$
- Ignoriere C_1 , wenn ein $C_2 \subset C_1$ existiert

Heuristiken:

- Priorisiere kleine Klauseln
- Priorisiere Klauseln mit hoher **Ableitungstiefe** $\rightsquigarrow$ Depth-first-Suche

Auswahl der Elternklauseln ist entscheidend für Effizienz

Optimierungen:

- Ignoriere **Tautologien** $\{X, \neg X, \dots\}$
- Ignoriere C_1 , wenn ein $C_2 \subset C_1$ existiert

Heuristiken:

- Priorisiere kleine Klauseln
- Priorisiere Klauseln mit hoher **Ableitungstiefe** $\rightsquigarrow$ Depth-first-Suche

Einschränkung auf Spezialfälle, z. B. Horn-Formeln

- **Horn-Klausel**: Höchstens ein positives Literal

Auswahl der Elternklauseln ist entscheidend für Effizienz

Optimierungen:

- Ignoriere **Tautologien** $\{X, \neg X, \dots\}$
- Ignoriere C_1 , wenn ein $C_2 \subset C_1$ existiert

Heuristiken:

- Priorisiere kleine Klauseln
- Priorisiere Klauseln mit hoher **Ableitungstiefe** $\rightsquigarrow$ Depth-first-Suche

Einschränkung auf Spezialfälle, z. B. Horn-Formeln

- **Horn-Klausel**: Höchstens ein positives Literal
- Horn-Regel: $X \wedge Y \wedge Z \wedge \dots \rightarrow W$
- nützlich für Regel-basiertes Schließen (**Prolog**)

Auswahl der Elternklauseln ist entscheidend für Effizienz

Optimierungen:

- Ignoriere **Tautologien** $\{X, \neg X, \dots\}$
- Ignoriere C_1 , wenn ein $C_2 \subset C_1$ existiert

Heuristiken:

- Priorisiere kleine Klauseln
- Priorisiere Klauseln mit hoher **Ableitungstiefe** $\rightsquigarrow$ Depth-first-Suche

Einschränkung auf Spezialfälle, z. B. Horn-Formeln

- **Horn-Klausel**: Höchstens ein positives Literal
- Horn-Regel: $X \wedge Y \wedge Z \wedge \dots \rightarrow W$
- nützlich für Regel-basiertes Schließen (**Prolog**)
- Laufzeit: quadratisch (n^2 Schritte)

Auswahl der Elternklauseln ist entscheidend für Effizienz

Optimierungen:

- Ignoriere **Tautologien** $\{X, \neg X, \dots\}$
- Ignoriere C_1 , wenn ein $C_2 \subset C_1$ existiert

Heuristiken:

- Priorisiere kleine Klauseln
- Priorisiere Klauseln mit hoher **Ableitungstiefe** $\rightsquigarrow$ Depth-first-Suche

Einschränkung auf Spezialfälle, z. B. Horn-Formeln

- **Horn-Klausel**: Höchstens ein positives Literal
- Horn-Regel: $X \wedge Y \wedge Z \wedge \dots \rightarrow W$
- nützlich für Regel-basiertes Schließen (**Prolog**)
- Laufzeit: quadratisch (n^2 Schritte)
 - mit optimierten Datenstrukturen: linear (n Schritte)

Übung 3.47

Für einen Juwelenraub gibt es genau drei Verdächtige: Anna, Bert und Claus. Über sie ist das Folgende bekannt (**Prämissen**):

- 1 Mindestens einer der Verdächtigen ist schuldig.
- 2 Wenn Anna schuldig ist, hatte sie genau einen der Verdächtigen als Komplizen.
- 3 Wenn Bert unschuldig ist, dann auch Claus.
- 4 Wenn genau zwei Verdächtige schuldig sind, ist Claus einer von ihnen.
- 5 Wenn Claus unschuldig ist, ist Anna schuldig.

Inspektor Craig vermutet (**Konklusion**):

- 6 Bert und Claus sind schuldig.
-
- a Formalisieren Sie die Aussagen in Aussagenlogik und bilden Sie die Konjunktion von Prämissen und Negation der Konklusion.
 - b Transformieren Sie die sich ergebende Formel in die KNF.
 - c Testen Sie durch Resolution, ob die Konklusion aus den Prämissen folgt.

1. Einführung

2. Mengen

3. Aussagenlogik

3.1 Syntax

3.2 Semantik

3.3 Schlussfolgerungsverfahren

3.3.1 Resolution

3.3.2 Tableaus

4. Prädikatenlogik

5. Prolog

- Erfüllbarkeitstest für aussagenlogische Formeln
- Evert Willem Beth: *Semantic entailment and formal derivability* (1955)



E. W. Beth
(1908–1964)

- Erfüllbarkeitstest für aussagenlogische Formeln
- Evert Willem Beth: *Semantic entailment and formal derivability* (1955)



E. W. Beth
(1908–1964)

Anderer Ansatz für Erfüllbarkeitstest

Resolution: suche einen Widerspruch

- Erfüllbarkeitstest für aussagenlogische Formeln
- Evert Willem Beth: *Semantic entailment and formal derivability* (1955)



E. W. Beth
(1908–1964)

Anderer Ansatz für Erfüllbarkeitstest

Resolution: suche einen Widerspruch

Tableau: suche ein Modell

- Erfüllbarkeitstest für aussagenlogische Formeln
- Evert Willem Beth: *Semantic entailment and formal derivability* (1955)



E. W. Beth
(1908–1964)

Anderer Ansatz für Erfüllbarkeitstest

Resolution: suche einen Widerspruch

Tableau: suche ein Modell

- erzeugt Tabellen-artige Struktur, bei der Spalten aufgeteilt werden
 $\rightsquigarrow$ „Tableau“

- Erfüllbarkeitstest für aussagenlogische Formeln
- Evert Willem Beth: *Semantic entailment and formal derivability* (1955)



E. W. Beth
(1908–1964)

Anderer Ansatz für Erfüllbarkeitstest

Resolution: suche einen Widerspruch

Tableau: suche ein Modell

- erzeugt Tabellen-artige Struktur, bei der Spalten aufgeteilt werden
 $\rightsquigarrow$ „Tableau“
- andere Sichtweise: Baumstruktur

- Jede Spalte n steht für eine Interpretation $\mathcal{I}_n$

- Jede **Spalte** n steht für eine Interpretation $\mathcal{I}_n$
- Jede **Zeile** einer Spalte enthält eine Formel, die $\mathcal{I}_n$ erfüllen muss

- Jede **Spalte** n steht für eine Interpretation $\mathcal{I}_n$
- Jede **Zeile** einer Spalte enthält eine Formel, die $\mathcal{I}_n$ erfüllen muss
- Beginne mit der Ausgangsformel χ und einer Spalte

- Jede **Spalte** n steht für eine Interpretation $\mathcal{I}_n$
- Jede **Zeile** einer Spalte enthält eine Formel, die $\mathcal{I}_n$ erfüllen muss
- Beginne mit der Ausgangsformel χ und einer Spalte
- **Regeln** brechen komplexe Formeln auf einfachere herunter
„Wenn $\varphi \wedge \psi$ vorhanden ist, aber nicht φ und ψ , füge φ und ψ hinzu.“

- Jede **Spalte** n steht für eine Interpretation $\mathcal{I}_n$
- Jede **Zeile** einer Spalte enthält eine Formel, die $\mathcal{I}_n$ erfüllen muss
- Beginne mit der Ausgangsformel χ und einer Spalte
- **Regeln** brechen komplexe Formeln auf einfachere herunter
„Wenn $\varphi \wedge \psi$ vorhanden ist, aber nicht φ und ψ , füge φ und ψ hinzu.“
 - Bei mehreren Möglichkeiten: Teile Spalte auf

- Jede **Spalte** n steht für eine Interpretation $\mathcal{I}_n$
- Jede **Zeile** einer Spalte enthält eine Formel, die $\mathcal{I}_n$ erfüllen muss
- Beginne mit der Ausgangsformel χ und einer Spalte
- **Regeln** brechen komplexe Formeln auf einfachere herunter
„Wenn $\varphi \wedge \psi$ vorhanden ist, aber nicht φ und ψ , füge φ und ψ hinzu.“
 - Bei mehreren Möglichkeiten: Teile Spalte auf
- **Clash** beschreibt Situation, in denen ein Widerspruch vorliegt
„ φ und $\neg\varphi$ sind vorhanden.“

- Jede **Spalte** n steht für eine Interpretation $\mathcal{I}_n$
- Jede **Zeile** einer Spalte enthält eine Formel, die $\mathcal{I}_n$ erfüllen muss
- Beginne mit der Ausgangsformel χ und einer Spalte
- **Regeln** brechen komplexe Formeln auf einfachere herunter
„Wenn $\varphi \wedge \psi$ vorhanden ist, aber nicht φ und ψ , füge φ und ψ hinzu.“
 - Bei mehreren Möglichkeiten: Teile Spalte auf
- **Clash** beschreibt Situation, in denen ein Widerspruch vorliegt
„ φ und $\neg\varphi$ sind vorhanden.“
 - Clashes gibt es zwischen Formeln in der aktuellen Spalte und den **darüberliegenden**

- Jede **Spalte** n steht für eine Interpretation $\mathcal{I}_n$
- Jede **Zeile** einer Spalte enthält eine Formel, die $\mathcal{I}_n$ erfüllen muss
- Beginne mit der Ausgangsformel χ und einer Spalte
- **Regeln** brechen komplexe Formeln auf einfachere herunter
„Wenn $\varphi \wedge \psi$ vorhanden ist, aber nicht φ und ψ , füge φ und ψ hinzu.“
 - Bei mehreren Möglichkeiten: Teile Spalte auf
- **Clash** beschreibt Situation, in denen ein Widerspruch vorliegt
„ φ und $\neg\varphi$ sind vorhanden.“
 - Clashes gibt es zwischen Formeln in der aktuellen Spalte und den **darüberliegenden**
- Ergebnis:

- Jede **Spalte** n steht für eine Interpretation $\mathcal{I}_n$
- Jede **Zeile** einer Spalte enthält eine Formel, die $\mathcal{I}_n$ erfüllen muss
- Beginne mit der Ausgangsformel χ und einer Spalte
- **Regeln** brechen komplexe Formeln auf einfachere herunter
„Wenn $\varphi \wedge \psi$ vorhanden ist, aber nicht φ und ψ , füge φ und ψ hinzu.“
 - Bei mehreren Möglichkeiten: Teile Spalte auf
- **Clash** beschreibt Situation, in denen ein Widerspruch vorliegt
„ φ und $\neg\varphi$ sind vorhanden.“
 - Clashes gibt es zwischen Formeln in der aktuellen Spalte und den **darüberliegenden**
- Ergebnis:
 - Wenn jede Spalte einen Clash enthält, gibt es kein Modell
 $\leadsto \chi$ ist unerfüllbar

- Jede **Spalte** n steht für eine Interpretation $\mathcal{I}_n$
- Jede **Zeile** einer Spalte enthält eine Formel, die $\mathcal{I}_n$ erfüllen muss
- Beginne mit der Ausgangsformel χ und einer Spalte
- **Regeln** brechen komplexe Formeln auf einfachere herunter
„Wenn $\varphi \wedge \psi$ vorhanden ist, aber nicht φ und ψ , füge φ und ψ hinzu.“
 - Bei mehreren Möglichkeiten: Teile Spalte auf
- **Clash** beschreibt Situation, in denen ein Widerspruch vorliegt
„ φ und $\neg\varphi$ sind vorhanden.“
 - Clashes gibt es zwischen Formeln in der aktuellen Spalte und den **darüberliegenden**
- Ergebnis:
 - Wenn jede Spalte einen Clash enthält, gibt es kein Modell
 $\leadsto \chi$ ist unerfüllbar
 - Wenn keine Regel mehr anwendbar ist (Tableau ist **vollständig**) und mindestens eine Spalte Clash-frei ist, gibt es ein Modell
 $\leadsto \chi$ ist erfüllbar

Vorbedingung „Wenn $\varphi \wedge \psi$ vorhanden ist. . .“

Vorbedingung „Wenn $\varphi \wedge \psi$ vorhanden ist. . .“

- Welche Formel ist zu verarbeiten?

Vorbedingung „Wenn $\varphi \wedge \psi$ vorhanden ist...“

- Welche Formel ist zu verarbeiten?

Anwendbarkeitsbedingung „... aber nicht φ und ψ , ...“

Vorbedingung „Wenn $\varphi \wedge \psi$ vorhanden ist...“

- Welche Formel ist zu verarbeiten?

Anwendbarkeitsbedingung „... aber nicht φ und ψ , ...“

- Wann ist Regel nicht mehr anwendbar?
($\rightsquigarrow$ Terminierung)

Vorbedingung „Wenn $\varphi \wedge \psi$ vorhanden ist...“

- Welche Formel ist zu verarbeiten?

Anwendbarkeitsbedingung „... aber nicht φ und ψ , ...“

- Wann ist Regel nicht mehr anwendbar?
($\rightsquigarrow$ Terminierung)
- bezieht sich auf aktuelle Spalte und darüberliegende

Vorbedingung „Wenn $\varphi \wedge \psi$ vorhanden ist...“

- Welche Formel ist zu verarbeiten?

Anwendbarkeitsbedingung „... aber nicht φ und ψ , ...“

- Wann ist Regel nicht mehr anwendbar?
($\rightsquigarrow$ Terminierung)
- bezieht sich auf aktuelle Spalte und darüberliegende

Nachbedingung „... füge φ und ψ hinzu.“

Vorbedingung „Wenn $\varphi \wedge \psi$ vorhanden ist...“

- Welche Formel ist zu verarbeiten?

Anwendbarkeitsbedingung „... aber nicht φ und ψ , ...“

- Wann ist Regel nicht mehr anwendbar?
($\rightsquigarrow$ Terminierung)
- bezieht sich auf aktuelle Spalte und darüberliegende

Nachbedingung „... füge φ und ψ hinzu.“

- Wie ist Formel zu verarbeiten?

Vorbedingung „Wenn $\varphi \wedge \psi$ vorhanden ist...“

- Welche Formel ist zu verarbeiten?

Anwendbarkeitsbedingung „... aber nicht φ und ψ , ...“

- Wann ist Regel nicht mehr anwendbar?
($\rightsquigarrow$ Terminierung)
- bezieht sich auf aktuelle Spalte und darüberliegende

Nachbedingung „... füge φ und ψ hinzu.“

- Wie ist Formel zu verarbeiten?

Vorbedingung „Wenn $\varphi \wedge \psi$ vorhanden ist. . .“

- Welche Formel ist zu verarbeiten?

Anwendbarkeitsbedingung „. . . aber nicht φ und ψ , . . .“

- Wann ist Regel nicht mehr anwendbar?
($\rightsquigarrow$ Terminierung)
- bezieht sich auf aktuelle Spalte und darüberliegende

Nachbedingung „. . . füge φ und ψ hinzu.“

- Wie ist Formel zu verarbeiten?

Regeln können nicht-deterministisch sein:

- „Wenn $\varphi \vee \psi$ enthalten ist. . . , füge φ **oder** ψ hinzu“
- Teile aktuelle Spalte, teste φ in einer Spalte, ψ in der anderen

Definition 3.48 (Negations-Normalform)

Eine aussagenlogische Formel ist in **Negations-Normalform (NNF)** wenn als binäre Junktoren nur $\wedge$ und $\vee$ enthalten sind und Negation nur direkt vor Aussagenvariablen vorkommt.

Definition 3.48 (Negations-Normalform)

Eine aussagenlogische Formel ist in **Negations-Normalform (NNF)** wenn als binäre Junktoren nur $\wedge$ und $\vee$ enthalten sind und Negation nur direkt vor Aussagenvariablen vorkommt.

Satz 3.49

Jede AL-Formel kann in NNF transformiert werden.

Beweis.

Verfahren:

- 1 Elimination von $\leftrightarrow$ und $\rightarrow$

Definition 3.48 (Negations-Normalform)

Eine aussagenlogische Formel ist in **Negations-Normalform (NNF)** wenn als binäre Junktoren nur $\wedge$ und $\vee$ enthalten sind und Negation nur direkt vor Aussagenvariablen vorkommt.

Satz 3.49

Jede AL-Formel kann in NNF transformiert werden.

Beweis.

Verfahren:

- 1 Elimination von $\leftrightarrow$ und $\rightarrow$
- 2 Anwendung der Gesetze von De Morgan

Definition 3.48 (Negations-Normalform)

Eine aussagenlogische Formel ist in **Negations-Normalform (NNF)** wenn als binäre Junktoren nur $\wedge$ und $\vee$ enthalten sind und Negation nur direkt vor Aussagenvariablen vorkommt.

Satz 3.49

Jede AL-Formel kann in NNF transformiert werden.

Beweis.

Verfahren:

- 1 Elimination von $\leftrightarrow$ und $\rightarrow$
- 2 Anwendung der Gesetze von De Morgan
- 3 Entfernung doppelter Negation

Definition 3.48 (Negations-Normalform)

Eine aussagenlogische Formel ist in **Negations-Normalform (NNF)** wenn als binäre Junktoren nur $\wedge$ und $\vee$ enthalten sind und Negation nur direkt vor Aussagenvariablen vorkommt.

Satz 3.49

Jede AL-Formel kann in NNF transformiert werden.

Beweis.

Verfahren:

- 1 Elimination von $\leftrightarrow$ und $\rightarrow$
- 2 Anwendung der Gesetze von De Morgan
- 3 Entfernung doppelter Negation

Definition 3.48 (Negations-Normalform)

Eine aussagenlogische Formel ist in **Negations-Normalform (NNF)** wenn als binäre Junktoren nur $\wedge$ und $\vee$ enthalten sind und Negation nur direkt vor Aussagenvariablen vorkommt.

Satz 3.49

Jede AL-Formel kann in NNF transformiert werden.

Beweis.

Verfahren:

- 1 Elimination von $\leftrightarrow$ und $\rightarrow$
- 2 Anwendung der Gesetze von De Morgan
- 3 Entfernung doppelter Negation

(Wie für KNF.)



$\wedge$ -Regel Wenn es ein S_i gibt mit $\varphi \wedge \psi \in S_i$
und $\{\varphi, \psi\} \not\subseteq S_i$
dann $S_i := S_i \cup \{\varphi, \psi\}$

$\wedge$ -Regel Wenn es ein S_i gibt mit $\varphi \wedge \psi \in S_i$
und $\{\varphi, \psi\} \not\subseteq S_i$
dann $S_i := S_i \cup \{\varphi, \psi\}$

$\vee$ -Regel Wenn es ein S_i gibt mit $\varphi \vee \psi \in S_i$
und $\{\varphi, \psi\} \cap S_i = \emptyset$
dann $S_i := S_i \cup \{\varphi\}$ und $S_{\text{neu}} := S_i \cup \{\psi\}$

$\wedge$ -Regel Wenn es ein S_i gibt mit $\varphi \wedge \psi \in S_i$
und $\{\varphi, \psi\} \not\subseteq S_i$
dann $S_i := S_i \cup \{\varphi, \psi\}$

$\vee$ -Regel Wenn es ein S_i gibt mit $\varphi \vee \psi \in S_i$
und $\{\varphi, \psi\} \cap S_i = \emptyset$
dann $S_i := S_i \cup \{\varphi\}$ und $S_{\text{neu}} := S_i \cup \{\psi\}$

$\wedge$ -Regel Wenn es ein S_i gibt mit $\varphi \wedge \psi \in S_i$
und $\{\varphi, \psi\} \not\subseteq S_i$
dann $S_i := S_i \cup \{\varphi, \psi\}$

$\vee$ -Regel Wenn es ein S_i gibt mit $\varphi \vee \psi \in S_i$
und $\{\varphi, \psi\} \cap S_i = \emptyset$
dann $S_i := S_i \cup \{\varphi\}$ und $S_{\text{neu}} := S_i \cup \{\psi\}$

Eingabe: AL-Formel φ

Ausgabe: „erfüllbar“ oder „unerfüllbar“

$\wedge$ -Regel Wenn es ein S_i gibt mit $\varphi \wedge \psi \in S_i$
und $\{\varphi, \psi\} \not\subseteq S_i$
dann $S_i := S_i \cup \{\varphi, \psi\}$

$\vee$ -Regel Wenn es ein S_i gibt mit $\varphi \vee \psi \in S_i$
und $\{\varphi, \psi\} \cap S_i = \emptyset$
dann $S_i := S_i \cup \{\varphi\}$ und $S_{\text{neu}} := S_i \cup \{\psi\}$

Eingabe: AL-Formel φ

Ausgabe: „erfüllbar“ oder „unerfüllbar“

- 1: $\varphi' := \text{nnf}(\varphi)$
- 2: initialisiere S_0 mit $\{\varphi'\}$

$\wedge$ -Regel Wenn es ein S_i gibt mit $\varphi \wedge \psi \in S_i$
und $\{\varphi, \psi\} \not\subseteq S_i$
dann $S_i := S_i \cup \{\varphi, \psi\}$

$\vee$ -Regel Wenn es ein S_i gibt mit $\varphi \vee \psi \in S_i$
und $\{\varphi, \psi\} \cap S_i = \emptyset$
dann $S_i := S_i \cup \{\varphi\}$ und $S_{\text{neu}} := S_i \cup \{\psi\}$

Eingabe: AL-Formel φ

Ausgabe: „erfüllbar“ oder „unerfüllbar“

- 1: $\varphi' := \text{nnf}(\varphi)$
- 2: initialisiere S_0 mit $\{\varphi'\}$
- 3: **while** eine Regel R ist auf ein $\psi \in S_i$ anwendbar **do**
- 4: wende R auf ψ an

$\wedge$ -Regel Wenn es ein S_i gibt mit $\varphi \wedge \psi \in S_i$
und $\{\varphi, \psi\} \not\subseteq S_i$
dann $S_i := S_i \cup \{\varphi, \psi\}$

$\vee$ -Regel Wenn es ein S_i gibt mit $\varphi \vee \psi \in S_i$
und $\{\varphi, \psi\} \cap S_i = \emptyset$
dann $S_i := S_i \cup \{\varphi\}$ und $S_{\text{neu}} := S_i \cup \{\psi\}$

Eingabe: AL-Formel φ

Ausgabe: „erfüllbar“ oder „unerfüllbar“

- 1: $\varphi' := \text{nnf}(\varphi)$
- 2: initialisiere S_0 mit $\{\varphi'\}$
- 3: **while** eine Regel R ist auf ein $\psi \in S_i$ anwendbar **do**
- 4: wende R auf ψ an
- 5: **if** jedes S_i enthält einen Clash **then**
- 6: **return** „unerfüllbar“

$\wedge$ -Regel Wenn es ein S_i gibt mit $\varphi \wedge \psi \in S_i$
und $\{\varphi, \psi\} \not\subseteq S_i$
dann $S_i := S_i \cup \{\varphi, \psi\}$

$\vee$ -Regel Wenn es ein S_i gibt mit $\varphi \vee \psi \in S_i$
und $\{\varphi, \psi\} \cap S_i = \emptyset$
dann $S_i := S_i \cup \{\varphi\}$ und $S_{\text{neu}} := S_i \cup \{\psi\}$

Eingabe: AL-Formel φ

Ausgabe: „erfüllbar“ oder „unerfüllbar“

- 1: $\varphi' := \text{nnf}(\varphi)$
- 2: initialisiere S_0 mit $\{\varphi'\}$
- 3: **while** eine Regel R ist auf ein $\psi \in S_i$ anwendbar **do**
- 4: wende R auf ψ an
- 5: **if** jedes S_i enthält einen Clash **then**
- 6: **return** „unerfüllbar“
- 7: **else**
- 8: **return** „erfüllbar“

Beispiel: Bestimmung des Schicksals von Hans mit Tableau

Prämissen: $(R \vee \neg E \vee M) \wedge (\neg C \vee E) \wedge (C \vee K) \wedge \neg M \wedge \neg R$

Konklusion: K

Zeile		Regel
1	$(R \vee \neg E \vee M) \wedge (\neg C \vee E) \wedge (C \vee K) \wedge \neg M \wedge \neg R \wedge \neg K$	

Beispiel: Bestimmung des Schicksals von Hans mit Tableau

Prämissen: $(R \vee \neg E \vee M) \wedge (\neg C \vee E) \wedge (C \vee K) \wedge \neg M \wedge \neg R$

Konklusion: K

Zeile		Regel
1	$(R \vee \neg E \vee M) \wedge (\neg C \vee E) \wedge (C \vee K) \wedge \neg M \wedge \neg R \wedge \neg K$	Eingabe
2	$R \vee \neg E \vee M$	1: $\wedge$
3	$\neg C \vee E$	
4	$C \vee K$	
5	$\neg M$	
6	$\neg R$	
7	$\neg K$	

Beispiel: Bestimmung des Schicksals von Hans mit Tableau

Prämissen: $(R \vee \neg E \vee M) \wedge (\neg C \vee E) \wedge (C \vee K) \wedge \neg M \wedge \neg R$

Konklusion: K

Zeile		Regel
1	$(R \vee \neg E \vee M) \wedge (\neg C \vee E) \wedge (C \vee K) \wedge \neg M \wedge \neg R \wedge \neg K$	Eingabe
2	$R \vee \neg E \vee M$	1: $\wedge$
3	$\neg C \vee E$	
4	$C \vee K$	
5	$\neg M$	
6	$\neg R$	
7	$\neg K$	
8	C $K \text{ ⚡ 7}$	4: $\vee$

Beispiel: Bestimmung des Schicksals von Hans mit Tableau

Prämissen: $(R \vee \neg E \vee M) \wedge (\neg C \vee E) \wedge (C \vee K) \wedge \neg M \wedge \neg R$

Konklusion: K

Zeile		Regel
1	$(R \vee \neg E \vee M) \wedge (\neg C \vee E) \wedge (C \vee K) \wedge \neg M \wedge \neg R \wedge \neg K$	Eingabe
2	$R \vee \neg E \vee M$	1: $\wedge$
3	$\neg C \vee E$	
4	$C \vee K$	
5	$\neg M$	
6	$\neg R$	
7	$\neg K$	
8	C	4: $\vee$
9	$\neg C \text{ ⚡ 8}$ E $K \text{ ⚡ 7}$	3: $\vee$

Beispiel: Bestimmung des Schicksals von Hans mit Tableau

Prämissen: $(R \vee \neg E \vee M) \wedge (\neg C \vee E) \wedge (C \vee K) \wedge \neg M \wedge \neg R$

Konklusion: K

Zeile		Regel
1	$(R \vee \neg E \vee M) \wedge (\neg C \vee E) \wedge (C \vee K) \wedge \neg M \wedge \neg R \wedge \neg K$	Eingabe
2	$R \vee \neg E \vee M$	1: $\wedge$
3	$\neg C \vee E$	
4	$C \vee K$	
5	$\neg M$	
6	$\neg R$	
7	$\neg K$	
8	C $K \text{ ⚡ 7}$	4: $\vee$
9	$\neg C \text{ ⚡ 8}$ E	3: $\vee$
10	$R \text{ ⚡ 6}$ $\neg E \text{ ⚡ 9}$ $M \text{ ⚡ 5}$	2: $\vee$

Beispiel: Bestimmung des Schicksals von Hans mit Tableau

Prämissen: $(R \vee \neg E \vee M) \wedge (\neg C \vee E) \wedge (C \vee K) \wedge \neg M \wedge \neg R$

Konklusion: K

Zeile		Regel
1	$(R \vee \neg E \vee M) \wedge (\neg C \vee E) \wedge (C \vee K) \wedge \neg M \wedge \neg R \wedge \neg K$	Eingabe
2	$R \vee \neg E \vee M$	1: $\wedge$
3	$\neg C \vee E$	
4	$C \vee K$	
5	$\neg M$	
6	$\neg R$	
7	$\neg K$	
8	C $K \downarrow 7$	4: $\vee$
9	$\neg C \downarrow 8$ E	3: $\vee$
10	$R \downarrow 6$ $\neg E \downarrow 9$ $M \downarrow 5$	2: $\vee$

Jede Spalte enthält einen Clash

Beispiel: Bestimmung des Schicksals von Hans mit Tableau

Prämissen: $(R \vee \neg E \vee M) \wedge (\neg C \vee E) \wedge (C \vee K) \wedge \neg M \wedge \neg R$

Konklusion: K

Zeile		Regel
1	$(R \vee \neg E \vee M) \wedge (\neg C \vee E) \wedge (C \vee K) \wedge \neg M \wedge \neg R \wedge \neg K$	Eingabe
2	$R \vee \neg E \vee M$	1: $\wedge$
3	$\neg C \vee E$	
4	$C \vee K$	
5	$\neg M$	
6	$\neg R$	
7	$\neg K$	
8	C $K \downarrow 7$	4: $\vee$
9	$\neg C \downarrow 8$ E	3: $\vee$
10	$R \downarrow 6$ $\neg E \downarrow 9$ $M \downarrow 5$	2: $\vee$

Jede Spalte enthält einen Clash

$\leadsto$ Formel $1 \wedge 2 \wedge 3 \wedge 4 \wedge 5 \wedge \neg 6$ ist unerfüllbar

Beispiel: Bestimmung des Schicksals von Hans mit Tableau

Prämissen: $(R \vee \neg E \vee M) \wedge (\neg C \vee E) \wedge (C \vee K) \wedge \neg M \wedge \neg R$

Konklusion: K

Zeile		Regel
1	$(R \vee \neg E \vee M) \wedge (\neg C \vee E) \wedge (C \vee K) \wedge \neg M \wedge \neg R \wedge \neg K$	Eingabe
2	$R \vee \neg E \vee M$	1: $\wedge$
3	$\neg C \vee E$	
4	$C \vee K$	
5	$\neg M$	
6	$\neg R$	
7	$\neg K$	
8	C	4: $\vee$
9	$\neg C$ ⚡ 8	3: $\vee$
10	R ⚡ 6 $\neg E$ ⚡ 9 M ⚡ 5	2: $\vee$

Jede Spalte enthält einen Clash

⇒ Formel $1 \wedge 2 \wedge 3 \wedge 4 \wedge 5 \wedge \neg 6$ ist unerfüllbar

⇒ Prämissen 1–5 implizieren die Konklusion 6

- **Vorrang** beachten: Formeln von „außen“ nach „innen“ abarbeiten

- **Vorrang** beachten: Formeln von „außen“ nach „innen“ abarbeiten
- **Clash**: nur zwischen aktueller Spalte und den darüberliegenden

- **Vorrang** beachten: Formeln von „außen“ nach „innen“ abarbeiten
- **Clash**: nur zwischen aktueller Spalte und den darüberliegenden
 - Spalte steht logisch für Menge aller Formeln, die in ihr und den darüberliegenden Spalten enthalten sind

- **Vorrang** beachten: Formeln von „außen“ nach „innen“ abarbeiten
- **Clash**: nur zwischen aktueller Spalte und den darüberliegenden
 - Spalte steht logisch für Menge aller Formeln, die in ihr und den darüberliegenden Spalten enthalten sind
- Effizienz: Wenn mehrere Regeln anwendbar sind: **$\wedge$ -Regel zuerst**

- **Vorrang** beachten: Formeln von „außen“ nach „innen“ abarbeiten
- **Clash**: nur zwischen aktueller Spalte und den darüberliegenden
 - Spalte steht logisch für Menge aller Formeln, die in ihr und den darüberliegenden Spalten enthalten sind
- **Effizienz**: Wenn mehrere Regeln anwendbar sind: **$\wedge$ -Regel zuerst**
 - sonst: $\wedge$ -Regel in jeder neuen Spalte anwendbar $\rightsquigarrow$ Ineffizienz

- **Vorrang** beachten: Formeln von „außen“ nach „innen“ abarbeiten
- **Clash**: nur zwischen aktueller Spalte und den darüberliegenden
 - Spalte steht logisch für Menge aller Formeln, die in ihr und den darüberliegenden Spalten enthalten sind
- **Effizienz**: Wenn mehrere Regeln anwendbar sind: **$\wedge$ -Regel zuerst**
 - sonst: $\wedge$ -Regel in jeder neuen Spalte anwendbar $\rightsquigarrow$ Ineffizienz
 - Beispiel: $\{A \wedge B, \neg C \wedge \neg D, \neg A \vee \neg B \vee C \vee D\}$

- **Vorrang** beachten: Formeln von „außen“ nach „innen“ abarbeiten
- **Clash**: nur zwischen aktueller Spalte und den darüberliegenden
 - Spalte steht logisch für Menge aller Formeln, die in ihr und den darüberliegenden Spalten enthalten sind
- Effizienz: Wenn mehrere Regeln anwendbar sind: **$\wedge$ -Regel zuerst**
 - sonst: $\wedge$ -Regel in jeder neuen Spalte anwendbar $\rightsquigarrow$ Ineffizienz
 - Beispiel: $\{A \wedge B, \neg C \wedge \neg D, \neg A \vee \neg B \vee C \vee D\}$
- Effizienz: $\vee$ -Regel so anwenden, dass möglichst wenig offene Spalten verbleiben

- **Vorrang** beachten: Formeln von „außen“ nach „innen“ abarbeiten
- **Clash**: nur zwischen aktueller Spalte und den darüberliegenden
 - Spalte steht logisch für Menge aller Formeln, die in ihr und den darüberliegenden Spalten enthalten sind
- Effizienz: Wenn mehrere Regeln anwendbar sind: **$\wedge$ -Regel zuerst**
 - sonst: $\wedge$ -Regel in jeder neuen Spalte anwendbar $\rightsquigarrow$ Ineffizienz
 - Beispiel: $\{A \wedge B, \neg C \wedge \neg D, \neg A \vee \neg B \vee C \vee D\}$
- Effizienz: $\vee$ -Regel so anwenden, dass möglichst wenig offene Spalten verbleiben
 - Beispiel: $\{\neg A, A \vee B, C \vee D\} \rightsquigarrow$ zuerst $A \vee B$

- **Vorrang** beachten: Formeln von „außen“ nach „innen“ abarbeiten
- **Clash**: nur zwischen aktueller Spalte und den darüberliegenden
 - Spalte steht logisch für Menge aller Formeln, die in ihr und den darüberliegenden Spalten enthalten sind
- Effizienz: Wenn mehrere Regeln anwendbar sind: **$\wedge$ -Regel zuerst**
 - sonst: $\wedge$ -Regel in jeder neuen Spalte anwendbar $\rightsquigarrow$ Ineffizienz
 - Beispiel: $\{A \wedge B, \neg C \wedge \neg D, \neg A \vee \neg B \vee C \vee D\}$
- Effizienz: **$\vee$ -Regel** so anwenden, dass möglichst wenig offene Spalten verbleiben
 - Beispiel: $\{\neg A, A \vee B, C \vee D\} \rightsquigarrow$ zuerst $A \vee B$
- Transformation von NNF in **KNF** ist **kontraproduktiv**

- **Vorrang** beachten: Formeln von „außen“ nach „innen“ abarbeiten
- **Clash**: nur zwischen aktueller Spalte und den darüberliegenden
 - Spalte steht logisch für Menge aller Formeln, die in ihr und den darüberliegenden Spalten enthalten sind
- Effizienz: Wenn mehrere Regeln anwendbar sind: **$\wedge$ -Regel zuerst**
 - sonst: $\wedge$ -Regel in jeder neuen Spalte anwendbar $\rightsquigarrow$ Ineffizienz
 - Beispiel: $\{A \wedge B, \neg C \wedge \neg D, \neg A \vee \neg B \vee C \vee D\}$
- Effizienz: **$\vee$ -Regel** so anwenden, dass möglichst wenig offene Spalten verbleiben
 - Beispiel: $\{\neg A, A \vee B, C \vee D\} \rightsquigarrow$ zuerst $A \vee B$
- Transformation von NNF in **KNF** ist **kontraproduktiv**
 - nach erstem Schritt wird nur $\vee$ -Regel angewendet $\rightsquigarrow$ Ineffizienz

- **Vorrang** beachten: Formeln von „außen“ nach „innen“ abarbeiten
- **Clash**: nur zwischen aktueller Spalte und den darüberliegenden
 - Spalte steht logisch für Menge aller Formeln, die in ihr und den darüberliegenden Spalten enthalten sind
- Effizienz: Wenn mehrere Regeln anwendbar sind: **$\wedge$ -Regel zuerst**
 - sonst: $\wedge$ -Regel in jeder neuen Spalte anwendbar $\rightsquigarrow$ Ineffizienz
 - Beispiel: $\{A \wedge B, \neg C \wedge \neg D, \neg A \vee \neg B \vee C \vee D\}$
- Effizienz: **$\vee$ -Regel** so anwenden, dass möglichst wenig offene Spalten verbleiben
 - Beispiel: $\{\neg A, A \vee B, C \vee D\} \rightsquigarrow$ zuerst $A \vee B$
- Transformation von NNF in **KNF** ist **kontraproduktiv**
 - nach erstem Schritt wird nur $\vee$ -Regel angewendet $\rightsquigarrow$ Ineffizienz
 - Beispiel: vergleiche NNF $\neg A \vee B \wedge \neg C \vee \neg B \wedge C$ (3 Spalten)
mit KNF $(\neg A \vee B \vee C) \wedge (\neg A \vee \neg B \vee \neg C)$ (7 Spalten)

Übung 3.50

Zeigen Sie mit einem Tableau, dass Bert und Claus schuldig sind.

$$\begin{aligned} & (A \vee B \vee C) \wedge (\neg A \vee (B \wedge \neg C) \vee (\neg B \wedge C)) \wedge \\ & (B \vee \neg C) \wedge (\neg A \vee \neg B \vee C) \wedge (C \vee A) \wedge (\neg B \vee \neg C) \end{aligned}$$

Satz 3.51 (Korrektheit)

Wenn das vollständige Tableau für φ eine Clash-freie Spalte enthält, ist φ erfüllbar.

Satz 3.51 (Korrektheit)

Wenn das vollständige Tableau für φ eine Clash-freie Spalte enthält, ist φ erfüllbar.

Beweis.

Idee: Literale der Clash-freien Spalte S bilden ein Modell $\mathcal{M}$.

- Für alle Variablen X : Wenn $X \in S$, dann $X^{\mathcal{M}} = 1$; sonst $X^{\mathcal{M}} = 0$

Satz 3.51 (Korrektheit)

Wenn das vollständige Tableau für φ eine Clash-freie Spalte enthält, ist φ erfüllbar.

Beweis.

Idee: Literale der Clash-freien Spalte S bilden ein Modell $\mathcal{M}$.

- Für alle Variablen X : Wenn $X \in S$, dann $X^{\mathcal{M}} = 1$; sonst $X^{\mathcal{M}} = 0$
 $\rightsquigarrow$ Jede allein vorkommende Variable wird wahr.

Satz 3.51 (Korrektheit)

Wenn das vollständige Tableau für φ eine Clash-freie Spalte enthält, ist φ erfüllbar.

Beweis.

Idee: Literale der Clash-freien Spalte S bilden ein Modell $\mathcal{M}$.

- Für alle Variablen X : Wenn $X \in S$, dann $X^{\mathcal{M}} = 1$; sonst $X^{\mathcal{M}} = 0$
 $\rightsquigarrow$ Jede allein vorkommende Variable wird wahr.
- Clash-frei: für keine Variable X ist X und $\neg X$ enthalten

Satz 3.51 (Korrektheit)

Wenn das vollständige Tableau für φ eine Clash-freie Spalte enthält, ist φ erfüllbar.

Beweis.

Idee: Literale der Clash-freien Spalte S bilden ein Modell $\mathcal{M}$.

- Für alle Variablen X : Wenn $X \in S$, dann $X^{\mathcal{M}} = 1$; sonst $X^{\mathcal{M}} = 0$
 - ↪ Jede allein vorkommende Variable wird wahr.
- Clash-frei: für keine Variable X ist X und $\neg X$ enthalten
 - ↪ Jedes allein vorkommende **negierte** Literal wird wahr.

Satz 3.51 (Korrektheit)

Wenn das vollständige Tableau für φ eine Clash-freie Spalte enthält, ist φ erfüllbar.

Beweis.

Idee: Literale der Clash-freien Spalte S bilden ein Modell $\mathcal{M}$.

- Für alle Variablen X : Wenn $X \in S$, dann $X^{\mathcal{M}} = 1$; sonst $X^{\mathcal{M}} = 0$
 - ↪ Jede allein vorkommende Variable wird wahr.
- Clash-frei: für keine Variable X ist X und $\neg X$ enthalten
 - ↪ Jedes allein vorkommende **negierte** Literal wird wahr.
- Vollständig:

Satz 3.51 (Korrektheit)

Wenn das vollständige Tableau für φ eine Clash-freie Spalte enthält, ist φ erfüllbar.

Beweis.

Idee: Literale der Clash-freien Spalte S bilden ein Modell $\mathcal{M}$.

- Für alle Variablen X : Wenn $X \in S$, dann $X^{\mathcal{M}} = 1$; sonst $X^{\mathcal{M}} = 0$
 - ↪ Jede allein vorkommende Variable wird wahr.
- Clash-frei: für keine Variable X ist X und $\neg X$ enthalten
 - ↪ Jedes allein vorkommende **negierte** Literal wird wahr.
- Vollständig:
 - für jede Konjunktion $\chi \wedge \psi \in S$ gilt auch $\{\chi, \psi\} \subseteq S$

Satz 3.51 (Korrektheit)

Wenn das vollständige Tableau für φ eine Clash-freie Spalte enthält, ist φ erfüllbar.

Beweis.

Idee: Literale der Clash-freien Spalte S bilden ein Modell $\mathcal{M}$.

- Für alle Variablen X : Wenn $X \in S$, dann $X^{\mathcal{M}} = 1$; sonst $X^{\mathcal{M}} = 0$
 - $\rightsquigarrow$ Jede allein vorkommende Variable wird wahr.
- Clash-frei: für keine Variable X ist X und $\neg X$ enthalten
 - $\rightsquigarrow$ Jedes allein vorkommende **negierte** Literal wird wahr.
- Vollständig:
 - für jede Konjunktion $\chi \wedge \psi \in S$ gilt auch $\{\chi, \psi\} \subseteq S$
 - für jede Disjunktion $\chi \vee \psi \in S$ gilt auch $\{\chi, \psi\} \cap S \neq \emptyset$

Satz 3.51 (Korrektheit)

Wenn das vollständige Tableau für φ eine Clash-freie Spalte enthält, ist φ erfüllbar.

Beweis.

Idee: Literale der Clash-freien Spalte S bilden ein Modell $\mathcal{M}$.

- Für alle Variablen X : Wenn $X \in S$, dann $X^{\mathcal{M}} = 1$; sonst $X^{\mathcal{M}} = 0$
 - $\rightsquigarrow$ Jede allein vorkommende Variable wird wahr.
- Clash-frei: für keine Variable X ist X und $\neg X$ enthalten
 - $\rightsquigarrow$ Jedes allein vorkommende **negierte** Literal wird wahr.
- Vollständig:
 - für jede Konjunktion $\chi \wedge \psi \in S$ gilt auch $\{\chi, \psi\} \subseteq S$
 - für jede Disjunktion $\chi \vee \psi \in S$ gilt auch $\{\chi, \psi\} \cap S \neq \emptyset$
- Induktion: Für jedes $\psi \in S$ gilt: $\mathcal{M} \models \psi$



Satz 3.52 (Vollständigkeit)

Wenn φ erfüllbar ist, enthält das vollständige Tableau für φ eine Clash-freie Spalte.

Satz 3.52 (Vollständigkeit)

Wenn φ erfüllbar ist, enthält das vollständige Tableau für φ eine Clash-freie Spalte.

Beweis.

Idee: „Regelanwendung erhält Erfüllbarkeit.“

Satz 3.52 (Vollständigkeit)

Wenn φ erfüllbar ist, enthält das vollständige Tableau für φ eine Clash-freie Spalte.

Beweis.

Idee: „Regelanwendung erhält Erfüllbarkeit.“

Wenn eine Formelmenge S vor Regelanwendung erfüllbar war, kann die Regel so angewendet werden, dass S auch nach der Anwendung noch erfüllbar ist.

Satz 3.52 (Vollständigkeit)

Wenn φ erfüllbar ist, enthält das vollständige Tableau für φ eine Clash-freie Spalte.

Beweis.

Idee: „Regelanwendung erhält Erfüllbarkeit.“

Wenn eine Formelmenge S vor Regelanwendung erfüllbar war, kann die Regel so angewendet werden, dass S auch nach der Anwendung noch erfüllbar ist.

$\wedge$ -Regel Wenn $\mathcal{M} \models \psi \wedge \chi$ gilt, gilt auch $\mathcal{M} \models \psi$ und $\mathcal{M} \models \chi$.

Satz 3.52 (Vollständigkeit)

Wenn φ erfüllbar ist, enthält das vollständige Tableau für φ eine Clash-freie Spalte.

Beweis.

Idee: „Regelanwendung erhält Erfüllbarkeit.“

Wenn eine Formelmengende S vor Regelanwendung erfüllbar war, kann die Regel so angewendet werden, dass S auch nach der Anwendung noch erfüllbar ist.

$\wedge$ -Regel Wenn $\mathcal{M} \models \psi \wedge \chi$ gilt, gilt auch $\mathcal{M} \models \psi$ und $\mathcal{M} \models \chi$.

$\vee$ -Regel Wenn $\mathcal{M} \models \psi \vee \chi$ gilt, gilt auch $\mathcal{M} \models \psi$ oder $\mathcal{M} \models \chi$.

Satz 3.52 (Vollständigkeit)

Wenn φ erfüllbar ist, enthält das vollständige Tableau für φ eine Clash-freie Spalte.

Beweis.

Idee: „Regelanwendung erhält Erfüllbarkeit.“

Wenn eine Formelmengens S vor Regelanwendung erfüllbar war, kann die Regel so angewendet werden, dass S auch nach der Anwendung noch erfüllbar ist.

$\wedge$ -Regel Wenn $\mathcal{M} \models \psi \wedge \chi$ gilt, gilt auch $\mathcal{M} \models \psi$ und $\mathcal{M} \models \chi$.

$\vee$ -Regel Wenn $\mathcal{M} \models \psi \vee \chi$ gilt, gilt auch $\mathcal{M} \models \psi$ oder $\mathcal{M} \models \chi$.

Satz 3.52 (Vollständigkeit)

Wenn φ erfüllbar ist, enthält das vollständige Tableau für φ eine Clash-freie Spalte.

Beweis.

Idee: „Regelanwendung erhält Erfüllbarkeit.“

Wenn eine Formelmengens S vor Regelanwendung erfüllbar war, kann die Regel so angewendet werden, dass S auch nach der Anwendung noch erfüllbar ist.

$\wedge$ -Regel Wenn $\mathcal{M} \models \psi \wedge \chi$ gilt, gilt auch $\mathcal{M} \models \psi$ und $\mathcal{M} \models \chi$.

$\vee$ -Regel Wenn $\mathcal{M} \models \psi \vee \chi$ gilt, gilt auch $\mathcal{M} \models \psi$ oder $\mathcal{M} \models \chi$.

Erfüllbare Mengen enthalten keinen Clash.



Definition 3.53 (Teilformel)

Sei φ eine aussagenlogische Formel.

Die Formel ψ ist **Teilformel** von φ , wenn eine der folgenden Bedingungen erfüllt ist:

- $\varphi = \psi$

Definition 3.53 (Teilformel)

Sei φ eine aussagenlogische Formel.

Die Formel ψ ist **Teilformel** von φ , wenn eine der folgenden Bedingungen erfüllt ist:

- $\varphi = \psi$
- $\varphi = \neg\chi$ und ψ ist Teilformel von χ

Definition 3.53 (Teilformel)

Sei φ eine aussagenlogische Formel.

Die Formel ψ ist **Teilformel** von φ , wenn eine der folgenden Bedingungen erfüllt ist:

- $\varphi = \psi$
- $\varphi = \neg\chi$ und ψ ist Teilformel von χ
- $\varphi = \chi_1 \circ \chi_2$ und ψ ist Teilformel von χ_1 oder χ_2
(mit $\circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$)

Definition 3.53 (Teilformel)

Sei φ eine aussagenlogische Formel.

Die Formel ψ ist **Teilformel** von φ , wenn eine der folgenden Bedingungen erfüllt ist:

- $\varphi = \psi$
- $\varphi = \neg\chi$ und ψ ist Teilformel von χ
- $\varphi = \chi_1 \circ \chi_2$ und ψ ist Teilformel von χ_1 oder χ_2
(mit $\circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$)

Definition 3.53 (Teilformel)

Sei φ eine aussagenlogische Formel.

Die Formel ψ ist **Teilformel** von φ , wenn eine der folgenden Bedingungen erfüllt ist:

- $\varphi = \psi$
- $\varphi = \neg \chi$ und ψ ist Teilformel von χ
- $\varphi = \chi_1 \circ \chi_2$ und ψ ist Teilformel von χ_1 oder χ_2
(mit $\circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$)

Beispiel 3.54 ($\varphi = A \wedge \neg(B \vee \neg A) \vee (A \rightarrow \neg C)$)

Die Menge der Teilformeln von φ ist

$\{\varphi, A \wedge \neg(B \vee \neg A), A \rightarrow \neg C, A, \neg(B \vee \neg A), B \vee \neg A, B, \neg A, \neg C, C\}$.

Satz 3.55 (Terminierung)

Der Tableau-Algorithmus terminiert für jede Formel φ nach endlich vielen Schritten.

Satz 3.55 (Terminierung)

Der Tableau-Algorithmus terminiert für jede Formel φ nach endlich vielen Schritten.

Beweis.

- Es gibt nur endlich viele Teilformeln von $\text{nnf}(\varphi)$, also ein endliches Vokabular.

Satz 3.55 (Terminierung)

Der Tableau-Algorithmus terminiert für jede Formel φ nach endlich vielen Schritten.

Beweis.

- Es gibt nur endlich viele Teilformeln von $\text{nnf}(\varphi)$, also ein endliches Vokabular.
- Jede Regelanwendung fügt nur Teilformeln zur Formelmenge hinzu.

Satz 3.55 (Terminierung)

Der Tableau-Algorithmus terminiert für jede Formel φ nach endlich vielen Schritten.

Beweis.

- Es gibt nur endlich viele Teilformeln von $\text{nnf}(\varphi)$, also ein endliches Vokabular.
- Jede Regelanwendung fügt nur Teilformeln zur Formelmenge hinzu.
- Auf jede Teilformel wird in jeder Spalte höchstens einmal eine Regel angewendet.

Satz 3.55 (Terminierung)

Der Tableau-Algorithmus terminiert für jede Formel φ nach endlich vielen Schritten.

Beweis.

- Es gibt nur endlich viele Teilformeln von $\text{nnf}(\varphi)$, also ein endliches Vokabular.
- Jede Regelanwendung fügt nur Teilformeln zur Formelmenge hinzu.
- Auf jede Teilformel wird in jeder Spalte höchstens einmal eine Regel angewendet.
- Die Anzahl der Spalten ist beschränkt durch die Anzahl der Disjunktionen in $\text{nnf}(\varphi)$.

Satz 3.55 (Terminierung)

Der Tableau-Algorithmus terminiert für jede Formel φ nach endlich vielen Schritten.

Beweis.

- Es gibt nur endlich viele Teilformeln von $\text{nnf}(\varphi)$, also ein endliches Vokabular.
- Jede Regelanwendung fügt nur Teilformeln zur Formelmenge hinzu.
- Auf jede Teilformel wird in jeder Spalte höchstens einmal eine Regel angewendet.
- Die Anzahl der Spalten ist beschränkt durch die Anzahl der Disjunktionen in $\text{nnf}(\varphi)$.
- Formeln werden nie entfernt.



- sind mehrere Regeln anwendbar, ist die Auswahl der nächsten Regel *don't-care-nichtdeterministisch*

- sind mehrere Regeln anwendbar, ist die Auswahl der nächsten Regel *don't-care-nichtdeterministisch*
 - Jede Auswahl führt zur Lösung

- sind mehrere Regeln anwendbar, ist die Auswahl der nächsten Regel **don't-care-nichtdeterministisch**
 - Jede Auswahl führt zur Lösung
- die Auswahl der Alternative durch die V-Regel ist **don't-know-nichtdeterministisch**

- sind mehrere Regeln anwendbar, ist die Auswahl der nächsten Regel **don't-care-nichtdeterministisch**
 - Jede Auswahl führt zur Lösung
- die Auswahl der Alternative durch die V-Regel ist **don't-know-nichtdeterministisch**
 - Finden der Lösung kann von „richtiger“ Alternative abhängen

- sind mehrere Regeln anwendbar, ist die Auswahl der nächsten Regel **don't-care-nichtdeterministisch**
 - Jede Auswahl führt zur Lösung
- die Auswahl der Alternative durch die V-Regel ist **don't-know-nichtdeterministisch**
 - Finden der Lösung kann von „richtiger“ Alternative abhängen
 - Jede Alternative muss getestet werden

- Anzahl der Spalten kann exponentiell in der Größe von φ sein

- Anzahl der Spalten kann exponentiell in der Größe von φ sein
 - Beispiel: $\varphi = (A \vee B) \wedge (C \vee D) \wedge (E \vee F)$

- Anzahl der Spalten kann exponentiell in der Größe von φ sein
 - Beispiel: $\varphi = (A \vee B) \wedge (C \vee D) \wedge (E \vee F)$
- Für Zeit- und Platz-Effizienz: Erzeuge Tableau **depth-first**

- Anzahl der Spalten kann exponentiell in der Größe von φ sein
 - Beispiel: $\varphi = (A \vee B) \wedge (C \vee D) \wedge (E \vee F)$
- Für Zeit- und Platz-Effizienz: Erzeuge Tableau **depth-first**
 - Halte nur eine Spalte gleichzeitig im Speicher

- Anzahl der Spalten kann exponentiell in der Größe von φ sein
 - Beispiel: $\varphi = (A \vee B) \wedge (C \vee D) \wedge (E \vee F)$
- Für Zeit- und Platz-Effizienz: Erzeuge Tableau **depth-first**
 - Halte nur eine Spalte gleichzeitig im Speicher
 - Wende dort alle möglichen Regeln an

- Anzahl der Spalten kann exponentiell in der Größe von φ sein
 - Beispiel: $\varphi = (A \vee B) \wedge (C \vee D) \wedge (E \vee F)$
- Für Zeit- und Platz-Effizienz: Erzeuge Tableau **depth-first**
 - Halte nur eine Spalte gleichzeitig im Speicher
 - Wende dort alle möglichen Regeln an
 - Spalte ist Clash-frei und vollständig: Ausgabe „erfüllbar“

- Anzahl der Spalten kann exponentiell in der Größe von φ sein
 - Beispiel: $\varphi = (A \vee B) \wedge (C \vee D) \wedge (E \vee F)$
- Für Zeit- und Platz-Effizienz: Erzeuge Tableau **depth-first**
 - Halte nur eine Spalte gleichzeitig im Speicher
 - Wende dort alle möglichen Regeln an
 - Spalte ist Clash-frei und vollständig: Ausgabe „erfüllbar“
 - Spalte enthält Clash: Versuche nächste Spalte (oder Ausgabe „unerfüllbar“)

- Anzahl der Spalten kann exponentiell in der Größe von φ sein
 - Beispiel: $\varphi = (A \vee B) \wedge (C \vee D) \wedge (E \vee F)$
- Für Zeit- und Platz-Effizienz: Erzeuge Tableau **depth-first**
 - Halte nur eine Spalte gleichzeitig im Speicher
 - Wende dort alle möglichen Regeln an
 - Spalte ist Clash-frei und vollständig: Ausgabe „erfüllbar“
 - Spalte enthält Clash: Versuche nächste Spalte (oder Ausgabe „unerfüllbar“)
- Overhead: Backtracking-Information for jede Anwendung der $\vee$ -Regel

- Anzahl der Spalten kann exponentiell in der Größe von φ sein
 - Beispiel: $\varphi = (A \vee B) \wedge (C \vee D) \wedge (E \vee F)$
- Für Zeit- und Platz-Effizienz: Erzeuge Tableau **depth-first**
 - Halte nur eine Spalte gleichzeitig im Speicher
 - Wende dort alle möglichen Regeln an
 - Spalte ist Clash-frei und vollständig: Ausgabe „erfüllbar“
 - Spalte enthält Clash: Versuche nächste Spalte (oder Ausgabe „unerfüllbar“)
- Overhead: Backtracking-Information for jede Anwendung der $\vee$ -Regel
 - aus Effizienzgründen auch hier zuerst $\wedge$ -Regel anwenden

Gemeinsamkeiten

- Erfüllbarkeitstests

Gemeinsamkeiten

- Erfüllbarkeitstests
- Können auch Gültigkeit und Implikation testen

Gemeinsamkeiten

- Erfüllbarkeitstests
- Können auch Gültigkeit und Implikation testen
- Entscheidungsverfahren

Gemeinsamkeiten

- Erfüllbarkeitstests
- Können auch Gültigkeit und Implikation testen
- Entscheidungsverfahren
- Effizienter als Wahrheitstabelle

Gemeinsamkeiten

- Erfüllbarkeitstests
- Können auch Gültigkeit und Implikation testen
- Entscheidungsverfahren
- Effizienter als Wahrheitstabelle

Gemeinsamkeiten

- Erfüllbarkeitstests
- Können auch Gültigkeit und Implikation testen
- Entscheidungsverfahren
- Effizienter als Wahrheitstabelle

Vorteile von Tableaus

- Effizient für erfüllbare Eingaben

Gemeinsamkeiten

- Erfüllbarkeitstests
- Können auch Gültigkeit und Implikation testen
- Entscheidungsverfahren
- Effizienter als Wahrheitstabelle

Vorteile von Tableaus

- Effizient für erfüllbare Eingaben
- Erzeugt Modell für erfüllbare Eingaben

Gemeinsamkeiten

- Erfüllbarkeitstests
- Können auch Gültigkeit und Implikation testen
- Entscheidungsverfahren
- Effizienter als Wahrheitstabelle

Vorteile von Tableaus

- Effizient für erfüllbare Eingaben
- Erzeugt Modell für erfüllbare Eingaben

Gemeinsamkeiten

- Erfüllbarkeitstests
- Können auch Gültigkeit und Implikation testen
- Entscheidungsverfahren
- Effizienter als Wahrheitstabelle

Vorteile von Tableaus

- Effizient für erfüllbare Eingaben
- Erzeugt Modell für erfüllbare Eingaben

Vorteile der Resolution

- Effizient für unerfüllbare Eingaben

Gemeinsamkeiten

- Erfüllbarkeitstests
- Können auch Gültigkeit und Implikation testen
- Entscheidungsverfahren
- Effizienter als Wahrheitstabelle

Vorteile von Tableaus

- Effizient für erfüllbare Eingaben
- Erzeugt Modell für erfüllbare Eingaben

Vorteile der Resolution

- Effizient für unerfüllbare Eingaben
- Gesamtes Verfahren ist don't-care-nichtdeterministisch (kein Backtracking)

Aussagenvariable repräsentiert Elementaraussage
kann wahr oder falsch sein

Aussagenvariable repräsentiert Elementaraussage
kann wahr oder falsch sein

Junktoren verbinden Aussagenvariablen zu komplexen Formeln
Auswertung über Wahrheitstabelle

Aussagenvariable repräsentiert Elementaraussage
kann wahr oder falsch sein

Junktoren verbinden Aussagenvariablen zu komplexen Formeln
Auswertung über Wahrheitstabelle

Interpretation belegt Variablen mit 0 oder 1
macht Formel wahr oder falsch

Aussagenvariable repräsentiert Elementaraussage
kann wahr oder falsch sein

Junktoren verbinden Aussagenvariablen zu komplexen Formeln
Auswertung über Wahrheitstabelle

Interpretation belegt Variablen mit 0 oder 1
macht Formel wahr oder falsch

Modell macht Formel wahr

Aussagenvariable repräsentiert Elementaraussage
kann wahr oder falsch sein

Junktoren verbinden Aussagenvariablen zu komplexen Formeln
Auswertung über Wahrheitstabelle

Interpretation belegt Variablen mit 0 oder 1
macht Formel wahr oder falsch

Modell macht Formel wahr

Erfüllbar es gibt ein Modell

Aussagenvariable repräsentiert Elementaraussage
kann wahr oder falsch sein

Junktoren verbinden Aussagenvariablen zu komplexen Formeln
Auswertung über Wahrheitstabelle

Interpretation belegt Variablen mit 0 oder 1
macht Formel wahr oder falsch

Modell macht Formel wahr

Erfüllbar es gibt ein Modell

Gültig jede Interpretation ist Modell (Tautologie)

Aussagenvariable repräsentiert Elementaraussage
kann wahr oder falsch sein

Junktoren verbinden Aussagenvariablen zu komplexen Formeln
Auswertung über Wahrheitstabelle

Interpretation belegt Variablen mit 0 oder 1
macht Formel wahr oder falsch

Modell macht Formel wahr

Erfüllbar es gibt ein Modell

Gültig jede Interpretation ist Modell (Tautologie)

Schlussfolgerung Erfüllbarkeit, Gültigkeit, Implikation
entscheidbar (korrekt, vollständig, terminierend)

- Aussagenvariable** repräsentiert Elementaraussage
kann wahr oder falsch sein
- Junktoren** verbinden Aussagenvariablen zu komplexen Formeln
Auswertung über Wahrheitstabelle
- Interpretation** belegt Variablen mit 0 oder 1
macht Formel wahr oder falsch
- Modell** macht Formel wahr
- Erfüllbar** es gibt ein Modell
- Gültig** jede Interpretation ist Modell (Tautologie)
- Schlussfolgerung** Erfüllbarkeit, Gültigkeit, Implikation
entscheidbar (korrekt, vollständig, terminierend)
 - Resolution** sucht Widerspruch
KNF, Klausel, Resolvente

- Aussagenvariable** repräsentiert Elementaraussage
kann wahr oder falsch sein
- Junktoren** verbinden Aussagenvariablen zu komplexen Formeln
Auswertung über Wahrheitstabelle
- Interpretation** belegt Variablen mit 0 oder 1
macht Formel wahr oder falsch
- Modell** macht Formel wahr
- Erfüllbar** es gibt ein Modell
- Gültig** jede Interpretation ist Modell (Tautologie)
- Schlussfolgerung** Erfüllbarkeit, Gültigkeit, Implikation
entscheidbar (korrekt, vollständig, terminierend)
 - Resolution** sucht Widerspruch
KNF, Klausel, Resolvente
 - Tableau** sucht Modell
NNF, Regel, Clash

1. Einführung

2. Mengen

3. Aussagenlogik

4. Prädikatenlogik

4.1 Relationen und Funktionen

4.2 Syntax

4.3 Semantik

4.4 Schlussfolgerungsverfahren

5. Prolog

- Aussagenlogik kann Aussagen nur als Ganzes beschreiben

- Aussagenlogik kann Aussagen nur als Ganzes beschreiben
- Aussagenvariablen können nur wahr oder falsch sein

- Aussagenlogik kann Aussagen nur als Ganzes beschreiben
- Aussagenvariablen können nur wahr oder falsch sein
- Innere Struktur der Aussagen kann nicht repräsentiert werden

- Aussagenlogik kann Aussagen nur als Ganzes beschreiben
- Aussagenvariablen können nur wahr oder falsch sein
- Innere Struktur der Aussagen kann nicht repräsentiert werden

- Aussagenlogik kann Aussagen nur als Ganzes beschreiben
- Aussagenvariablen können nur wahr oder falsch sein
- Innere Struktur der Aussagen kann nicht repräsentiert werden

Beispiel 4.1

- Innere Struktur von Aussagen
„Anna kennt Bert und Anna kennt die Mutter von Bert.“ $\rightsquigarrow X \wedge Y$

- Aussagenlogik kann Aussagen nur als Ganzes beschreiben
- Aussagenvariablen können nur wahr oder falsch sein
- Innere Struktur der Aussagen kann nicht repräsentiert werden

Beispiel 4.1

- Innere Struktur von Aussagen
„Anna kennt Bert und Anna kennt die Mutter von Bert.“ $\rightsquigarrow X \wedge Y$

- Aussagenlogik kann Aussagen nur als Ganzes beschreiben
- Aussagenvariablen können nur wahr oder falsch sein
- Innere Struktur der Aussagen kann nicht repräsentiert werden

Beispiel 4.1

- Innere Struktur von Aussagen

„Anna kennt Bert und Anna kennt die Mutter von Bert.“ $\rightsquigarrow X \wedge Y$

„Anna kennt Bert und Anna mag Bert.“ $\rightsquigarrow X \wedge Y$

- Aussagenlogik kann Aussagen nur als Ganzes beschreiben
- Aussagenvariablen können nur wahr oder falsch sein
- Innere Struktur der Aussagen kann nicht repräsentiert werden

Beispiel 4.1

- Innere Struktur von Aussagen

„Anna kennt Bert und Anna kennt die Mutter von Bert.“ $\rightsquigarrow X \wedge Y$

„Anna kennt Bert und Anna mag Bert.“ $\rightsquigarrow X \wedge Y$

„Anna kennt Bert und Claus kennt Bert.“ $\rightsquigarrow X \wedge Y$

- Aussagenlogik kann Aussagen nur als Ganzes beschreiben
- Aussagenvariablen können nur wahr oder falsch sein
- Innere Struktur der Aussagen kann nicht repräsentiert werden

Beispiel 4.1

■ Innere Struktur von Aussagen

„Anna kennt Bert und Anna kennt die Mutter von Bert.“ $\rightsquigarrow X \wedge Y$

„Anna kennt Bert und Anna mag Bert.“ $\rightsquigarrow X \wedge Y$

„Anna kennt Bert und Claus kennt Bert.“ $\rightsquigarrow X \wedge Y$

■ Schlussfolgerung

Prämissen: „Tux ist ein Pinguin.“ $\rightsquigarrow A$

- Aussagenlogik kann Aussagen nur als **Ganzes** beschreiben
- Aussagenvariablen können nur wahr oder falsch sein
- Innere Struktur der Aussagen kann nicht repräsentiert werden

Beispiel 4.1

- Innere Struktur von Aussagen

„Anna kennt Bert und Anna kennt **die Mutter von Bert**.“ $\rightsquigarrow X \wedge Y$

„Anna kennt Bert und Anna **mag** Bert.“ $\rightsquigarrow X \wedge Y$

„Anna kennt Bert und **Claus** kennt Bert.“ $\rightsquigarrow X \wedge Y$

- Schlussfolgerung

Prämissen: „Tux ist ein Pinguin.“ $\rightsquigarrow A$

- Aussagenlogik kann Aussagen nur als **Ganzes** beschreiben
- Aussagenvariablen können nur wahr oder falsch sein
- Innere Struktur der Aussagen kann nicht repräsentiert werden

Beispiel 4.1

■ Innere Struktur von Aussagen

„Anna kennt Bert und Anna kennt **die Mutter von Bert**.“ $\rightsquigarrow X \wedge Y$

„Anna kennt Bert und Anna **mag** Bert.“ $\rightsquigarrow X \wedge Y$

„Anna kennt Bert und **Claus** kennt Bert.“ $\rightsquigarrow X \wedge Y$

■ Schlussfolgerung

Prämissen: „Tux ist ein Pinguin.“ $\rightsquigarrow A$

„Kein Pinguin kann fliegen.“ $\rightsquigarrow \neg B$

- Aussagenlogik kann Aussagen nur als Ganzes beschreiben
- Aussagenvariablen können nur wahr oder falsch sein
- Innere Struktur der Aussagen kann nicht repräsentiert werden

Beispiel 4.1

■ Innere Struktur von Aussagen

„Anna kennt Bert und Anna kennt die Mutter von Bert.“ $\rightsquigarrow X \wedge Y$

„Anna kennt Bert und Anna mag Bert.“ $\rightsquigarrow X \wedge Y$

„Anna kennt Bert und Claus kennt Bert.“ $\rightsquigarrow X \wedge Y$

■ Schlussfolgerung

Prämissen: „Tux ist ein Pinguin.“ $\rightsquigarrow A$

„Kein Pinguin kann fliegen.“ $\rightsquigarrow \neg B$

„Alle Pinguine sind Vögel.“ $\rightsquigarrow C$

- Aussagenlogik kann Aussagen nur als **Ganzes** beschreiben
- Aussagenvariablen können nur wahr oder falsch sein
- Innere Struktur der Aussagen kann nicht repräsentiert werden

Beispiel 4.1

■ Innere Struktur von Aussagen

„Anna kennt Bert und Anna kennt **die Mutter von Bert**.“ $\rightsquigarrow X \wedge Y$

„Anna kennt Bert und Anna **mag** Bert.“ $\rightsquigarrow X \wedge Y$

„Anna kennt Bert und **Claus** kennt Bert.“ $\rightsquigarrow X \wedge Y$

■ Schlussfolgerung

Prämissen: „Tux ist ein Pinguin.“ $\rightsquigarrow A$

„Kein Pinguin kann fliegen.“ $\rightsquigarrow \neg B$

„Alle Pinguine sind Vögel.“ $\rightsquigarrow C$

Konklusion: „Manche Vögel können nicht fliegen.“ $\rightsquigarrow D$

- Aussagenlogik kann Aussagen nur als **Ganzes** beschreiben
- Aussagenvariablen können nur wahr oder falsch sein
- Innere Struktur der Aussagen kann nicht repräsentiert werden

Beispiel 4.1

■ Innere Struktur von Aussagen

„Anna kennt Bert und Anna kennt **die Mutter von Bert**.“ $\rightsquigarrow X \wedge Y$

„Anna kennt Bert und Anna **mag** Bert.“ $\rightsquigarrow X \wedge Y$

„Anna kennt Bert und **Claus** kennt Bert.“ $\rightsquigarrow X \wedge Y$

■ Schlussfolgerung

Prämissen: „Tux ist ein Pinguin.“ $\rightsquigarrow A$

„Kein Pinguin kann fliegen.“ $\rightsquigarrow \neg B$

„Alle Pinguine sind Vögel.“ $\rightsquigarrow C$

Konklusion: „Manche Vögel können nicht fliegen.“ $\rightsquigarrow D$

- Formalisierung zu grob: kann inneren Aufbau der Elementaraussagen nicht abbilden

- Aussagenlogik kann Aussagen nur als **Ganzes** beschreiben
- Aussagenvariablen können nur wahr oder falsch sein
- Innere Struktur der Aussagen kann nicht repräsentiert werden

Beispiel 4.1

■ Innere Struktur von Aussagen

„Anna kennt Bert und Anna kennt **die Mutter von Bert.**“ $\rightsquigarrow X \wedge Y$

„Anna kennt Bert und Anna **mag** Bert.“ $\rightsquigarrow X \wedge Y$

„Anna kennt Bert und **Claus** kennt Bert.“ $\rightsquigarrow X \wedge Y$

■ Schlussfolgerung

Prämissen: „Tux ist ein Pinguin.“ $\rightsquigarrow A$

„Kein Pinguin kann fliegen.“ $\rightsquigarrow \neg B$

„Alle Pinguine sind Vögel.“ $\rightsquigarrow C$

Konklusion: „Manche Vögel können nicht fliegen.“ $\rightsquigarrow D$

- Formalisierung zu grob: kann inneren Aufbau der Elementaraussagen nicht abbilden
- Zusammenhänge werden nicht klar („kann fliegen“, „Pinguin“, „Vogel“)

- Aussagenlogik kann Aussagen nur als **Ganzes** beschreiben
- Aussagenvariablen können nur wahr oder falsch sein
- Innere Struktur der Aussagen kann nicht repräsentiert werden

Beispiel 4.1

■ Innere Struktur von Aussagen

„Anna kennt Bert und Anna kennt **die Mutter von Bert**.“ $\rightsquigarrow X \wedge Y$

„Anna kennt Bert und Anna **mag** Bert.“ $\rightsquigarrow X \wedge Y$

„Anna kennt Bert und **Claus** kennt Bert.“ $\rightsquigarrow X \wedge Y$

■ Schlussfolgerung

Prämissen: „Tux ist ein Pinguin.“ $\rightsquigarrow A$

„Kein Pinguin kann fliegen.“ $\rightsquigarrow \neg B$

„Alle Pinguine sind Vögel.“ $\rightsquigarrow C$

Konklusion: „Manche Vögel können nicht fliegen.“ $\rightsquigarrow D$

- Formalisierung zu grob: kann inneren Aufbau der Elementaraussagen nicht abbilden
- Zusammenhänge werden nicht klar („kann fliegen“, „Pinguin“, „Vogel“)
- Konklusion folgt nicht aus Prämissen: $A \wedge \neg B \wedge C \wedge \neg D$ ist erfüllbar

Konstantensymbole einzelne Individuen

Konstantensymbole einzelne Individuen

Prädikate Beziehungen zwischen Individuen

Konstantensymbole einzelne Individuen

Prädikate Beziehungen zwischen Individuen

Funktionssymbole Abbildungen von Individuen auf Individuen

Konstantensymbole einzelne Individuen

Prädikate Beziehungen zwischen Individuen

Funktionssymbole Abbildungen von Individuen auf Individuen

Konstantensymbole einzelne Individuen

Prädikate Beziehungen zwischen Individuen

Funktionssymbole Abbildungen von Individuen auf Individuen

Beispiel 4.2

„Anna kennt Bert und Anna kennt die Mutter von Bert.“ $\rightsquigarrow K(a, b) \wedge K(a, m(b))$

Konstantensymbole einzelne Individuen

Prädikate Beziehungen zwischen Individuen

Funktionssymbole Abbildungen von Individuen auf Individuen

Beispiel 4.2

„Anna kennt Bert und Anna kennt die Mutter von Bert.“	$\rightsquigarrow$	$K(a, b) \wedge K(a, m(b))$
„Anna kennt Bert und Anna mag Bert.“	$\rightsquigarrow$	$K(a, b) \wedge M(a, b)$

Konstantensymbole einzelne Individuen

Prädikate Beziehungen zwischen Individuen

Funktionssymbole Abbildungen von Individuen auf Individuen

Beispiel 4.2

„Anna kennt Bert und Anna kennt die Mutter von Bert.“	$\rightsquigarrow$	$K(a, b) \wedge K(a, m(b))$
„Anna kennt Bert und Anna mag Bert.“	$\rightsquigarrow$	$K(a, b) \wedge M(a, b)$
„Anna kennt Bert und Claus kennt Bert.“	$\rightsquigarrow$	$K(a, b) \wedge K(c, b)$

Konstantensymbole einzelne Individuen

Prädikate Beziehungen zwischen Individuen

Funktionssymbole Abbildungen von Individuen auf Individuen

Beispiel 4.2

„Anna kennt Bert und Anna kennt die Mutter von Bert.“	$\rightsquigarrow$	$K(a, b) \wedge K(a, m(b))$
„Anna kennt Bert und Anna mag Bert.“	$\rightsquigarrow$	$K(a, b) \wedge M(a, b)$
„Anna kennt Bert und Claus kennt Bert.“	$\rightsquigarrow$	$K(a, b) \wedge K(c, b)$
„Tux ist ein Pinguin.“	$\rightsquigarrow$	$P(t)$

Konstantensymbole einzelne Individuen

Prädikate Beziehungen zwischen Individuen

Funktionssymbole Abbildungen von Individuen auf Individuen

Beispiel 4.2

„Anna kennt Bert und Anna kennt die Mutter von Bert.“	$\rightsquigarrow$	$K(a, b) \wedge K(a, m(b))$
„Anna kennt Bert und Anna mag Bert.“	$\rightsquigarrow$	$K(a, b) \wedge M(a, b)$
„Anna kennt Bert und Claus kennt Bert.“	$\rightsquigarrow$	$K(a, b) \wedge K(c, b)$
„Tux ist ein Pinguin.“	$\rightsquigarrow$	$P(t)$
„Kein Pinguin kann fliegen.“	$\rightsquigarrow$	$\forall x(P(x) \rightarrow \neg F(x))$

Konstantensymbole einzelne Individuen

Prädikate Beziehungen zwischen Individuen

Funktionssymbole Abbildungen von Individuen auf Individuen

Beispiel 4.2

„Anna kennt Bert und Anna kennt die Mutter von Bert.“	$\rightsquigarrow$	$K(a, b) \wedge K(a, m(b))$
„Anna kennt Bert und Anna mag Bert.“	$\rightsquigarrow$	$K(a, b) \wedge M(a, b)$
„Anna kennt Bert und Claus kennt Bert.“	$\rightsquigarrow$	$K(a, b) \wedge K(c, b)$
„Tux ist ein Pinguin.“	$\rightsquigarrow$	$P(t)$
„Kein Pinguin kann fliegen.“	$\rightsquigarrow$	$\forall x(P(x) \rightarrow \neg F(x))$
„Alle Pinguine sind Vögel.“	$\rightsquigarrow$	$\forall x(P(x) \rightarrow V(x))$

Konstantensymbole einzelne Individuen

Prädikate Beziehungen zwischen Individuen

Funktionssymbole Abbildungen von Individuen auf Individuen

Beispiel 4.2

„Anna kennt Bert und Anna kennt die Mutter von Bert.“	$\rightsquigarrow$	$K(a, b) \wedge K(a, m(b))$
„Anna kennt Bert und Anna mag Bert.“	$\rightsquigarrow$	$K(a, b) \wedge M(a, b)$
„Anna kennt Bert und Claus kennt Bert.“	$\rightsquigarrow$	$K(a, b) \wedge K(c, b)$
„Tux ist ein Pinguin.“	$\rightsquigarrow$	$P(t)$
„Kein Pinguin kann fliegen.“	$\rightsquigarrow$	$\forall x(P(x) \rightarrow \neg F(x))$
„Alle Pinguine sind Vögel.“	$\rightsquigarrow$	$\forall x(P(x) \rightarrow V(x))$
„Manche Vögel können nicht fliegen.“	$\rightsquigarrow$	$\exists x(V(x) \wedge \neg F(x))$

1. Einführung

2. Mengen

3. Aussagenlogik

4. Prädikatenlogik

4.1 Relationen und Funktionen

4.2 Syntax

4.3 Semantik

4.4 Schlussfolgerungsverfahren

5. Prolog

Definition 4.3 (Kartesisches Produkt)

Das **kartesische Produkt** ($M_1 \times M_2$) zweier Mengen M_1 und M_2 ist die Menge $\{(x, y) \mid x \in M_1, y \in M_2\}$.



René Descartes
(1596–1650)

Definition 4.3 (Kartesisches Produkt)

Das **kartesische Produkt** ($M_1 \times M_2$) zweier Mengen M_1 und M_2 ist die Menge $\{(x, y) \mid x \in M_1, y \in M_2\}$.

- $M_1 \times M_2$ ist eine Menge von Paaren oder 2-Tupeln



René Descartes
(1596–1650)

Definition 4.3 (Kartesisches Produkt)

Das **kartesische Produkt** ($M_1 \times M_2$) zweier Mengen M_1 und M_2 ist die Menge $\{(x, y) \mid x \in M_1, y \in M_2\}$.

- $M_1 \times M_2$ ist eine Menge von Paaren oder 2-Tupeln
- Verallgemeinerung:
 $M_1 \times M_2 \times \dots \times M_n = \{(x_1, x_2, \dots, x_n) \mid x_i \in M_i\}$
ist eine Menge von n -Tupeln



René Descartes
(1596–1650)

Definition 4.3 (Kartesisches Produkt)

Das **kartesische Produkt** ($M_1 \times M_2$) zweier Mengen M_1 und M_2 ist die Menge $\{(x, y) \mid x \in M_1, y \in M_2\}$.

- $M_1 \times M_2$ ist eine Menge von Paaren oder 2-Tupeln
- Verallgemeinerung:
 $M_1 \times M_2 \times \dots \times M_n = \{(x_1, x_2, \dots, x_n) \mid x_i \in M_i\}$
ist eine Menge von n -Tupeln



René Descartes
(1596–1650)

Beispiel 4.4

$M_1 = \{1, 2, 3\}, M_2 = \{a, b\}$

- $M_1 \times M_2 = \{(1, a), (2, a), (3, a), (1, b), (2, b), (3, b)\}$

Definition 4.3 (Kartesisches Produkt)

Das **kartesische Produkt** ($M_1 \times M_2$) zweier Mengen M_1 und M_2 ist die Menge $\{(x, y) \mid x \in M_1, y \in M_2\}$.

- $M_1 \times M_2$ ist eine Menge von Paaren oder 2-Tupeln
- Verallgemeinerung:
 $M_1 \times M_2 \times \dots \times M_n = \{(x_1, x_2, \dots, x_n) \mid x_i \in M_i\}$
ist eine Menge von n -Tupeln



René Descartes
(1596–1650)

Beispiel 4.4

$M_1 = \{1, 2, 3\}$, $M_2 = \{a, b\}$

- $M_1 \times M_2 = \{(1, a), (2, a), (3, a), (1, b), (2, b), (3, b)\}$
- $M_2 \times M_1 = ?$

Definition 4.3 (Kartesisches Produkt)

Das **kartesische Produkt** ($M_1 \times M_2$) zweier Mengen M_1 und M_2 ist die Menge $\{(x, y) \mid x \in M_1, y \in M_2\}$.

- $M_1 \times M_2$ ist eine Menge von Paaren oder 2-Tupeln
- Verallgemeinerung:
 $M_1 \times M_2 \times \dots \times M_n = \{(x_1, x_2, \dots, x_n) \mid x_i \in M_i\}$
ist eine Menge von n -Tupeln



René Descartes
(1596–1650)

Beispiel 4.4

$M_1 = \{1, 2, 3\}$, $M_2 = \{a, b\}$

- $M_1 \times M_2 = \{(1, a), (2, a), (3, a), (1, b), (2, b), (3, b)\}$
- $M_2 \times M_1 = ?$
- $M_1 \times M_1 = ?$

Übung 4.5

Sei $M_1 = \{1, 3, 5, 7\}$, $M_2 = \{2, 4, 6\}$.

Berechnen Sie:

1 $M_1 \times M_1$

2 $M_1 \times M_2$

3 $M_2 \times M_1$

Definition 4.6 (Relation)

Seien $M_1, M_2, \dots, M_n$ Mengen. Eine (n-stellige) Relation R über $M_1, M_2, \dots, M_n$ ist eine Teilmenge des kartesischen Produkts der Mengen, also $R \subseteq M_1 \times M_2 \times \dots \times M_n$.

Definition 4.6 (Relation)

Seien $M_1, M_2, \dots, M_n$ Mengen. Eine (n-stellige) Relation R über $M_1, M_2, \dots, M_n$ ist eine Teilmenge des kartesischen Produkts der Mengen, also $R \subseteq M_1 \times M_2 \times \dots \times M_n$.

Notation: Statt $(x, y) \in R$ schreibt man oft $R(x, y)$

Definition 4.6 (Relation)

Seien $M_1, M_2, \dots, M_n$ Mengen. Eine (n-stellige) Relation R über $M_1, M_2, \dots, M_n$ ist eine Teilmenge des kartesischen Produkts der Mengen, also $R \subseteq M_1 \times M_2 \times \dots \times M_n$.

Notation: Statt $(x, y) \in R$ schreibt man oft $R(x, y)$

Beispiel 4.7

- $M_1 = \{\text{Müller, Mayer, Schulze, Schmidt, Becker}\}$ (z. B. Menschen)

Definition 4.6 (Relation)

Seien $M_1, M_2, \dots, M_n$ Mengen. Eine (n-stellige) Relation R über $M_1, M_2, \dots, M_n$ ist eine Teilmenge des kartesischen Produkts der Mengen, also $R \subseteq M_1 \times M_2 \times \dots \times M_n$.

Notation: Statt $(x, y) \in R$ schreibt man oft $R(x, y)$

Beispiel 4.7

- $M_1 = \{\text{Müller, Mayer, Schulze, Schmidt, Becker}\}$ (z. B. Menschen)
- $M_2 = \{\text{Logik, LA, BWL, Digitaltechnik, PM}\}$ (z. B. Kurse)

Definition 4.6 (Relation)

Seien $M_1, M_2, \dots, M_n$ Mengen. Eine (n-stellige) Relation R über $M_1, M_2, \dots, M_n$ ist eine Teilmenge des kartesischen Produkts der Mengen, also $R \subseteq M_1 \times M_2 \times \dots \times M_n$.

Notation: Statt $(x, y) \in R$ schreibt man oft $R(x, y)$

Beispiel 4.7

- $M_1 = \{\text{Müller, Mayer, Schulze, Schmidt, Becker}\}$ (z. B. Menschen)
- $M_2 = \{\text{Logik, LA, BWL, Digitaltechnik, PM}\}$ (z. B. Kurse)
- $\text{Belegt} = \{(\text{Müller, Logik}), (\text{Müller, BWL}), (\text{Müller, Digitaltechnik}), (\text{Mayer, BWL}), (\text{Mayer, PM}), (\text{Schulze, LA}), (\text{Schulze, Digitaltechnik}), (\text{Schmidt, PM})\}$

Definition 4.6 (Relation)

Seien $M_1, M_2, \dots, M_n$ Mengen. Eine (n-stellige) Relation R über $M_1, M_2, \dots, M_n$ ist eine Teilmenge des kartesischen Produkts der Mengen, also $R \subseteq M_1 \times M_2 \times \dots \times M_n$.

Notation: Statt $(x, y) \in R$ schreibt man oft $R(x, y)$

Beispiel 4.7

- $M_1 = \{\text{Müller, Mayer, Schulze, Schmidt, Becker}\}$ (z. B. Menschen)
- $M_2 = \{\text{Logik, LA, BWL, Digitaltechnik, PM}\}$ (z. B. Kurse)
- $\text{Belegt} = \{(\text{Müller, Logik}), (\text{Müller, BWL}), (\text{Müller, Digitaltechnik}), (\text{Mayer, BWL}), (\text{Mayer, PM}), (\text{Schulze, LA}), (\text{Schulze, Digitaltechnik}), (\text{Schmidt, PM})\}$

Definition 4.6 (Relation)

Seien $M_1, M_2, \dots, M_n$ Mengen. Eine (**n-stellige**) **Relation** R über $M_1, M_2, \dots, M_n$ ist eine Teilmenge des kartesischen Produkts der Mengen, also $R \subseteq M_1 \times M_2 \times \dots \times M_n$.

Notation: Statt $(x, y) \in R$ schreibt man oft $R(x, y)$

Beispiel 4.7

- $M_1 = \{\text{Müller, Mayer, Schulze, Schmidt, Becker}\}$ (z. B. Menschen)
- $M_2 = \{\text{Logik, LA, BWL, Digitaltechnik, PM}\}$ (z. B. Kurse)
- $\text{Belegt} = \{(\text{Müller, Logik}), (\text{Müller, BWL}), (\text{Müller, Digitaltechnik}), (\text{Mayer, BWL}), (\text{Mayer, PM}), (\text{Schulze, LA}), (\text{Schulze, Digitaltechnik}), (\text{Schmidt, PM})\}$
- Welche Kurse hat Mayer / Schmidt belegt?

Übung 4.8

Geben Sie jeweils ein Beispiel für eine **möglichst interessante** Relation aus dem realen Leben und aus der Mathematik an.

- Welche Mengen sind beteiligt?
- Welche Elemente stehen in Relation?

Definition 4.9 (Homogenität)

Sei R eine Relation über $M_1, M_2, \dots, M_n$. R heißt **homogen**, falls $M_i = M_j$ für alle $i, j \in \{1, \dots, n\}$ gilt.

- Wenn R homogen ist, so nennen wir R auch eine **n -stellige Relation über M** .

Definition 4.9 (Homogenität)

Sei R eine Relation über $M_1, M_2, \dots, M_n$. R heißt **homogen**, falls $M_i = M_j$ für alle $i, j \in \{1, \dots, n\}$ gilt.

- Wenn R homogen ist, so nennen wir R auch eine **n -stellige Relation über M** .
- Eine **n -stellige Relation $R^{(n)}$** über einer Menge M ist

Definition 4.9 (Homogenität)

Sei R eine Relation über $M_1, M_2, \dots, M_n$. R heißt **homogen**, falls $M_i = M_j$ für alle $i, j \in \{1, \dots, n\}$ gilt.

- Wenn R homogen ist, so nennen wir R auch eine **n -stellige Relation über M** .
- Eine **n -stellige Relation $R^{(n)}$** über einer Menge M ist
 - eine Teilmenge von $M^n = \underbrace{M \times M \times \dots \times M}_{n\text{-mal}}$

Definition 4.9 (Homogenität)

Sei R eine Relation über $M_1, M_2, \dots, M_n$. R heißt **homogen**, falls $M_i = M_j$ für alle $i, j \in \{1, \dots, n\}$ gilt.

- Wenn R homogen ist, so nennen wir R auch eine **n -stellige Relation über M** .
- Eine **n -stellige Relation $R^{(n)}$** über einer Menge M ist
 - eine Teilmenge von $M^n = \underbrace{M \times M \times \dots \times M}_{n\text{-mal}}$
 - eine Menge von n -Tupeln von Elementen aus M

Beispiel 4.10

- Einstellige (unäre) Relationen sind Teilmengen von M

Beispiel 4.10

- Einstellige (unäre) Relationen sind Teilmengen von M
 - $P_{\mathbb{N}}^{(1)} = \{2, 3, 5, 7, 11, \dots\}$ über $\mathbb{N}$

Beispiel 4.10

- Einstellige (unäre) Relationen sind Teilmengen von M
 - $P_{\mathbb{N}}^{(1)} = \{2, 3, 5, 7, 11, \dots\}$ über $\mathbb{N}$
- Zweistellige (binäre) Relationen bestehen aus Paaren

Beispiel 4.10

- Einstellige (unäre) Relationen sind Teilmengen von M
 - $P_{\mathbb{N}}^{(1)} = \{2, 3, 5, 7, 11, \dots\}$ über $\mathbb{N}$
- Zweistellige (binäre) Relationen bestehen aus Paaren
 - $T_{\mathbb{N}}^{(2)} = \{(2, 2), (2, 4), (2, 6), \dots, (3, 3), (3, 6), (3, 9), \dots\}$ über $\mathbb{N}$

Beispiel 4.10

- Einstellige (unäre) Relationen sind Teilmengen von M
 - $P_{\mathbb{N}}^{(1)} = \{2, 3, 5, 7, 11, \dots\}$ über $\mathbb{N}$
- Zweistellige (binäre) Relationen bestehen aus Paaren
 - $T_{\mathbb{N}}^{(2)} = \{(2, 2), (2, 4), (2, 6), \dots, (3, 3), (3, 6), (3, 9), \dots\}$ über $\mathbb{N}$
 - $<_{\mathbb{N}}^{(2)} = \{(0, 1), (0, 2), \dots, (1, 2), (1, 3), \dots\}$ über $\mathbb{N}$

Beispiel 4.10

- Einstellige (unäre) Relationen sind Teilmengen von M
 - $P_{\mathbb{N}}^{(1)} = \{2, 3, 5, 7, 11, \dots\}$ über $\mathbb{N}$
- Zweistellige (binäre) Relationen bestehen aus Paaren
 - $T_{\mathbb{N}}^{(2)} = \{(2, 2), (2, 4), (2, 6), \dots, (3, 3), (3, 6), (3, 9), \dots\}$ über $\mathbb{N}$
 - $<_{\mathbb{N}}^{(2)} = \{(0, 1), (0, 2), \dots, (1, 2), (1, 3), \dots\}$ über $\mathbb{N}$
 - Bei binären Relationen schreiben wir oft xRy statt $R(x, y)$.
(z. B. $1 <_{\mathbb{N}} 2$ statt $<_{\mathbb{N}}(1, 2)$ oder $(1, 2) \in <_{\mathbb{N}}$)

Beispiel 4.10

- Einstellige (unäre) Relationen sind Teilmengen von M
 - $P_{\mathbb{N}}^{(1)} = \{2, 3, 5, 7, 11, \dots\}$ über $\mathbb{N}$
- Zweistellige (binäre) Relationen bestehen aus Paaren
 - $T_{\mathbb{N}}^{(2)} = \{(2, 2), (2, 4), (2, 6), \dots, (3, 3), (3, 6), (3, 9), \dots\}$ über $\mathbb{N}$
 - $<_{\mathbb{N}}^{(2)} = \{(0, 1), (0, 2), \dots, (1, 2), (1, 3), \dots\}$ über $\mathbb{N}$
 - Bei binären Relationen schreiben wir oft xRy statt $R(x, y)$.
(z. B. $1 <_{\mathbb{N}} 2$ statt $<_{\mathbb{N}}(1, 2)$ oder $(1, 2) \in <_{\mathbb{N}}$)
- Nullstellige Relationen sind leer oder enthalten das **leere Tupel** $()$

Beispiel 4.10

- Einstellige (unäre) Relationen sind Teilmengen von M
 - $P_{\mathbb{N}}^{(1)} = \{2, 3, 5, 7, 11, \dots\}$ über $\mathbb{N}$
- Zweistellige (binäre) Relationen bestehen aus Paaren
 - $T_{\mathbb{N}}^{(2)} = \{(2, 2), (2, 4), (2, 6), \dots, (3, 3), (3, 6), (3, 9), \dots\}$ über $\mathbb{N}$
 - $<_{\mathbb{N}}^{(2)} = \{(0, 1), (0, 2), \dots, (1, 2), (1, 3), \dots\}$ über $\mathbb{N}$
 - Bei binären Relationen schreiben wir oft xRy statt $R(x, y)$.
(z. B. $1 <_{\mathbb{N}} 2$ statt $<_{\mathbb{N}}(1, 2)$ oder $(1, 2) \in <_{\mathbb{N}}$)
- Nullstellige Relationen sind leer oder enthalten das **leere Tupel** $()$
 - die einzigen nullstelligen Relation über jeder Menge sind
 $F^{(0)} = \{\}$ und $W^{(0)} = \{()\}$

Beispiel 4.11

$$\mathbf{I} =_{\mathbb{N}} = \{(0, 0), (1, 1), (2, 2), \dots\}$$

Beispiel 4.11

- $=_{\mathbb{N}} = \{(0, 0), (1, 1), (2, 2), \dots\}$
- $<_{\mathbb{Z}} = \{(i, i + j) \mid i \in \mathbb{Z}, j \in \mathbb{N}^{\geq 1}\}$

Beispiel 4.11

- $=_{\mathbb{N}} = \{(0, 0), (1, 1), (2, 2), \dots\}$
- $<_{\mathbb{Z}} = \{(i, i + j) \mid i \in \mathbb{Z}, j \in \mathbb{N}^{\geq 1}\}$
- $\neq_N$ mit $N = \{w \mid w \text{ ist ein Nachname}\}$

Beispiel 4.11

- $=_{\mathbb{N}} = \{(0, 0), (1, 1), (2, 2), \dots\}$
- $<_{\mathbb{Z}} = \{(i, i + j) \mid i \in \mathbb{Z}, j \in \mathbb{N}^{\geq 1}\}$
- $\neq_N$ mit $N = \{w \mid w \text{ ist ein Nachname}\}$
 - $\{(Müller, Mayer), (Müller, Schulze), (Mayer, Schulze), (Mayer, Müller), (Schulze, Mayer), (Schulze, Müller)\} \subseteq \neq_N$

Definition 4.12 (linkstotal, rechtseindeutig)

Sei R eine binäre Relation über M und N .

Definition 4.12 (linkstotal, rechtseindeutig)

Sei R eine binäre Relation über M und N .

- Gibt es für jedes $x \in M$ ein $y \in N$ mit $R(x, y)$, so heißt R **linkstotal**.

Definition 4.12 (linkstotal, rechtseindeutig)

Sei R eine binäre Relation über M und N .

- Gibt es für jedes $x \in M$ ein $y \in N$ mit $R(x, y)$, so heißt R **linkstotal**.
- Gilt für alle $x \in M$ und $y, z \in N$, dass aus $R(x, y)$ und $R(x, z)$ folgt, dass auch $y = z$ gilt, so heißt R **rechtseindeutig**.

Definition 4.12 (linkstotal, rechtseindeutig)

Sei R eine binäre Relation über M und N .

- Gibt es für jedes $x \in M$ ein $y \in N$ mit $R(x, y)$, so heißt R **linkstotal**.
 - Gilt für alle $x \in M$ und $y, z \in N$, dass aus $R(x, y)$ und $R(x, z)$ folgt, dass auch $y = z$ gilt, so heißt R **rechtseindeutig**.
-
- Linkstotal: Jedes Element aus M steht mit mindestens einem Element aus N in Relation.

Definition 4.12 (linkstotal, rechtseindeutig)

Sei R eine binäre Relation über M und N .

- Gibt es für jedes $x \in M$ ein $y \in N$ mit $R(x, y)$, so heißt R **linkstotal**.
 - Gilt für alle $x \in M$ und $y, z \in N$, dass aus $R(x, y)$ und $R(x, z)$ folgt, dass auch $y = z$ gilt, so heißt R **rechtseindeutig**.
-
- Linkstotal: Jedes Element aus M steht mit mindestens einem Element aus N in Relation.
 - Rechtseindeutig: Jedes Element aus M steht mit höchstens einem Element aus N in Relation.

Definition 4.13 (Funktion)

Seien M, N Mengen.

Definition 4.13 (Funktion)

Seien M, N Mengen.

- Eine **partielle Funktion** $f : M \rightarrow N$ ist eine Relation $f \subseteq (M \times N)$, die rechtseindeutig ist.

Definition 4.13 (Funktion)

Seien M, N Mengen.

- Eine **partielle Funktion** $f : M \rightarrow N$ ist eine Relation $f \subseteq (M \times N)$, die rechtseindeutig ist.
- Eine **(totale) Funktion** $f : M \rightarrow N$ ist eine Relation $f \subseteq (M \times N)$, die linkstotal und rechtseindeutig ist.

Definition 4.13 (Funktion)

Seien M, N Mengen.

- Eine **partielle Funktion** $f : M \rightarrow N$ ist eine Relation $f \subseteq (M \times N)$, die rechtseindeutig ist.
- Eine **(totale) Funktion** $f : M \rightarrow N$ ist eine Relation $f \subseteq (M \times N)$, die linkstotal und rechtseindeutig ist.
- M heißt **Definitionsmenge** von f .

Definition 4.13 (Funktion)

Seien M, N Mengen.

- Eine **partielle Funktion** $f : M \rightarrow N$ ist eine Relation $f \subseteq (M \times N)$, die rechtseindeutig ist.
- Eine **(totale) Funktion** $f : M \rightarrow N$ ist eine Relation $f \subseteq (M \times N)$, die linkstotal und rechtseindeutig ist.
- M heißt **Definitionsmenge** von f .
- N heißt **Zielmenge** von f .

Definition 4.13 (Funktion)

Seien M, N Mengen.

- Eine **partielle Funktion** $f : M \rightarrow N$ ist eine Relation $f \subseteq (M \times N)$, die rechtseindeutig ist.
- Eine **(totale) Funktion** $f : M \rightarrow N$ ist eine Relation $f \subseteq (M \times N)$, die linkstotal und rechtseindeutig ist.
- M heißt **Definitionsmenge** von f .
- N heißt **Zielmenge** von f .

Definition 4.13 (Funktion)

Seien M, N Mengen.

- Eine **partielle Funktion** $f : M \rightarrow N$ ist eine Relation $f \subseteq (M \times N)$, die rechtseindeutig ist.
- Eine **(totale) Funktion** $f : M \rightarrow N$ ist eine Relation $f \subseteq (M \times N)$, die linkstotal und rechtseindeutig ist.
- M heißt **Definitionsmenge** von f .
- N heißt **Zielmenge** von f .

Eine Funktion (auch: **Abbildung**) ordnet (jedem) Element der Definitionsmenge **höchstens** ein Element der Zielmenge zu.

Definition 4.14 (Notation von Funktionen)

Funktionen können mit der folgenden Schreibweise angegeben werden:

Funktionssymbol : Definitionsmenge $\rightarrow$ Zielmenge ; Zuordnungsvorschrift

Definition 4.14 (Notation von Funktionen)

Funktionen können mit der folgenden Schreibweise angegeben werden:

Funktionssymbol : Definitionsmenge $\rightarrow$ Zielmenge ; Zuordnungsvorschrift

Man schreibt dann auch $f(x) = y$ statt $(x, y) \in f$ oder xfy .

Definition 4.14 (Notation von Funktionen)

Funktionen können mit der folgenden Schreibweise angegeben werden:

Funktionssymbol : Definitionsmenge $\rightarrow$ Zielmenge ; Zuordnungsvorschrift

Man schreibt dann auch $f(x) = y$ statt $(x, y) \in f$ oder xfy .

Für endliche Definitionsmengen kann man eine Funktion als Menge der einzelnen Zuordnungen angeben.

Definition 4.14 (Notation von Funktionen)

Funktionen können mit der folgenden Schreibweise angegeben werden:

Funktionssymbol : Definitionsmenge $\rightarrow$ Zielmenge ; Zuordnungsvorschrift

Man schreibt dann auch $f(x) = y$ statt $(x, y) \in f$ oder xfy .

Für endliche Definitionsmengen kann man eine Funktion als Menge der einzelnen Zuordnungen angeben.

Beispiel 4.15

■ $f : \mathbb{N} \rightarrow \mathbb{N}; x \mapsto x^2$

Definition 4.14 (Notation von Funktionen)

Funktionen können mit der folgenden Schreibweise angegeben werden:

Funktionssymbol : Definitionsmenge $\rightarrow$ Zielmenge ; Zuordnungsvorschrift

Man schreibt dann auch $f(x) = y$ statt $(x, y) \in f$ oder xfy .

Für endliche Definitionsmengen kann man eine Funktion als Menge der einzelnen Zuordnungen angeben.

Beispiel 4.15

- $f : \mathbb{N} \rightarrow \mathbb{N}; x \mapsto x^2$
- $g : \mathbb{Z} \rightarrow \mathbb{N}; x \mapsto |x|$

Definition 4.14 (Notation von Funktionen)

Funktionen können mit der folgenden Schreibweise angegeben werden:

Funktionssymbol : Definitionsmenge $\rightarrow$ Zielmenge ; Zuordnungsvorschrift

Man schreibt dann auch $f(x) = y$ statt $(x, y) \in f$ oder xfy .

Für endliche Definitionsmengen kann man eine Funktion als Menge der einzelnen Zuordnungen angeben.

Beispiel 4.15

- $f : \mathbb{N} \rightarrow \mathbb{N}; x \mapsto x^2$
- $g : \mathbb{Z} \rightarrow \mathbb{N}; x \mapsto |x|$
- $h : \{A, B, C\} \rightarrow \mathbb{B}; \{A \mapsto 1, B \mapsto 0, C \mapsto 1\}$

Definition 4.16 (n -stellige Funktion auf einer Menge)

Sei M eine Menge.

Eine Funktion $f : M^n \rightarrow M$ heißt n -stellige Funktion oder Operation auf M .

Definition 4.16 (n -stellige Funktion auf einer Menge)

Sei M eine Menge.

Eine Funktion $f : M^n \rightarrow M$ heißt n -stellige Funktion oder Operation auf M .

Beispiel 4.17

- $+$ _{$\mathbb{N}$} , die Addition der natürlichen Zahlen, ist eine zweistellige Operation auf $\mathbb{N}$.
Z. B.: $(2, 3) \mapsto 5$

Definition 4.16 (n -stellige Funktion auf einer Menge)

Sei M eine Menge.

Eine Funktion $f : M^n \rightarrow M$ heißt n -stellige Funktion oder Operation auf M .

Beispiel 4.17

- $+$, die Addition der natürlichen Zahlen, ist eine zweistellige Operation auf $\mathbb{N}$.
Z. B.: $(2, 3) \mapsto 5$
- $!$, die Fakultät der natürlichen Zahlen, ist eine einstellige Operation auf $\mathbb{N}$.
Z. B.: $4 \mapsto 24$

- Funktion $f : M^0 \rightarrow M$

- Funktion $f : M^0 \rightarrow M$
- $M^0 = \{()\}$

- Funktion $f : M^0 \rightarrow M$
- $M^0 = \{()\}$
- Definitionsmenge hat nur ein Element

- Funktion $f : M^0 \rightarrow M$
- $M^0 = \{()\}$
- Definitionsmenge hat nur ein Element
- f ist durch $f(())$ eindeutig festgelegt

- Funktion $f : M^0 \rightarrow M$
- $M^0 = \{()\}$
- Definitionsmenge hat nur ein Element
- f ist durch $f(())$ eindeutig festgelegt
- Nullstellige Funktionen beschreiben Konstanten!

- Funktion $f : M^0 \rightarrow M$
- $M^0 = \{()\}$
- Definitionsmenge hat nur ein Element
- f ist durch $f(())$ eindeutig festgelegt
- Nullstellige Funktionen beschreiben Konstanten!

- Funktion $f : M^0 \rightarrow M$
- $M^0 = \{()\}$
- Definitionsmenge hat nur ein Element
- f ist durch $f(())$ eindeutig festgelegt
- Nullstellige Funktionen beschreiben Konstanten!

Beispiel 4.18

- $c : \mathbb{N}^0 \rightarrow \mathbb{N}; () \mapsto 5$ ist die Konstante 5

- Funktion $f : M^0 \rightarrow M$
- $M^0 = \{()\}$
- Definitionsmenge hat nur ein Element
- f ist durch $f(())$ eindeutig festgelegt
- Nullstellige Funktionen beschreiben Konstanten!

Beispiel 4.18

- $c : \mathbb{N}^0 \rightarrow \mathbb{N}; () \mapsto 5$ ist die Konstante 5
- Für die Menge aller Menschen M bezeichnet $d : M^0 \rightarrow M; () \mapsto \text{Kurt Gödel}$ den einzelnen Menschen Kurt Gödel.

Funktionssymbol Zeichen, z. B. „+“

Funktionssymbol Zeichen, z. B. „+“
zugeordnete Stelligkeit: „+⁽²⁾“

Funktionssymbol Zeichen, z. B. „+“

zugeordnete **Stelligkeit**: „+⁽²⁾“

„ $x +^{(2)} y$ “ ist korrekter Ausdruck, „ $x +^{(2)}$ “ nicht

Funktionssymbol Zeichen, z. B. „+“
zugeordnete **Stelligkeit**: „+⁽²⁾“
„ $x +^{(2)} y$ “ ist korrekter Ausdruck, „ $x +^{(2)}$ “ nicht
wenn Stelligkeit klar ist, wird sie oft nicht angegeben

Funktionssymbol Zeichen, z. B. „+“
zugeordnete **Stelligkeit**: „+⁽²⁾“
„ $x +^{(2)} y$ “ ist korrekter Ausdruck, „ $x +^{(2)}$ “ nicht
wenn Stelligkeit klar ist, wird sie oft nicht angegeben
auch: **Operator**

Funktionssymbol Zeichen, z. B. „+“
zugeordnete **Stelligkeit**: „+⁽²⁾“
„ $x +^{(2)} y$ “ ist korrekter Ausdruck, „ $x +^{(2)}$ “ nicht
wenn Stelligkeit klar ist, wird sie oft nicht angegeben
auch: **Operator**

Funktion Mathematisches Konstrukt

Funktionssymbol Zeichen, z. B. „+“
zugeordnete **Stelligkeit**: „+⁽²⁾“
„ $x +^{(2)} y$ “ ist korrekter Ausdruck, „ $x +^{(2)}$ “ nicht
wenn Stelligkeit klar ist, wird sie oft nicht angegeben
auch: **Operator**

Funktion Mathematisches Konstrukt
z. B. $+_{\mathbb{N}}$: Abbildung von $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ (zweistellig)

Funktionssymbol Zeichen, z. B. „+“
zugeordnete **Stelligkeit**: „ $+^{(2)}$ “
„ $x +^{(2)} y$ “ ist korrekter Ausdruck, „ $x +^{(2)}$ “ nicht
wenn Stelligkeit klar ist, wird sie oft nicht angegeben
auch: **Operator**

Funktion Mathematisches Konstrukt
z. B. $+\mathbb{N}$: Abbildung von $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ (zweistellig)
auch: **Operation**

Funktionssymbol Zeichen, z. B. „+“
zugeordnete **Stelligkeit**: „+⁽²⁾“
„ $x +^{(2)} y$ “ ist korrekter Ausdruck, „ $x +^{(2)}$ “ nicht
wenn Stelligkeit klar ist, wird sie oft nicht angegeben
auch: **Operator**

Funktion Mathematisches Konstrukt
z. B. $+_N$: Abbildung von $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ (zweistellig)
auch: **Operation**

Achtung

Dasselbe **Funktionssymbol** kann für verschiedene Trägermengen durch unterschiedliche **Funktionen** interpretiert werden!

Beispiel 4.19 (Symbol „ $\cdot^{(2)}$ “)

- Funktion $\cdot_{\mathbb{N}} : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}; (x, y) \mapsto x \cdot y$ (Multiplikation)
z. B. $2 \cdot_{\mathbb{N}} 3 = 6$

Beispiel 4.19 (Symbol „ $\cdot^{(2)}$ “)

- Funktion $\cdot_{\mathbb{N}} : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}; (x, y) \mapsto x \cdot y$ (Multiplikation)
z. B. $2 \cdot_{\mathbb{N}} 3 = 6$
- Funktion $\cdot_S : S \times S \rightarrow S; (x, y) \mapsto xy$ (Konkatenation)
 S : Menge der Strings über $\{a, b, c, \dots, z\}$
z. B. $ab \cdot_S bc = abbc$

Beispiel 4.19 (Symbol „ $\cdot^{(2)}$ “)

- Funktion $\cdot_{\mathbb{N}} : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}; (x, y) \mapsto x \cdot y$ (Multiplikation)
z. B. $2 \cdot_{\mathbb{N}} 3 = 6$
- Funktion $\cdot_S : S \times S \rightarrow S; (x, y) \mapsto xy$ (Konkatenation)
 S : Menge der Strings über $\{a, b, c, \dots, z\}$
z. B. $ab \cdot_S bc = abbc$

Beispiel 4.19 (Symbol „ $\cdot^{(2)}$ “)

- Funktion $\cdot_{\mathbb{N}} : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}; (x, y) \mapsto x \cdot y$ (Multiplikation)
z. B. $2 \cdot_{\mathbb{N}} 3 = 6$
- Funktion $\cdot_S : S \times S \rightarrow S; (x, y) \mapsto xy$ (Konkatenation)
 S : Menge der Strings über $\{a, b, c, \dots, z\}$
z. B. $ab \cdot_S bc = abbc$

Analogie aus der OO-Programmierung: **Überladen** von Operatoren

- ein Symbol „+“
- **unterschiedliche** Funktionen je nach Typ der Objekte
 - Integer-Addition
 - Matrix-Addition
 - Vereinigung von Mengen
 - ...

Analog unterscheidet man Relationen und Prädikate:

Prädikat Zeichen „ $<$ “

Analog unterscheidet man Relationen und Prädikate:

Prädikat Zeichen „ $<$ “
Stelligkeit $<^{(2)}$

Analog unterscheidet man Relationen und Prädikate:

Prädikat Zeichen „ $<$ “

Stelligkeit $<^{(2)}$

$3 < 4$ ist korrekter Ausdruck

Analog unterscheidet man Relationen und Prädikate:

Prädikat Zeichen „ $<$ “

Stelligkeit $<^{(2)}$

$3 < 4$ ist korrekter Ausdruck

Relation $<_{\mathbb{N}} \subseteq \mathbb{N} \times \mathbb{N}$

Analog unterscheidet man Relationen und Prädikate:

Prädikat Zeichen „ $<$ “

Stelligkeit $<^{(2)}$

$3 < 4$ ist korrekter Ausdruck

Relation $<_{\mathbb{N}} \subseteq \mathbb{N} \times \mathbb{N}$

Definition 4.20 (Signatur)

Eine **Signatur** ist ein Paar (F, P) , wobei

- F eine Menge von Funktionssymbolen ist,

Analog unterscheidet man Relationen und Prädikate:

Prädikat Zeichen „ $<$ “

Stelligkeit $<^{(2)}$

$3 < 4$ ist korrekter Ausdruck

Relation $<_{\mathbb{N}} \subseteq \mathbb{N} \times \mathbb{N}$

Definition 4.20 (Signatur)

Eine **Signatur** ist ein Paar (F, P) , wobei

- F eine Menge von Funktionssymbolen ist,
- P eine Menge von Prädikaten ist,

Analog unterscheidet man Relationen und Prädikate:

Prädikat Zeichen „ $<$ “

Stelligkeit $<^{(2)}$

$3 < 4$ ist korrekter Ausdruck

Relation $<_{\mathbb{N}} \subseteq \mathbb{N} \times \mathbb{N}$

Definition 4.20 (Signatur)

Eine **Signatur** ist ein Paar (F, P) , wobei

- F eine Menge von Funktionssymbolen ist,
- P eine Menge von Prädikaten ist,
- jedem Element von $F \cup P$ eine Stelligkeit zugeordnet ist.

Analog unterscheidet man Relationen und Prädikate:

Prädikat Zeichen „ $<$ “

Stelligkeit $<^{(2)}$

$3 < 4$ ist korrekter Ausdruck

Relation $<_{\mathbb{N}} \subseteq \mathbb{N} \times \mathbb{N}$

Definition 4.20 (Signatur)

Eine **Signatur** ist ein Paar (F, P) , wobei

- F eine Menge von Funktionssymbolen ist,
- P eine Menge von Prädikaten ist,
- jedem Element von $F \cup P$ eine Stelligkeit zugeordnet ist.

Analog unterscheidet man Relationen und Prädikate:

Prädikat Zeichen „ $<$ “

Stelligkeit $<^{(2)}$

$3 < 4$ ist korrekter Ausdruck

Relation $<_{\mathbb{N}} \subseteq \mathbb{N} \times \mathbb{N}$

Definition 4.20 (Signatur)

Eine **Signatur** ist ein Paar (F, P) , wobei

- F eine Menge von Funktionssymbolen ist,
- P eine Menge von Prädikaten ist,
- jedem Element von $F \cup P$ eine Stelligkeit zugeordnet ist.

Beispiel 4.21

- $(\{+^{(2)}, \cdot^{(2)}, 0^{(0)}, 1^{(0)}\}, \{\})$ ist die Signatur der Arithmetik

Analog unterscheidet man Relationen und Prädikate:

Prädikat Zeichen „ $<$ “

Stelligkeit $<^{(2)}$

$3 < 4$ ist korrekter Ausdruck

Relation $<_{\mathbb{N}} \subseteq \mathbb{N} \times \mathbb{N}$

Definition 4.20 (Signatur)

Eine **Signatur** ist ein Paar (F, P) , wobei

- F eine Menge von Funktionssymbolen ist,
- P eine Menge von Prädikaten ist,
- jedem Element von $F \cup P$ eine Stelligkeit zugeordnet ist.

Beispiel 4.21

- $(\{+^{(2)}, \cdot^{(2)}, 0^{(0)}, 1^{(0)}\}, \{\})$ ist die Signatur der Arithmetik
- $(\{+^{(2)}, \cdot^{(2)}, 0^{(0)}, 1^{(0)}\}, \{\leq^{(2)}\})$ ist die Signatur der geordneten Arithmetik

Analog unterscheidet man Relationen und Prädikate:

Prädikat Zeichen „ $<$ “

Stelligkeit $<^{(2)}$

$3 < 4$ ist korrekter Ausdruck

Relation $<_{\mathbb{N}} \subseteq \mathbb{N} \times \mathbb{N}$

Definition 4.20 (Signatur)

Eine **Signatur** ist ein Paar (F, P) , wobei

- F eine Menge von Funktionssymbolen ist,
- P eine Menge von Prädikaten ist,
- jedem Element von $F \cup P$ eine Stelligkeit zugeordnet ist.

Beispiel 4.21

- $(\{+^{(2)}, \cdot^{(2)}, 0^{(0)}, 1^{(0)}\}, \{\})$ ist die Signatur der Arithmetik
- $(\{+^{(2)}, \cdot^{(2)}, 0^{(0)}, 1^{(0)}\}, \{\leq^{(2)}\})$ ist die Signatur der geordneten Arithmetik
- $(\{\}, \{E^{(2)}\})$ ist die Signatur der Graphen

Definition 4.22 (Struktur)

Sei $\Sigma = (F, P)$ eine Signatur. Eine Σ -Struktur $\mathcal{A}$ besteht aus

- einer nichtleeren Menge A ,

Definition 4.22 (Struktur)

Sei $\Sigma = (F, P)$ eine Signatur. Eine Σ -Struktur $\mathcal{A}$ besteht aus

- einer nichtleeren Menge A ,
- einer n -stelligen Funktion $f_{\mathcal{A}} : A^n \rightarrow A$ für jedes n -stellige Funktionssymbol $f \in F$,

Definition 4.22 (Struktur)

Sei $\Sigma = (F, P)$ eine Signatur. Eine Σ -Struktur $\mathcal{A}$ besteht aus

- einer nichtleeren Menge A ,
- einer n -stelligen Funktion $f_{\mathcal{A}} : A^n \rightarrow A$ für jedes n -stellige Funktionssymbol $f \in F$,
- einer n -stelligen Relation $S_{\mathcal{A}} \subseteq A^n$ für jedes n -stellige Prädikat $S \in P$.

Definition 4.22 (Struktur)

Sei $\Sigma = (F, P)$ eine Signatur. Eine Σ -Struktur $\mathcal{A}$ besteht aus

- einer nichtleeren Menge A ,
- einer n -stelligen Funktion $f_{\mathcal{A}} : A^n \rightarrow A$ für jedes n -stellige Funktionssymbol $f \in F$,
- einer n -stelligen Relation $S_{\mathcal{A}} \subseteq A^n$ für jedes n -stellige Prädikat $S \in P$.

Definition 4.22 (Struktur)

Sei $\Sigma = (F, P)$ eine Signatur. Eine Σ -Struktur $\mathcal{A}$ besteht aus

- einer nichtleeren Menge A ,
- einer n -stelligen Funktion $f_{\mathcal{A}} : A^n \rightarrow A$ für jedes n -stellige Funktionssymbol $f \in F$,
- einer n -stelligen Relation $S_{\mathcal{A}} \subseteq A^n$ für jedes n -stellige Prädikat $S \in P$.

Beispiel 4.23 (Strukturen mit Signatur der Arithmetik)

- $(\mathbb{N}, +_{\mathbb{N}}, \cdot_{\mathbb{N}}, 0_{\mathbb{N}}, 1_{\mathbb{N}})$

Definition 4.22 (Struktur)

Sei $\Sigma = (F, P)$ eine Signatur. Eine Σ -Struktur $\mathcal{A}$ besteht aus

- einer nichtleeren Menge A ,
- einer n -stelligen Funktion $f_{\mathcal{A}} : A^n \rightarrow A$ für jedes n -stellige Funktionssymbol $f \in F$,
- einer n -stelligen Relation $S_{\mathcal{A}} \subseteq A^n$ für jedes n -stellige Prädikat $S \in P$.

Beispiel 4.23 (Strukturen mit Signatur der Arithmetik)

- $(\mathbb{N}, +_{\mathbb{N}}, \cdot_{\mathbb{N}}, 0_{\mathbb{N}}, 1_{\mathbb{N}})$
- $(\mathbb{N}^{2 \times 2}, +_M, \cdot_M, \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix})$,
wobei $+_M$ und $\cdot_M$ als Matrizen-Addition und -Multiplikation definiert sind

Definition 4.22 (Struktur)

Sei $\Sigma = (F, P)$ eine Signatur. Eine Σ -Struktur $\mathcal{A}$ besteht aus

- einer nichtleeren Menge A ,
- einer n -stelligen Funktion $f_{\mathcal{A}} : A^n \rightarrow A$ für jedes n -stellige Funktionssymbol $f \in F$,
- einer n -stelligen Relation $S_{\mathcal{A}} \subseteq A^n$ für jedes n -stellige Prädikat $S \in P$.

Beispiel 4.23 (Strukturen mit Signatur der Arithmetik)

- $(\mathbb{N}, +_{\mathbb{N}}, \cdot_{\mathbb{N}}, 0_{\mathbb{N}}, 1_{\mathbb{N}})$
- $(\mathbb{N}^{2 \times 2}, +_M, \cdot_M, \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix})$,
wobei $+_M$ und $\cdot_M$ als Matrizen-Addition und -Multiplikation definiert sind
- $(\mathbb{B}, +_{\mathbb{B}}, \cdot_{\mathbb{B}}, 0_{\mathbb{B}}, 1_{\mathbb{B}})$ mit $\mathbb{B} = \{0_{\mathbb{B}}, 1_{\mathbb{B}}\}$ und

$+$	0	1
0	0	1
1	1	1

,

$\cdot$	0	1
0	0	0
1	0	1

- Das Gebiet der **Algebra** beschäftigt sich mit den Eigenschaften von **Rechenoperationen**

- Das Gebiet der **Algebra** beschäftigt sich mit den Eigenschaften von **Rechenoperationen**
- Eine **Algebraische Struktur** (kurz: Algebra) hat eine **funktionale** Signatur.

- Das Gebiet der **Algebra** beschäftigt sich mit den Eigenschaften von **Rechenoperationen**
- Eine **Algebraische Struktur** (kurz: Algebra) hat eine **funktionale** Signatur.

Beispiel 4.24 (Bekannte algebraische Strukturen)

- Das Gebiet der **Algebra** beschäftigt sich mit den Eigenschaften von **Rechenoperationen**
- Eine **Algebraische Struktur** (kurz: Algebra) hat eine **funktionale** Signatur.

Beispiel 4.24 (Bekannte algebraische Strukturen)

- $(\mathbb{Z}, +_{\mathbb{Z}}^{(2)}, 0_{\mathbb{Z}}^{(0)}, -_{\mathbb{Z}}^{(1)})$ ist eine **Gruppe**

- Das Gebiet der **Algebra** beschäftigt sich mit den Eigenschaften von **Rechenoperationen**
- Eine **Algebraische Struktur** (kurz: Algebra) hat eine **funktionale** Signatur.

Beispiel 4.24 (Bekannte algebraische Strukturen)

- $(\mathbb{Z}, +_{\mathbb{Z}}^{(2)}, 0_{\mathbb{Z}}^{(0)}, -_{\mathbb{Z}}^{(1)})$ ist eine **Gruppe**
- $(\mathbb{Z}, +_{\mathbb{Z}}^{(2)}, 0_{\mathbb{Z}}^{(0)}, -_{\mathbb{Z}}^{(1)}, \cdot_{\mathbb{Z}}^{(2)})$ ist ein **Ring**

Relation Menge von Tupeln

Relation Menge von Tupeln

Funktion Abbildung von Tupeln auf Elemente

Relation Menge von Tupeln

Funktion Abbildung von Tupeln auf Elemente

Signatur Funktionssymbole und Prädikate mit fester Stelligkeit

Relation Menge von Tupeln

Funktion Abbildung von Tupeln auf Elemente

Signatur Funktionssymbole und Prädikate mit fester Stelligkeit

Struktur Menge mit entsprechenden Funktionen und Relationen

- 1. Einführung
- 2. Mengen
- 3. Aussagenlogik
- 4. Prädikatenlogik**
 - 4.1 Relationen und Funktionen
 - 4.2 Syntax**
 - 4.3 Semantik
 - 4.4 Schlussfolgerungsverfahren
- 5. Prolog

Konstruktor	Notation	reale Welt	Mathematik
Variablen	x, y	(Menschen)	(Zahlen)
Konstantensymbole	a, b, c	calvin, hobbes, mom, dad	1, 2, π

Konstruktor	Notation	reale Welt	Mathematik
Variablen	x, y	(Menschen)	(Zahlen)
Konstantensymbole	a, b, c	calvin, hobbes, mom, dad	$1, 2, \pi$
Funktionssymbole	$f(x), h(x, y)$	mutter(calvin), ehepartner(mom)	$x + y, \sqrt{x},$ $\log_b x$
Prädikate (Relationssymbole)	$R(x), S(x, y)$	Sohn(mom,calvin), Freund(calvin,hobbes)	$x > y, x = y,$ $\text{Prim}(x)$

Konstruktor	Notation	reale Welt	Mathematik
Variablen	x, y	(Menschen)	(Zahlen)
Konstantensymbole	a, b, c	calvin, hobbes, mom, dad	$1, 2, \pi$
Funktionssymbole	$f(x), h(x, y)$	mutter(calvin), ehepartner(mom)	$x + y, \sqrt{x},$ $\log_b x$
Prädikate (Relationssymbole)	$R(x), S(x, y)$	Sohn(mom,calvin), Freund(calvin,hobbes)	$x > y, x = y,$ $\text{Prim}(x)$
Quantoren	$\forall x P(x),$ $\exists y S(x, y)$	$\forall x (\text{Freund}(x, \text{hobbes}))$ $\exists x (\text{Freund}(\text{calvin}, x))$	$\forall x (x \geq 0)$ $\exists x (x + x = 2)$

Konstruktor	Notation	reale Welt	Mathematik
Variablen	x, y	(Menschen)	(Zahlen)
Konstantensymbole	a, b, c	calvin, hobbes, mom, dad	$1, 2, \pi$
Funktionssymbole	$f(x), h(x, y)$	mutter(calvin), ehepartner(mom)	$x + y, \sqrt{x},$ $\log_b x$
Prädikate (Relationssymbole)	$R(x), S(x, y)$	Sohn(mom,calvin), Freund(calvin,hobbes)	$x > y, x = y,$ $\text{Prim}(x)$
Quantoren	$\forall x P(x),$ $\exists y S(x, y)$	$\forall x (\text{Freund}(x, \text{hobbes}))$ $\exists x (\text{Freund}(\text{calvin}, x))$	$\forall x (x \geq 0)$ $\exists x (x + x = 2)$
Junktoren	$\wedge, \vee, \neg, \rightarrow, \leftrightarrow$		

- Ein n -stelliges Funktionssymbol hat genau n Argumente

- Ein n -stelliges Funktionssymbol hat genau n Argumente
- $+$ und $\log$ sind **zweistellig** (binär) ($x + y, \log_b x$)

- Ein n -stelliges Funktionssymbol hat genau n Argumente
- $+$ und $\log$ sind **zweistellig** (binär) ($x + y, \log_b x$)
- $\sqrt{}$ ist **einstellig** (unär) ($\sqrt{x}$)

- Ein n -stelliges Funktionssymbol hat genau n Argumente
- $+$ und $\log$ sind **zweistellig** (binär) ($x + y, \log_b x$)
- $\sqrt{}$ ist **einstellig** (unär) ($\sqrt{x}$)
- `mutter()` und `ehepartner()` sind einstellig

- Ein n -stelliges Funktionssymbol hat genau n Argumente
- $+$ und $\log$ sind **zweistellig** (binär) ($x + y, \log_b x$)
- $\sqrt{}$ ist **einstellig** (unär) ($\sqrt{x}$)
- `mutter()` und `ehepartner()` sind einstellig
- **nullstellige** Funktionssymbole sind Konstantensymbole

- Ein n -stelliges Funktionssymbol hat genau n Argumente
- $+$ und $\log$ sind **zweistellig** (binär) ($x + y, \log_b x$)
- $\sqrt{}$ ist **einstellig** (unär) ($\sqrt{x}$)
- `mutter()` und `ehepartner()` sind einstellig
- **nullstellige** Funktionssymbole sind Konstantensymbole
- Notation (falls notwendig): $f^{(n)}$

Aus Variablen und Funktionssymbolen (einschließlich Konstantensymbolen) sind Terme zusammengesetzt.

Aus Variablen und Funktionssymbolen (einschließlich Konstantensymbolen) sind Terme zusammengesetzt.

Definition 4.25 (Term)

Terme sind wie folgt induktiv definiert:

- Jede Variable ist ein Term.

Aus Variablen und Funktionssymbolen (einschließlich Konstantensymbolen) sind Terme zusammengesetzt.

Definition 4.25 (Term)

Terme sind wie folgt induktiv definiert:

- Jede Variable ist ein Term.
- Wenn f ein n -stelliges Funktionssymbol ist und $t_1, \dots, t_n$ Terme sind, ist auch $f(t_1, \dots, t_n)$ ein Term.

Aus Variablen und Funktionssymbolen (einschließlich Konstantensymbolen) sind Terme zusammengesetzt.

Definition 4.25 (Term)

Terme sind wie folgt induktiv definiert:

- Jede Variable ist ein Term.
- Wenn f ein n -stelliges Funktionssymbol ist und $t_1, \dots, t_n$ Terme sind, ist auch $f(t_1, \dots, t_n)$ ein Term.

Aus Variablen und Funktionssymbolen (einschließlich Konstantensymbolen) sind Terme zusammengesetzt.

Definition 4.25 (Term)

Terme sind wie folgt induktiv definiert:

- Jede Variable ist ein Term.
- Wenn f ein n -stelliges Funktionssymbol ist und $t_1, \dots, t_n$ Terme sind, ist auch $f(t_1, \dots, t_n)$ ein Term.

Terme, die keine Variablen enthalten, heißen auch Grundterme.

Aus Variablen und Funktionssymbolen (einschließlich Konstantensymbolen) sind Terme zusammengesetzt.

Definition 4.25 (Term)

Terme sind wie folgt induktiv definiert:

- Jede Variable ist ein Term.
- Wenn f ein n -stelliges Funktionssymbol ist und $t_1, \dots, t_n$ Terme sind, ist auch $f(t_1, \dots, t_n)$ ein Term.

Terme, die keine Variablen enthalten, heißen auch Grundterme.

Beachte:

- Jede Konstante ist ein Term.
- Terme bezeichnen Elemente der Domäne.

- Variablen: $\{x, y\}$
- Funktionssymbole: $\{c^{(0)}, d^{(0)}, f^{(1)}, +^{(2)}\}$

- Variablen: $\{x, y\}$
- Funktionssymbole: $\{c^{(0)}, d^{(0)}, f^{(1)}, +^{(2)}\}$

Beispiel 4.26 (Grundterme)

■ c

- Variablen: $\{x, y\}$
- Funktionssymbole: $\{c^{(0)}, d^{(0)}, f^{(1)}, +^{(2)}\}$

Beispiel 4.26 (Grundterme)

- c
- $f(f(d))$

- Variablen: $\{x, y\}$
- Funktionssymbole: $\{c^{(0)}, d^{(0)}, f^{(1)}, +^{(2)}\}$

Beispiel 4.26 (Grundterme)

- c
- $f(f(d))$
- $c + f(c)$ oder $+(c, f(c))$

- Variablen: $\{x, y\}$
- Funktionssymbole: $\{c^{(0)}, d^{(0)}, f^{(1)}, +^{(2)}\}$

Beispiel 4.26 (Grundterme)

- c
- $f(f(d))$
- $c + f(c)$ oder $+(c, f(c))$
- $+(c, +(d, +(f(c), f(d))))$

- Variablen: $\{x, y\}$
- Funktionssymbole: $\{c^{(0)}, d^{(0)}, f^{(1)}, +^{(2)}\}$

Beispiel 4.26 (Grundterme)

- c
- $f(f(d))$
- $c + f(c)$ oder $+(c, f(c))$
- $+(c, +(d, +(f(c), f(d))))$

- Variablen: $\{x, y\}$
- Funktionssymbole: $\{c^{(0)}, d^{(0)}, f^{(1)}, +^{(2)}\}$

Beispiel 4.26 (Grundterme)

- c
- $f(f(d))$
- $c + f(c)$ oder $+(c, f(c))$
- $+(c, +(d, +(f(c), f(d))))$

Beispiel 4.27 (Terme, die keine Grundterme sind)

- x

- Variablen: $\{x, y\}$
- Funktionssymbole: $\{c^{(0)}, d^{(0)}, f^{(1)}, +^{(2)}\}$

Beispiel 4.26 (Grundterme)

- c
- $f(f(d))$
- $c + f(c)$ oder $+(c, f(c))$
- $+(c, +(d, +(f(c), f(d))))$

Beispiel 4.27 (Terme, die keine Grundterme sind)

- x
- $f(y)$

- Variablen: $\{x, y\}$
- Funktionssymbole: $\{c^{(0)}, d^{(0)}, f^{(1)}, +^{(2)}\}$

Beispiel 4.26 (Grundterme)

- c
- $f(f(d))$
- $c + f(c)$ oder $+(c, f(c))$
- $+(c, +(d, +(f(c), f(d))))$

Beispiel 4.27 (Terme, die keine Grundterme sind)

- x
- $f(y)$
- $c + f(y)$

- Variablen: $\{x, y\}$
- Funktionssymbole: $\{c^{(0)}, d^{(0)}, f^{(1)}, +^{(2)}\}$

Beispiel 4.26 (Grundterme)

- c
- $f(f(d))$
- $c + f(c)$ oder $+(c, f(c))$
- $+(c, +(d, +(f(c), f(d))))$

Beispiel 4.27 (Terme, die keine Grundterme sind)

- x
- $f(y)$
- $c + f(y)$
- $+(c, +(y, +(f(x), f(y))))$

- Ein n -stelliges Prädikat hat genau n Terme als Argumente.

- Ein n -stelliges Prädikat hat genau n Terme als Argumente.
 - Keine Prädikate in Prädikaten!

- Ein n -stelliges Prädikat hat genau n Terme als Argumente.
 - Keine Prädikate in Prädikaten!
- $>$ ist zweistellig ($x > y$)

- Ein n -stelliges Prädikat hat genau n Terme als Argumente.
 - Keine Prädikate in Prädikaten!
- $>$ ist zweistellig ($x > y$)
- $\text{Prim}()$ ist einstellig ($\text{Prim}(x)$).

- Ein n -stelliges Prädikat hat genau n Terme als Argumente.
 - Keine Prädikate in Prädikaten!
- $>$ ist zweistellig ($x > y$)
- $\text{Prim}()$ ist einstellig ($\text{Prim}(x)$).
- $\text{Sohn}()$ und $\text{Freund}()$ sind zweistellig.

Aus Prädikaten und Termen sind Atome zusammengesetzt.

Aus Prädikaten und Termen sind Atome zusammengesetzt.

Definition 4.28 (Atom)

Atome sind wie folgt definiert:

- Wenn R ein n -stelliges Prädikat ist und $t_1, \dots, t_n$ Terme sind, ist $R(t_1, \dots, t_n)$ ein Atom.

Aus Prädikaten und Termen sind Atome zusammengesetzt.

Definition 4.28 (Atom)

Atome sind wie folgt definiert:

- Wenn R ein n -stelliges Prädikat ist und $t_1, \dots, t_n$ Terme sind, ist $R(t_1, \dots, t_n)$ ein Atom.

Aus Prädikaten und Termen sind Atome zusammengesetzt.

Definition 4.28 (Atom)

Atome sind wie folgt definiert:

- Wenn R ein n -stelliges Prädikat ist und $t_1, \dots, t_n$ Terme sind, ist $R(t_1, \dots, t_n)$ ein Atom.
- Atome, die keine Variablen enthalten, heißen auch Grundatome.

Aus Prädikaten und Termen sind Atome zusammengesetzt.

Definition 4.28 (Atom)

Atome sind wie folgt definiert:

- Wenn R ein n -stelliges Prädikat ist und $t_1, \dots, t_n$ Terme sind, ist $R(t_1, \dots, t_n)$ ein Atom.
Atome, die keine Variablen enthalten, heißen auch **Grundatome**.
- Jedes nullstellige Prädikat ist ein Atom.
 - Nullstellige Prädikate sind **Aussagenvariablen**.
- Atome repräsentieren **Elementaraussagen** (sind wahr oder falsch).

- Prädikate: $\{A^{(0)}, P^{(1)}, >^{(2)}\}$

Beispiel 4.29 (Grundatome)

- $P(c)$

- Prädikate: $\{A^{(0)}, P^{(1)}, >^{(2)}\}$

Beispiel 4.29 (Grundatome)

- $P(c)$
- $c > f(c)$ oder $>(c, f(c))$

- Prädikate: $\{A^{(0)}, P^{(1)}, >^{(2)}\}$

Beispiel 4.29 (Grundatome)

- $P(c)$
- $c > f(c)$ oder $>(c, f(c))$
- $c > c$

- Prädikate: $\{A^{(0)}, P^{(1)}, >^{(2)}\}$

Beispiel 4.29 (Grundatome)

- $P(c)$
- $c > f(c)$ oder $>(c, f(c))$
- $c > c$
- $P(c + f(c))$

- Prädikate: $\{A^{(0)}, P^{(1)}, >^{(2)}\}$

Beispiel 4.29 (Grundatome)

- $P(c)$
- $c > f(c)$ oder $>(c, f(c))$
- $c > c$
- $P(c + f(c))$
- A

- Prädikate: $\{A^{(0)}, P^{(1)}, >^{(2)}\}$

Beispiel 4.29 (Grundatome)

- $P(c)$
- $c > f(c)$ oder $>(c, f(c))$
- $c > c$
- $P(c + f(c))$
- A

- Prädikate: $\{A^{(0)}, P^{(1)}, >^{(2)}\}$

Beispiel 4.29 (Grundatome)

- $P(c)$
- $c > f(c)$ oder $>(c, f(c))$
- $c > c$
- $P(c + f(c))$
- A

Beispiel 4.30 (Atome, die keine Grundatome sind)

- $x > y$

- Prädikate: $\{A^{(0)}, P^{(1)}, >^{(2)}\}$

Beispiel 4.29 (Grundatome)

- $P(c)$
- $c > f(c)$ oder $>(c, f(c))$
- $c > c$
- $P(c + f(c))$
- A

Beispiel 4.30 (Atome, die keine Grundatome sind)

- $x > y$
- $f(y) > c + f(x)$

- Prädikate: $\{A^{(0)}, P^{(1)}, >^{(2)}\}$

Beispiel 4.29 (Grundatome)

- $P(c)$
- $c > f(c)$ oder $>(c, f(c))$
- $c > c$
- $P(c + f(c))$
- A

Beispiel 4.30 (Atome, die keine Grundatome sind)

- $x > y$
- $f(y) > c + f(x)$
- $P(f(c + y))$

Aus Atomen, Quantoren und Junktoren sind (komplexe) Formeln zusammengesetzt.

Definition 4.31 (PrädikatenLogische Formel)

Formeln sind wie folgt induktiv definiert:

- Jedes Atom ist eine Formel.

Aus Atomen, Quantoren und Junktoren sind (komplexe) Formeln zusammengesetzt.

Definition 4.31 (PrädikatenLogische Formel)

Formeln sind wie folgt induktiv definiert:

- Jedes Atom ist eine Formel.
- Wenn φ und ψ Formeln sind und x eine Variable ist, sind auch die folgenden Ausdrücke Formeln:

Aus Atomen, Quantoren und Junktoren sind (komplexe) Formeln zusammengesetzt.

Definition 4.31 (PrädikatenLogische Formel)

Formeln sind wie folgt induktiv definiert:

- Jedes Atom ist eine Formel.
- Wenn φ und ψ Formeln sind und x eine Variable ist, sind auch die folgenden Ausdrücke Formeln:
 - $(\neg\varphi)$, **W**, **F**

Aus Atomen, Quantoren und Junktoren sind (komplexe) Formeln zusammengesetzt.

Definition 4.31 (PrädikatenLogische Formel)

Formeln sind wie folgt induktiv definiert:

- Jedes Atom ist eine Formel.
- Wenn φ und ψ Formeln sind und x eine Variable ist, sind auch die folgenden Ausdrücke Formeln:
 - $(\neg\varphi)$, **W**, **F**
 - $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\varphi \rightarrow \psi)$, $(\varphi \leftrightarrow \psi)$

Aus Atomen, Quantoren und Junktoren sind (komplexe) Formeln zusammengesetzt.

Definition 4.31 (Prädikatenlogische Formel)

Formeln sind wie folgt induktiv definiert:

- Jedes Atom ist eine Formel.
- Wenn φ und ψ Formeln sind und x eine Variable ist, sind auch die folgenden Ausdrücke Formeln:
 - $(\neg\varphi)$, **W**, **F**
 - $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\varphi \rightarrow \psi)$, $(\varphi \leftrightarrow \psi)$
 - $(\forall x\varphi)$, $(\exists x\varphi)$

- Allquantor $\forall$: $\forall x\varphi$
- Existenzquantor $\exists$: $\exists x\varphi$

Die Formel φ steht im **Gültigkeitsbereich** (Scope) des Quantors.
Der Quantor **bindet** die freien Vorkommen von x in φ .

- Allquantor $\forall$: $\forall x\varphi$
- Existenzquantor $\exists$: $\exists x\varphi$

Die Formel φ steht im **Gültigkeitsbereich** (Scope) des Quantors.
Der Quantor **bindet** die freien Vorkommen von x in φ .

Definition 4.32 (frei, gebunden)

- Eine Variable, die in einer quantorenfreien Formel φ vorkommt, ist **frei**.

- Allquantor $\forall$: $\forall x\varphi$
- Existenzquantor $\exists$: $\exists x\varphi$

Die Formel φ steht im **Gültigkeitsbereich** (Scope) des Quantors.
Der Quantor **bindet** die freien Vorkommen von x in φ .

Definition 4.32 (frei, gebunden)

- Eine Variable, die in einer quantorenfreien Formel φ vorkommt, ist **frei**.
- In einer Formel $(\forall x\varphi)$ oder $(\exists x\varphi)$ **bindet** der Quantor die freien Vorkommen von x in φ (so dass diese Vorkommen nicht mehr frei sind).

- Allquantor $\forall$: $\forall x\varphi$
- Existenzquantor $\exists$: $\exists x\varphi$

Die Formel φ steht im **Gültigkeitsbereich** (Scope) des Quantors.
Der Quantor **bindet** die freien Vorkommen von x in φ .

Definition 4.32 (frei, gebunden)

- Eine Variable, die in einer quantorenfreien Formel φ vorkommt, ist **frei**.
- In einer Formel $(\forall x\varphi)$ oder $(\exists x\varphi)$ **bindet** der Quantor die freien Vorkommen von x in φ (so dass diese Vorkommen nicht mehr frei sind).
- Eine Formel mit freien Variablen heißt **offen**.

- Allquantor $\forall$: $\forall x\varphi$
- Existenzquantor $\exists$: $\exists x\varphi$

Die Formel φ steht im **Gültigkeitsbereich** (Scope) des Quantors.
Der Quantor **bindet** die freien Vorkommen von x in φ .

Definition 4.32 (frei, gebunden)

- Eine Variable, die in einer quantorenfreien Formel φ vorkommt, ist **frei**.
- In einer Formel $(\forall x\varphi)$ oder $(\exists x\varphi)$ **bindet** der Quantor die freien Vorkommen von x in φ (so dass diese Vorkommen nicht mehr frei sind).
- Eine Formel mit freien Variablen heißt **offen**.
- Eine Formel ohne freie Variablen heißt **geschlossen** (oder auch **Satz**).

- Allquantor $\forall$: $\forall x\varphi$
- Existenzquantor $\exists$: $\exists x\varphi$

Die Formel φ steht im **Gültigkeitsbereich** (Scope) des Quantors.
Der Quantor **bindet** die freien Vorkommen von x in φ .

Definition 4.32 (frei, gebunden)

- Eine Variable, die in einer quantorenfreien Formel φ vorkommt, ist **frei**.
- In einer Formel $(\forall x\varphi)$ oder $(\exists x\varphi)$ **bindet** der Quantor die freien Vorkommen von x in φ (so dass diese Vorkommen nicht mehr frei sind).
- Eine Formel mit freien Variablen heißt **offen**.
- Eine Formel ohne freie Variablen heißt **geschlossen** (oder auch **Satz**).

- Allquantor $\forall$: $\forall x\varphi$
- Existenzquantor $\exists$: $\exists x\varphi$

Die Formel φ steht im **Gültigkeitsbereich** (Scope) des Quantors.
Der Quantor **bindet** die freien Vorkommen von x in φ .

Definition 4.32 (frei, gebunden)

- Eine Variable, die in einer quantorenfreien Formel φ vorkommt, ist **frei**.
 - In einer Formel $(\forall x\varphi)$ oder $(\exists x\varphi)$ **bindet** der Quantor die freien Vorkommen von x in φ (so dass diese Vorkommen nicht mehr frei sind).
 - Eine Formel mit freien Variablen heißt **offen**.
 - Eine Formel ohne freie Variablen heißt **geschlossen** (oder auch **Satz**).
-
- Quantor bindet nur **Vorkommen** einer Variable, nicht Variable selbst.
 - In $P(x) \wedge \forall xQ(x)$ kommt x sowohl frei als auch gebunden vor.

$f \gg P \gg \forall, \exists \gg \neg \gg \wedge \gg \vee \gg \rightarrow \gg \leftrightarrow$

$f \gg P \gg \forall, \exists \gg \neg \gg \wedge \gg \vee \gg \rightarrow \gg \leftrightarrow$

- Funktionssymbole binden stärker als Prädikate

$f \gg P \gg \forall, \exists \gg \neg \gg \wedge \gg \vee \gg \rightarrow \gg \leftrightarrow$

- Funktionssymbole binden stärker als Prädikate
 - $x + y > c + d$ bedeutet $(x + y) > (c + d)$

$f \gg P \gg \forall, \exists \gg \neg \gg \wedge \gg \vee \gg \rightarrow \gg \leftrightarrow$

- Funktionssymbole binden stärker als Prädikate
 - $x + y > c + d$ bedeutet $(x + y) > (c + d)$
- Prädikate binden stärker als Quantoren

$f \gg P \gg \forall, \exists \gg \neg \gg \wedge \gg \vee \gg \rightarrow \gg \leftrightarrow$

- Funktionssymbole binden stärker als Prädikate
 - $x + y > c + d$ bedeutet $(x + y) > (c + d)$
- Prädikate binden stärker als Quantoren
 - $\forall x \forall y x + y = y + x$ bedeutet $\forall x \forall y (x + y = y + x)$

$f \gg P \gg \forall, \exists \gg \neg \gg \wedge \gg \vee \gg \rightarrow \gg \leftrightarrow$

- Funktionssymbole binden stärker als Prädikate
 - $x + y > c + d$ bedeutet $(x + y) > (c + d)$
- Prädikate binden stärker als Quantoren
 - $\forall x \forall y x + y = y + x$ bedeutet $\forall x \forall y (x + y = y + x)$
 - Sonst wäre es auch keine Formel: $(\forall x \forall y x) + y = y + x$

f $\gg$ P $\gg$ $\forall, \exists$ $\gg$ $\neg$ $\gg$ $\wedge$ $\gg$ $\vee$ $\gg$ $\rightarrow$ $\gg$ $\leftrightarrow$

- Funktionssymbole binden stärker als Prädikate
 - $x + y > c + d$ bedeutet $(x + y) > (c + d)$
- Prädikate binden stärker als Quantoren
 - $\forall x \forall y x + y = y + x$ bedeutet $\forall x \forall y (x + y = y + x)$
 - Sonst wäre es auch keine Formel: $(\forall x \forall y x) + y = y + x$
 - Klarer in Präfix-Notation: $\forall x \forall y = (+ (x, y), + (y, x))$

f $\gg$ P $\gg$ $\forall, \exists$ $\gg$ $\neg$ $\gg$ $\wedge$ $\gg$ $\vee$ $\gg$ $\rightarrow$ $\gg$ $\leftrightarrow$

- Funktionssymbole binden stärker als Prädikate
 - $x + y > c + d$ bedeutet $(x + y) > (c + d)$
- Prädikate binden stärker als Quantoren
 - $\forall x \forall y x + y = y + x$ bedeutet $\forall x \forall y (x + y = y + x)$
 - Sonst wäre es auch keine Formel: $(\forall x \forall y x) + y = y + x$
 - Klarer in Präfix-Notation: $\forall x \forall y = (+ (x, y), + (y, x))$
- Quantoren binden stärker als Junktoren

f $\gg$ P $\gg$ $\forall, \exists$ $\gg$ $\neg$ $\gg$ $\wedge$ $\gg$ $\vee$ $\gg$ $\rightarrow$ $\gg$ $\leftrightarrow$

- Funktionssymbole binden stärker als Prädikate
 - $x + y > c + d$ bedeutet $(x + y) > (c + d)$
- Prädikate binden stärker als Quantoren
 - $\forall x \forall y x + y = y + x$ bedeutet $\forall x \forall y (x + y = y + x)$
 - Sonst wäre es auch keine Formel: $(\forall x \forall y x) + y = y + x$
 - Klarer in Präfix-Notation: $\forall x \forall y = (+ (x, y), + (y, x))$
- Quantoren binden stärker als Junktoren
 - $\forall x R(x, y) \wedge S(x)$ bedeutet $(\forall x R(x, y)) \wedge S(x)$

$f \gg P \gg \forall, \exists \gg \neg \gg \wedge \gg \vee \gg \rightarrow \gg \leftrightarrow$

- Funktionssymbole binden stärker als Prädikate
 - $x + y > c + d$ bedeutet $(x + y) > (c + d)$
- Prädikate binden stärker als Quantoren
 - $\forall x \forall y x + y = y + x$ bedeutet $\forall x \forall y (x + y = y + x)$
 - Sonst wäre es auch keine Formel: $(\forall x \forall y x) + y = y + x$
 - Klarer in Präfix-Notation: $\forall x \forall y = (+ (x, y), + (y, x))$
- Quantoren binden stärker als Junktoren
 - $\forall x R(x, y) \wedge S(x)$ bedeutet $(\forall x R(x, y)) \wedge S(x)$
 - $\exists y R(c, y) \rightarrow S(y)$ bedeutet $(\exists y R(c, y)) \rightarrow S(y)$

$f \gg P \gg \forall, \exists \gg \neg \gg \wedge \gg \vee \gg \rightarrow \gg \leftrightarrow$

- Funktionssymbole binden stärker als Prädikate
 - $x + y > c + d$ bedeutet $(x + y) > (c + d)$
- Prädikate binden stärker als Quantoren
 - $\forall x \forall y x + y = y + x$ bedeutet $\forall x \forall y (x + y = y + x)$
 - Sonst wäre es auch keine Formel: $(\forall x \forall y x) + y = y + x$
 - Klarer in Präfix-Notation: $\forall x \forall y = (+ (x, y), + (y, x))$
- Quantoren binden stärker als Junktoren
 - $\forall x R(x, y) \wedge S(x)$ bedeutet $(\forall x R(x, y)) \wedge S(x)$
 - $\exists y R(c, y) \rightarrow S(y)$ bedeutet $(\exists y R(c, y)) \rightarrow S(y)$
 - anschaulich: Quantor bindet Variablen bis zum nächsten Junktor (oder bis zur schließenden Klammer)

$f \gg P \gg \forall, \exists \gg \neg \gg \wedge \gg \vee \gg \rightarrow \gg \leftrightarrow$

- Funktionssymbole binden stärker als Prädikate
 - $x + y > c + d$ bedeutet $(x + y) > (c + d)$
- Prädikate binden stärker als Quantoren
 - $\forall x \forall y x + y = y + x$ bedeutet $\forall x \forall y (x + y = y + x)$
 - Sonst wäre es auch keine Formel: $(\forall x \forall y x) + y = y + x$
 - Klarer in Präfix-Notation: $\forall x \forall y = (+ (x, y), + (y, x))$
- Quantoren binden stärker als Junktoren
 - $\forall x R(x, y) \wedge S(x)$ bedeutet $(\forall x R(x, y)) \wedge S(x)$
 - $\exists y R(c, y) \rightarrow S(y)$ bedeutet $(\exists y R(c, y)) \rightarrow S(y)$
 - anschaulich: Quantor bindet Variablen bis zum nächsten Junktor (oder bis zur schließenden Klammer)
- Klammern können weggelassen werden, wenn sie den Vorrang nicht verändern

Beispiel 4.33 (Offene Formeln)

■ $P(x)$

Beispiel 4.33 (Offene Formeln)

- $P(x)$
- $f(y) > c + f(x)$

Beispiel 4.33 (Offene Formeln)

- $P(x)$
- $f(y) > c + f(x)$
- $P(y) \vee x > c$

Beispiel 4.33 (Offene Formeln)

- $P(x)$
- $f(y) > c + f(x)$
- $P(y) \vee x > c$
- $x > y \rightarrow \neg \forall x (x > y)$

Beispiel 4.33 (Offene Formeln)

- $P(x)$
- $f(y) > c + f(x)$
- $P(y) \vee x > c$
- $x > y \rightarrow \neg \forall x(x > y)$
- $\forall x P(x) \vee x > c \quad (!)$

Beispiel 4.33 (Offene Formeln)

- $P(x)$
- $f(y) > c + f(x)$
- $P(y) \vee x > c$
- $x > y \rightarrow \neg \forall x(x > y)$
- $\forall x P(x) \vee x > c \quad (!)$

Beispiel 4.33 (Offene Formeln)

- $P(x)$
- $f(y) > c + f(x)$
- $P(y) \vee x > c$
- $x > y \rightarrow \neg \forall x(x > y)$
- $\forall x P(x) \vee x > c \quad (!)$

Beispiel 4.34 (Sätze)

- $\forall x x > c$

Beispiel 4.33 (Offene Formeln)

- $P(x)$
- $f(y) > c + f(x)$
- $P(y) \vee x > c$
- $x > y \rightarrow \neg \forall x(x > y)$
- $\forall x P(x) \vee x > c \quad (!)$

Beispiel 4.34 (Sätze)

- $\forall x x > c$
- $\forall x \forall y (x > y \vee y > x \vee x = y)$

Beispiel 4.33 (Offene Formeln)

- $P(x)$
- $f(y) > c + f(x)$
- $P(y) \vee x > c$
- $x > y \rightarrow \neg \forall x(x > y)$
- $\forall x P(x) \vee x > c$ (!)

Beispiel 4.34 (Sätze)

- $\forall x x > c$
- $\forall x \forall y (x > y \vee y > x \vee x = y)$
- $\exists x P(x)$

Beispiel 4.33 (Offene Formeln)

- $P(x)$
- $f(y) > c + f(x)$
- $P(y) \vee x > c$
- $x > y \rightarrow \neg \forall x(x > y)$
- $\forall x P(x) \vee x > c$ (!)

Beispiel 4.34 (Sätze)

- $\forall x x > c$
- $\forall x \forall y (x > y \vee y > x \vee x = y)$
- $\exists x P(x)$
- $\exists x (P(x) \vee x > d)$

Beispiel 4.33 (Offene Formeln)

- $P(x)$
- $f(y) > c + f(x)$
- $P(y) \vee x > c$
- $x > y \rightarrow \neg \forall x(x > y)$
- $\forall x P(x) \vee x > c$ (!)

Beispiel 4.34 (Sätze)

- $\forall x x > c$
- $\forall x \forall y (x > y \vee y > x \vee x = y)$
- $\exists x P(x)$
- $\exists x (P(x) \vee x > d)$
- $\exists x (P(x) \rightarrow \forall y (y + c > x))$

Übung 4.35

Gegeben sei die Signatur $(\{+^{(2)}, f^{(1)}, c^{(0)}, d^{(0)}\}, \{=^{(2)}, >^{(2)}, P^{(1)}\})$.

Entscheiden Sie, in welche Kategorien die folgenden Ausdrücke fallen:

- (Grund-) Term
- offene Formel
- Unsinn
- (Grund-) Atom
- Satz

Entscheiden Sie, ob die Variablenvorkommen gebunden oder frei sind.

1 $x + y$	9 $P(\forall x(x = x))$
2 $x + y > c$	10 $x > y \wedge x + y$
3 $P(c, d)$	11 $\forall x(x > c \vee c > x) \vee x = c$
4 $x > P(d)$	12 $c > f(d)$
5 $\forall x(x + c > x)$	13 $P(P(x))$
6 $\forall x > (+ (x, c), x)$	14 $f(f(x))$
7 $\forall x, y(x > y)$	15 $\forall x \forall y(P(x) > P(y))$
8 $\exists x \exists y(x + y)$	16 $\exists x P(x) \rightarrow \forall x P(x)$

- 1. Einführung
- 2. Mengen
- 3. Aussagenlogik
- 4. Prädikatenlogik**
 - 4.1 Relationen und Funktionen
 - 4.2 Syntax
 - 4.3 Semantik**
 - 4.4 Schlussfolgerungsverfahren
- 5. Prolog

Interpretation in der Aussagenlogik

- weist Aussagenvariablen Wert 1 oder 0 zu
- macht Formel wahr oder falsch

Interpretation in der Aussagenlogik

- weist Aussagenvariablen Wert 1 oder 0 zu
- macht Formel wahr oder falsch

Interpretation in der Prädikatenlogik

- bestimmt das **Universum** (auch **Domäne** genannt)

Interpretation in der Aussagenlogik

- weist Aussagenvariablen Wert 1 oder 0 zu
- macht Formel wahr oder falsch

Interpretation in der Prädikatenlogik

- bestimmt das **Universum** (auch **Domäne** genannt)
- weist jedem n -stelligen **Funktionssymbol** f eine n -stellige **Funktion** f^I über dem Universum zu

Interpretation in der Aussagenlogik

- weist Aussagenvariablen Wert 1 oder 0 zu
- macht Formel wahr oder falsch

Interpretation in der Prädikatenlogik

- bestimmt das **Universum** (auch **Domäne** genannt)
- weist jedem n -stelligen **Funktionssymbol** f eine n -stellige **Funktion** $f^{\mathcal{I}}$ über dem Universum zu
- weist jedem n -stelligen **Prädikat** R eine n -stellige **Relation** $R^{\mathcal{I}}$ über dem Universum zu

Interpretation in der Aussagenlogik

- weist Aussagenvariablen Wert 1 oder 0 zu
- macht Formel wahr oder falsch

Interpretation in der Prädikatenlogik

- bestimmt das **Universum** (auch **Domäne** genannt)
- weist jedem n -stelligen **Funktionssymbol** f eine n -stellige **Funktion** $f^{\mathcal{I}}$ über dem Universum zu
- weist jedem n -stelligen **Prädikat** R eine n -stellige **Relation** $R^{\mathcal{I}}$ über dem Universum zu
- macht Formel wahr oder falsch

$$\varphi = \forall x R(x, f(x))$$

$$\varphi = \forall x R(x, f(x))$$

Beispiel 4.36 (Interpretation $\mathcal{I}$ mit Menschen als Domäne)

- Universum $\Delta^{\mathcal{I}}$: $M = \{\text{Anna}, \text{Bob}, \text{Clara}, \text{Dirk}\}$

$$\varphi = \forall x R(x, f(x))$$

Beispiel 4.36 (Interpretation $\mathcal{I}$ mit Menschen als Domäne)

- Universum $\Delta^{\mathcal{I}}$: $M = \{\text{Anna}, \text{Bob}, \text{Clara}, \text{Dirk}\}$
- Funktion $f^{\mathcal{I}}$: $\text{ehpartner}_M = \{\text{Anna} \mapsto \text{Bob}, \text{Bob} \mapsto \text{Anna}, \text{Clara} \mapsto \text{Dirk}, \text{Dirk} \mapsto \text{Clara}\}$

$$\varphi = \forall x R(x, f(x))$$

Beispiel 4.36 (Interpretation $\mathcal{I}$ mit Menschen als Domäne)

- Universum $\Delta^{\mathcal{I}}$: $M = \{\text{Anna}, \text{Bob}, \text{Clara}, \text{Dirk}\}$
- Funktion $f^{\mathcal{I}}$: $\text{ehpartner}_M = \{\text{Anna} \mapsto \text{Bob}, \text{Bob} \mapsto \text{Anna}, \text{Clara} \mapsto \text{Dirk}, \text{Dirk} \mapsto \text{Clara}\}$
- Relation $R^{\mathcal{I}}$: $\text{Mag}_M = \{(\text{Anna}, \text{Bob}), (\text{Anna}, \text{Clara}), (\text{Bob}, \text{Anna}), (\text{Bob}, \text{Dirk}), (\text{Clara}, \text{Dirk}), (\text{Dirk}, \text{Anna}), (\text{Dirk}, \text{Bob})\}$

$$\varphi = \forall x R(x, f(x))$$

Beispiel 4.36 (Interpretation $\mathcal{I}$ mit Menschen als Domäne)

- Universum $\Delta^{\mathcal{I}}$: $M = \{\text{Anna}, \text{Bob}, \text{Clara}, \text{Dirk}\}$
- Funktion $f^{\mathcal{I}}$: $\text{ehpartner}_M = \{\text{Anna} \mapsto \text{Bob}, \text{Bob} \mapsto \text{Anna}, \text{Clara} \mapsto \text{Dirk}, \text{Dirk} \mapsto \text{Clara}\}$
- Relation $R^{\mathcal{I}}$: $\text{Mag}_M = \{(\text{Anna}, \text{Bob}), (\text{Anna}, \text{Clara}), (\text{Bob}, \text{Anna}), (\text{Bob}, \text{Dirk}), (\text{Clara}, \text{Dirk}), (\text{Dirk}, \text{Anna}), (\text{Dirk}, \text{Bob})\}$
- $\mathcal{I}$ macht φ falsch (weil Dirk Clara nicht mag)

$$\varphi = \forall x R(x, f(x))$$

Beispiel 4.37 (Interpretation $\mathcal{I}$ mit Zahlen als Domäne)

- Universum $\Delta^{\mathcal{I}}: \mathbb{N} = \{0, 1, 2, \dots\}$

$$\varphi = \forall x R(x, f(x))$$

Beispiel 4.37 (Interpretation $\mathcal{I}$ mit Zahlen als Domäne)

- Universum $\Delta^{\mathcal{I}}: \mathbb{N} = \{0, 1, 2, \dots\}$
- Funktion $f^{\mathcal{I}}: s_{\mathbb{N}}$ (Nachfolger) = $\{0 \mapsto 1, 1 \mapsto 2, \dots\}$

$$\varphi = \forall x R(x, f(x))$$

Beispiel 4.37 (Interpretation $\mathcal{I}$ mit Zahlen als Domäne)

- Universum $\Delta^{\mathcal{I}}: \mathbb{N} = \{0, 1, 2, \dots\}$
- Funktion $f^{\mathcal{I}}: s_{\mathbb{N}}$ (Nachfolger) = $\{0 \mapsto 1, 1 \mapsto 2, \dots\}$
- Relation $R^{\mathcal{I}}: <_{\mathbb{N}}$ (kleiner als) = $\{(0, 1), (0, 2), \dots, (1, 2), (1, 3), \dots\}$

$$\varphi = \forall x R(x, f(x))$$

Beispiel 4.37 (Interpretation $\mathcal{I}$ mit Zahlen als Domäne)

- Universum $\Delta^{\mathcal{I}}: \mathbb{N} = \{0, 1, 2, \dots\}$
- Funktion $f^{\mathcal{I}}: s_{\mathbb{N}}$ (Nachfolger) = $\{0 \mapsto 1, 1 \mapsto 2, \dots\}$
- Relation $R^{\mathcal{I}}: <_{\mathbb{N}}$ (kleiner als) = $\{(0, 1), (0, 2), \dots, (1, 2), (1, 3), \dots\}$
- $\mathcal{I}$ macht φ wahr (weil jede Zahl kleiner ist als ihr Nachfolger)

$$\varphi = \forall x R(x, f(x))$$

Beispiel 4.37 (Interpretation $\mathcal{I}$ mit Zahlen als Domäne)

- Universum $\Delta^{\mathcal{I}}: \mathbb{N} = \{0, 1, 2, \dots\}$
- Funktion $f^{\mathcal{I}}: s_{\mathbb{N}}$ (Nachfolger) = $\{0 \mapsto 1, 1 \mapsto 2, \dots\}$
- Relation $R^{\mathcal{I}}: <_{\mathbb{N}}$ (kleiner als) = $\{(0, 1), (0, 2), \dots, (1, 2), (1, 3), \dots\}$
- $\mathcal{I}$ macht φ wahr (weil jede Zahl kleiner ist als ihr Nachfolger)

$$\varphi = \forall x R(x, f(x))$$

Beispiel 4.37 (Interpretation $\mathcal{I}$ mit Zahlen als Domäne)

- Universum $\Delta^{\mathcal{I}}: \mathbb{N} = \{0, 1, 2, \dots\}$
- Funktion $f^{\mathcal{I}}: s_{\mathbb{N}}$ (Nachfolger) = $\{0 \mapsto 1, 1 \mapsto 2, \dots\}$
- Relation $R^{\mathcal{I}}: <_{\mathbb{N}}$ (kleiner als) = $\{(0, 1), (0, 2), \dots, (1, 2), (1, 3), \dots\}$
- $\mathcal{I}$ macht φ wahr (weil jede Zahl kleiner ist als ihr Nachfolger)

Wahrheitswert hängt von der Interpretation ab

- der Interpretation der Funktionssymbole und Prädikate
Wenn $\mathcal{I}'$ R als $>_{\mathbb{N}}$ interpretiert, ist $\varphi^{\mathcal{I}'}$ falsch

$$\varphi = \forall x R(x, f(x))$$

Beispiel 4.37 (Interpretation $\mathcal{I}$ mit Zahlen als Domäne)

- Universum $\Delta^{\mathcal{I}}: \mathbb{N} = \{0, 1, 2, \dots\}$
- Funktion $f^{\mathcal{I}}: s_{\mathbb{N}}$ (Nachfolger) = $\{0 \mapsto 1, 1 \mapsto 2, \dots\}$
- Relation $R^{\mathcal{I}}: <_{\mathbb{N}}$ (kleiner als) = $\{(0, 1), (0, 2), \dots, (1, 2), (1, 3), \dots\}$
- $\mathcal{I}$ macht φ wahr (weil jede Zahl kleiner ist als ihr Nachfolger)

Wahrheitswert hängt von der Interpretation ab

- der Interpretation der Funktionssymbole und Prädikate
Wenn $\mathcal{I}'$ R als $>_{\mathbb{N}}$ interpretiert, ist $\varphi^{\mathcal{I}'}$ falsch
- dem Universum
 $\forall x \exists y (y < x)$ ist wahr in $\mathbb{Z}$, aber falsch in $\mathbb{N}$.

Übung 4.38

- 1 Sei $\varphi = \forall x \forall y (\neg G(x, y) \wedge R(x, y) \rightarrow \exists z (\neg G(z, x) \wedge \neg G(z, y) \wedge P(z)))$ und $\mathcal{I}$ wie folgt:

- $\Delta^{\mathcal{I}}$ ist die Menge aller Menschen;
- $G^{\mathcal{I}} = \{(x, x) \mid x \in \Delta^{\mathcal{I}}\}$ bezeichnet die Gleichheit;
- $R^{\mathcal{I}} = \text{streiten}$;
- $P^{\mathcal{I}} = \text{sich freuen}$.

Was ist die Aussage von $\varphi^{\mathcal{I}}$?

- 2 Sei $\psi = \forall x \exists y G(f(x, y), c)$ und $\mathcal{J}$ wie folgt:

- $\Delta^{\mathcal{J}} = \mathbb{Z}$;
- $G^{\mathcal{J}} = \{(x, x) \mid x \in \Delta^{\mathcal{J}}\}$ bezeichnet die Gleichheit;
- $f^{\mathcal{J}} = +_{\mathbb{Z}}$, d. h. Addition für Ganzzahlen;
- $c^{\mathcal{J}} = 0_{\mathbb{Z}}$, d. h. Null (als Element der Ganzzahlen).

Was ist die Aussage von $\psi^{\mathcal{J}}$? Ist $\psi^{\mathcal{J}}$ wahr oder falsch?

Wie werden freie Variablen interpretiert?

Wie werden freie Variablen interpretiert?

- Jede freie Variable wird von $\mathcal{I}$ auf ein Element der Domäne abgebildet

Wie werden freie Variablen interpretiert?

- Jede freie Variable wird von $\mathcal{I}$ auf ein Element der Domäne abgebildet
- Ähnlich Variablenzuweisung in Aussagenlogik

Wie werden freie Variablen interpretiert?

- Jede freie Variable wird von $\mathcal{I}$ auf ein Element der Domäne abgebildet
- Ähnlich Variablenzuweisung in Aussagenlogik
 - Jede Aussagenvariable wird auf ein Element von $\mathbb{B}$ abgebildet

Wie werden freie Variablen interpretiert?

- Jede freie Variable wird von $\mathcal{I}$ auf ein Element der Domäne abgebildet
- Ähnlich Variablenzuweisung in Aussagenlogik
 - Jede Aussagenvariable wird auf ein Element von $\mathbb{B}$ abgebildet
- In der Praxis interessieren wir uns meistens für Sätze.

Definition 4.39 (Interpretation in der PrädikatenLogik)

Für eine Formel φ mit Funktionssymbolen F , Prädikaten P und freien Variablen V ist eine **Interpretation** $\mathcal{I}$ ein Tripel $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}, \mu)$ mit

- $\Delta^{\mathcal{I}}$ ist eine nichtleere Menge,

Definition 4.39 (Interpretation in der Prädikatenlogik)

Für eine Formel φ mit Funktionssymbolen F , Prädikaten P und freien Variablen V ist eine **Interpretation** $\mathcal{I}$ ein Tripel $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}, \mu)$ mit

- $\Delta^{\mathcal{I}}$ ist eine nichtleere Menge,
- $\cdot^{\mathcal{I}}$ weist
 - jedem n -stelligen $f \in F$ eine n -stellige Funktion $f^{\mathcal{I}}$ über $\Delta^{\mathcal{I}}$ zu,

Definition 4.39 (Interpretation in der Prädikatenlogik)

Für eine Formel φ mit Funktionssymbolen F , Prädikaten P und freien Variablen V ist eine **Interpretation** $\mathcal{I}$ ein Tripel $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}, \mu)$ mit

- $\Delta^{\mathcal{I}}$ ist eine nichtleere Menge,
- $\cdot^{\mathcal{I}}$ weist
 - jedem n -stelligen $f \in F$ eine n -stellige Funktion $f^{\mathcal{I}}$ über $\Delta^{\mathcal{I}}$ zu,
 - jedem n -stelligen $R \in P$ eine n -stellige Relation $R^{\mathcal{I}}$ über $\Delta^{\mathcal{I}}$ zu.

Definition 4.39 (Interpretation in der Prädikatenlogik)

Für eine Formel φ mit Funktionssymbolen F , Prädikaten P und freien Variablen V ist eine **Interpretation** $\mathcal{I}$ ein Tripel $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}, \mu)$ mit

- $\Delta^{\mathcal{I}}$ ist eine nichtleere Menge,
- $\cdot^{\mathcal{I}}$ weist
 - jedem n -stelligen $f \in F$ eine n -stellige Funktion $f^{\mathcal{I}}$ über $\Delta^{\mathcal{I}}$ zu,
 - jedem n -stelligen $R \in P$ eine n -stellige Relation $R^{\mathcal{I}}$ über $\Delta^{\mathcal{I}}$ zu.
- $\mu : V \rightarrow \Delta^{\mathcal{I}}$ weist jedem $x \in V$ ein Element $x^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ zu.

Definition 4.39 (Interpretation in der Prädikatenlogik)

Für eine Formel φ mit Funktionssymbolen F , Prädikaten P und freien Variablen V ist eine **Interpretation** $\mathcal{I}$ ein Tripel $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}, \mu)$ mit

- $\Delta^{\mathcal{I}}$ ist eine nichtleere Menge,
 - $\cdot^{\mathcal{I}}$ weist
 - jedem n -stelligen $f \in F$ eine n -stellige Funktion $f^{\mathcal{I}}$ über $\Delta^{\mathcal{I}}$ zu,
 - jedem n -stelligen $R \in P$ eine n -stellige Relation $R^{\mathcal{I}}$ über $\Delta^{\mathcal{I}}$ zu.
 - $\mu : V \rightarrow \Delta^{\mathcal{I}}$ weist jedem $x \in V$ ein Element $x^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ zu.
-
- Domäne darf **nicht leer** sein.

Definition 4.39 (Interpretation in der Prädikatenlogik)

Für eine Formel φ mit Funktionssymbolen F , Prädikaten P und freien Variablen V ist eine **Interpretation** $\mathcal{I}$ ein Tripel $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}, \mu)$ mit

- $\Delta^{\mathcal{I}}$ ist eine nichtleere Menge,
 - $\cdot^{\mathcal{I}}$ weist
 - jedem n -stelligen $f \in F$ eine n -stellige Funktion $f^{\mathcal{I}}$ über $\Delta^{\mathcal{I}}$ zu,
 - jedem n -stelligen $R \in P$ eine n -stellige Relation $R^{\mathcal{I}}$ über $\Delta^{\mathcal{I}}$ zu.
 - $\mu : V \rightarrow \Delta^{\mathcal{I}}$ weist jedem $x \in V$ ein Element $x^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ zu.
-
- Domäne darf **nicht leer** sein.
 - Nullstellige Funktionssymbole (Konstantensymbole) werden auf Elemente der Domäne abgebildet.

Definition 4.39 (Interpretation in der Prädikatenlogik)

Für eine Formel φ mit Funktionssymbolen F , Prädikaten P und freien Variablen V ist eine **Interpretation** $\mathcal{I}$ ein Tripel $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}, \mu)$ mit

- $\Delta^{\mathcal{I}}$ ist eine nichtleere Menge,
 - $\cdot^{\mathcal{I}}$ weist
 - jedem n -stelligen $f \in F$ eine n -stellige Funktion $f^{\mathcal{I}}$ über $\Delta^{\mathcal{I}}$ zu,
 - jedem n -stelligen $R \in P$ eine n -stellige Relation $R^{\mathcal{I}}$ über $\Delta^{\mathcal{I}}$ zu.
 - $\mu : V \rightarrow \Delta^{\mathcal{I}}$ weist jedem $x \in V$ ein Element $x^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ zu.
-
- Domäne darf **nicht leer** sein.
 - Nullstellige Funktionssymbole (Konstantensymbole) werden auf Elemente der Domäne abgebildet.
 - Nullstellige Prädikate werden auf $\{()\}$ oder $\{\}$ abgebildet.

Definition 4.39 (Interpretation in der Prädikatenlogik)

Für eine Formel φ mit Funktionssymbolen F , Prädikaten P und freien Variablen V ist eine **Interpretation** $\mathcal{I}$ ein Tripel $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}, \mu)$ mit

- $\Delta^{\mathcal{I}}$ ist eine nichtleere Menge,
 - $\cdot^{\mathcal{I}}$ weist
 - jedem n -stelligen $f \in F$ eine n -stellige Funktion $f^{\mathcal{I}}$ über $\Delta^{\mathcal{I}}$ zu,
 - jedem n -stelligen $R \in P$ eine n -stellige Relation $R^{\mathcal{I}}$ über $\Delta^{\mathcal{I}}$ zu.
 - $\mu : V \rightarrow \Delta^{\mathcal{I}}$ weist jedem $x \in V$ ein Element $x^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ zu.
-
- Domäne darf **nicht leer** sein.
 - Nullstellige Funktionssymbole (Konstantensymbole) werden auf Elemente der Domäne abgebildet.
 - Nullstellige Prädikate werden auf $\{()\}$ oder $\{\}$ abgebildet.
 - Aussagenvariablen: $\{()\} \rightsquigarrow 1_{\mathbb{B}}$; $\{\} \rightsquigarrow 0_{\mathbb{B}}$

Der Wahrheitswert komplexer Formeln wird rekursiv bestimmt:

- Für einen komplexen Term $t = f(t_1, \dots, t_n)$ werden

Der Wahrheitswert komplexer Formeln wird rekursiv bestimmt:

- Für einen komplexen **Term** $t = f(t_1, \dots, t_n)$ werden
 - mögliche freie Variablen $x, y, \dots$ durch $\mu(x), \mu(y), \dots$ ersetzt;

Der Wahrheitswert komplexer Formeln wird rekursiv bestimmt:

- Für einen komplexen **Term** $t = f(t_1, \dots, t_n)$ werden
 - mögliche freie Variablen $x, y, \dots$ durch $\mu(x), \mu(y), \dots$ ersetzt;
 - die Terme $t_1, \dots, t_n$ rekursiv ausgewertet als $t_1^{\mathcal{I}}, \dots, t_n^{\mathcal{I}}$;

Der Wahrheitswert komplexer Formeln wird rekursiv bestimmt:

- Für einen komplexen **Term** $t = f(t_1, \dots, t_n)$ werden
 - mögliche freie Variablen $x, y, \dots$ durch $\mu(x), \mu(y), \dots$ ersetzt;
 - die Terme $t_1, \dots, t_n$ rekursiv ausgewertet als $t_1^{\mathcal{I}}, \dots, t_n^{\mathcal{I}}$;
 - die Funktion $f^{\mathcal{I}}$ für diese Ergebnisse ausgewertet: $t^{\mathcal{I}} = f^{\mathcal{I}}((\mu(t_1))^{\mathcal{I}}, \dots, (\mu(t_n))^{\mathcal{I}})$.

Der Wahrheitswert komplexer Formeln wird rekursiv bestimmt:

- Für einen komplexen **Term** $t = f(t_1, \dots, t_n)$ werden
 - mögliche freie Variablen $x, y, \dots$ durch $\mu(x), \mu(y), \dots$ ersetzt;
 - die Terme $t_1, \dots, t_n$ rekursiv ausgewertet als $t_1^{\mathcal{I}}, \dots, t_n^{\mathcal{I}}$;
 - die Funktion $f^{\mathcal{I}}$ für diese Ergebnisse ausgewertet: $t^{\mathcal{I}} = f^{\mathcal{I}}((\mu(t_1))^{\mathcal{I}}, \dots, (\mu(t_n))^{\mathcal{I}})$.
- Für ein **Atom** $\varphi = P(t_1, \dots, t_n)$ werden

Der Wahrheitswert komplexer Formeln wird rekursiv bestimmt:

- Für einen komplexen **Term** $t = f(t_1, \dots, t_n)$ werden
 - mögliche freie Variablen $x, y, \dots$ durch $\mu(x), \mu(y), \dots$ ersetzt;
 - die Terme $t_1, \dots, t_n$ rekursiv ausgewertet als $t_1^{\mathcal{I}}, \dots, t_n^{\mathcal{I}}$;
 - die Funktion $f^{\mathcal{I}}$ für diese Ergebnisse ausgewertet: $t^{\mathcal{I}} = f^{\mathcal{I}}((\mu(t_1))^{\mathcal{I}}, \dots, (\mu(t_n))^{\mathcal{I}})$.
- Für ein **Atom** $\varphi = P(t_1, \dots, t_n)$ werden
 - die Terme $t_1, \dots, t_n$ rekursiv ausgewertet;

Der Wahrheitswert komplexer Formeln wird rekursiv bestimmt:

- Für einen komplexen **Term** $t = f(t_1, \dots, t_n)$ werden
 - mögliche freie Variablen $x, y, \dots$ durch $\mu(x), \mu(y), \dots$ ersetzt;
 - die Terme $t_1, \dots, t_n$ rekursiv ausgewertet als $t_1^{\mathcal{I}}, \dots, t_n^{\mathcal{I}}$;
 - die Funktion $f^{\mathcal{I}}$ für diese Ergebnisse ausgewertet: $t^{\mathcal{I}} = f^{\mathcal{I}}((\mu(t_1))^{\mathcal{I}}, \dots, (\mu(t_n))^{\mathcal{I}})$.
- Für ein **Atom** $\varphi = P(t_1, \dots, t_n)$ werden
 - die Terme $t_1, \dots, t_n$ rekursiv ausgewertet;
 - geprüft, ob das sich ergebende Tupel in $P^{\mathcal{I}}$ enthalten ist.

Der Wahrheitswert komplexer Formeln wird rekursiv bestimmt:

- Für einen komplexen **Term** $t = f(t_1, \dots, t_n)$ werden
 - mögliche freie Variablen $x, y, \dots$ durch $\mu(x), \mu(y), \dots$ ersetzt;
 - die Terme $t_1, \dots, t_n$ rekursiv ausgewertet als $t_1^{\mathcal{I}}, \dots, t_n^{\mathcal{I}}$;
 - die Funktion $f^{\mathcal{I}}$ für diese Ergebnisse ausgewertet: $t^{\mathcal{I}} = f^{\mathcal{I}}((\mu(t_1))^{\mathcal{I}}, \dots, (\mu(t_n))^{\mathcal{I}})$.
- Für ein **Atom** $\varphi = P(t_1, \dots, t_n)$ werden
 - die Terme $t_1, \dots, t_n$ rekursiv ausgewertet;
 - geprüft, ob das sich ergebende Tupel in $P^{\mathcal{I}}$ enthalten ist.
- Für **gebundene** Variablen $\forall x\varphi/\exists x\varphi$ wird geprüft, ob φ für alle/ein Element der Domäne wahr ist.

Der Wahrheitswert komplexer Formeln wird rekursiv bestimmt:

- Für einen komplexen **Term** $t = f(t_1, \dots, t_n)$ werden
 - mögliche freie Variablen $x, y, \dots$ durch $\mu(x), \mu(y), \dots$ ersetzt;
 - die Terme $t_1, \dots, t_n$ rekursiv ausgewertet als $t_1^{\mathcal{I}}, \dots, t_n^{\mathcal{I}}$;
 - die Funktion $f^{\mathcal{I}}$ für diese Ergebnisse ausgewertet: $t^{\mathcal{I}} = f^{\mathcal{I}}((\mu(t_1))^{\mathcal{I}}, \dots, (\mu(t_n))^{\mathcal{I}})$.
- Für ein **Atom** $\varphi = P(t_1, \dots, t_n)$ werden
 - die Terme $t_1, \dots, t_n$ rekursiv ausgewertet;
 - geprüft, ob das sich ergebende Tupel in $P^{\mathcal{I}}$ enthalten ist.
- Für **gebundene** Variablen $\forall x\varphi/\exists x\varphi$ wird geprüft, ob φ für alle/ein Element der Domäne wahr ist.
- Formeln mit **Junktoren** werden wie in der Aussagenlogik ausgewertet.

Beispiel 4.40

$$\varphi = \neg P(x) \wedge \exists y R(c, f(y))$$

$$\Delta^{\mathcal{I}} = \mathbb{N}$$

Beispiel 4.40

$$\varphi = \neg P(x) \wedge \exists y R(c, f(y))$$

$$\begin{aligned}\Delta^{\mathcal{I}} &= \mathbb{N} \\ \mu &= \{x \mapsto 4_{\mathbb{N}}\}\end{aligned}$$

Beispiel 4.40

$$\varphi = \neg P(x) \wedge \exists y R(c, f(y))$$

$$\begin{aligned}\Delta^{\mathcal{I}} &= \mathbb{N} \\ \mu &= \{x \mapsto 4_{\mathbb{N}}\}\end{aligned}$$

$$\begin{aligned}P^{\mathcal{I}} &= \text{Prim}_{\mathbb{N}} \\ R^{\mathcal{I}} &= \leq_{\mathbb{N}}\end{aligned}$$

Beispiel 4.40

$$\varphi = \neg P(x) \wedge \exists y R(c, f(y))$$

$$\begin{aligned}\Delta^{\mathcal{I}} &= \mathbb{N} \\ \mu &= \{x \mapsto 4_{\mathbb{N}}\}\end{aligned}$$

$$\begin{aligned}P^{\mathcal{I}} &= \text{Prim}_{\mathbb{N}} \\ R^{\mathcal{I}} &= \leq_{\mathbb{N}}\end{aligned}$$

$$\begin{aligned}f^{\mathcal{I}} &= s_{\mathbb{N}} \\ c^{\mathcal{I}} &= 17_{\mathbb{N}}\end{aligned}$$

Beispiel 4.40

$$\varphi = \neg P(x) \wedge \exists y R(c, f(y))$$

$$\begin{aligned}\Delta^{\mathcal{I}} &= \mathbb{N} \\ \mu &= \{x \mapsto 4_{\mathbb{N}}\}\end{aligned}$$

$$\begin{aligned}P^{\mathcal{I}} &= \text{Prim}_{\mathbb{N}} \\ R^{\mathcal{I}} &= \leq_{\mathbb{N}}\end{aligned}$$

$$\begin{aligned}f^{\mathcal{I}} &= s_{\mathbb{N}} \\ c^{\mathcal{I}} &= 17_{\mathbb{N}}\end{aligned}$$

Bestimmung von $\varphi^{\mathcal{I}}$:

$$\varphi^{\mathcal{I}} = \neg P^{\mathcal{I}}(\mu(x)) \wedge \exists y R^{\mathcal{I}}(c^{\mathcal{I}}, f^{\mathcal{I}}(y))$$

Beispiel 4.40

$$\varphi = \neg P(x) \wedge \exists y R(c, f(y))$$

$$\begin{aligned}\Delta^{\mathcal{I}} &= \mathbb{N} \\ \mu &= \{x \mapsto 4_{\mathbb{N}}\}\end{aligned}$$

$$\begin{aligned}P^{\mathcal{I}} &= \text{Prim}_{\mathbb{N}} \\ R^{\mathcal{I}} &= \leq_{\mathbb{N}}\end{aligned}$$

$$\begin{aligned}f^{\mathcal{I}} &= s_{\mathbb{N}} \\ c^{\mathcal{I}} &= 17_{\mathbb{N}}\end{aligned}$$

Bestimmung von $\varphi^{\mathcal{I}}$:

$$\begin{aligned}\varphi^{\mathcal{I}} &= \\ \mu \text{ auswerten}\end{aligned}$$

$$\begin{aligned}&\neg P^{\mathcal{I}}(\mu(x)) \wedge \exists y R^{\mathcal{I}}(c^{\mathcal{I}}, f^{\mathcal{I}}(y)) \\ &\neg P^{\mathcal{I}}(4_{\mathbb{N}}) \wedge \exists y R^{\mathcal{I}}(c^{\mathcal{I}}, f^{\mathcal{I}}(y))\end{aligned}$$

Beispiel 4.40

$$\varphi = \neg P(x) \wedge \exists y R(c, f(y))$$

$$\begin{aligned}\Delta^{\mathcal{I}} &= \mathbb{N} \\ \mu &= \{x \mapsto 4_{\mathbb{N}}\}\end{aligned}$$

$$\begin{aligned}P^{\mathcal{I}} &= \text{Prim}_{\mathbb{N}} \\ R^{\mathcal{I}} &= \leq_{\mathbb{N}}\end{aligned}$$

$$\begin{aligned}f^{\mathcal{I}} &= s_{\mathbb{N}} \\ c^{\mathcal{I}} &= 17_{\mathbb{N}}\end{aligned}$$

Bestimmung von $\varphi^{\mathcal{I}}$:

$$\varphi^{\mathcal{I}} =$$

μ auswerten

Funktionssymbole

$$\neg P^{\mathcal{I}}(\mu(x)) \wedge \exists y R^{\mathcal{I}}(c^{\mathcal{I}}, f^{\mathcal{I}}(y))$$

$$\neg P^{\mathcal{I}}(4_{\mathbb{N}}) \wedge \exists y R^{\mathcal{I}}(c^{\mathcal{I}}, f^{\mathcal{I}}(y))$$

$$\neg P^{\mathcal{I}}(4_{\mathbb{N}}) \wedge \exists y R^{\mathcal{I}}(17_{\mathbb{N}}, s_{\mathbb{N}}(y))$$

Beispiel 4.40

$$\varphi = \neg P(x) \wedge \exists y R(c, f(y))$$

$$\begin{aligned}\Delta^{\mathcal{I}} &= \mathbb{N} \\ \mu &= \{x \mapsto 4_{\mathbb{N}}\}\end{aligned}$$

$$\begin{aligned}P^{\mathcal{I}} &= \text{Prim}_{\mathbb{N}} \\ R^{\mathcal{I}} &= \leq_{\mathbb{N}}\end{aligned}$$

$$\begin{aligned}f^{\mathcal{I}} &= s_{\mathbb{N}} \\ c^{\mathcal{I}} &= 17_{\mathbb{N}}\end{aligned}$$

Bestimmung von $\varphi^{\mathcal{I}}$:

$$\varphi^{\mathcal{I}} =$$

μ auswerten

Funktionssymbole

Prädikate

$$\neg P^{\mathcal{I}}(\mu(x)) \wedge \exists y R^{\mathcal{I}}(c^{\mathcal{I}}, f^{\mathcal{I}}(y))$$

$$\neg P^{\mathcal{I}}(4_{\mathbb{N}}) \wedge \exists y R^{\mathcal{I}}(c^{\mathcal{I}}, f^{\mathcal{I}}(y))$$

$$\neg P^{\mathcal{I}}(4_{\mathbb{N}}) \wedge \exists y R^{\mathcal{I}}(17_{\mathbb{N}}, s_{\mathbb{N}}(y))$$

$$\neg \text{Prim}_{\mathbb{N}}(4_{\mathbb{N}}) \wedge \exists y \leq_{\mathbb{N}} (17_{\mathbb{N}}, s_{\mathbb{N}}(y))$$

Beispiel 4.40

$$\varphi = \neg P(x) \wedge \exists y R(c, f(y))$$

$$\begin{aligned}\Delta^{\mathcal{I}} &= \mathbb{N} \\ \mu &= \{x \mapsto 4_{\mathbb{N}}\}\end{aligned}$$

$$\begin{aligned}P^{\mathcal{I}} &= \text{Prim}_{\mathbb{N}} \\ R^{\mathcal{I}} &= \leq_{\mathbb{N}}\end{aligned}$$

$$\begin{aligned}f^{\mathcal{I}} &= s_{\mathbb{N}} \\ c^{\mathcal{I}} &= 17_{\mathbb{N}}\end{aligned}$$

Bestimmung von $\varphi^{\mathcal{I}}$:

$$\varphi^{\mathcal{I}} =$$

μ auswerten

Funktionssymbole

Prädikate

Quantor: Wähle $y = 20_{\mathbb{N}}$

$$\neg P^{\mathcal{I}}(\mu(x)) \wedge \exists y R^{\mathcal{I}}(c^{\mathcal{I}}, f^{\mathcal{I}}(y))$$

$$\neg P^{\mathcal{I}}(4_{\mathbb{N}}) \wedge \exists y R^{\mathcal{I}}(c^{\mathcal{I}}, f^{\mathcal{I}}(y))$$

$$\neg P^{\mathcal{I}}(4_{\mathbb{N}}) \wedge \exists y R^{\mathcal{I}}(17_{\mathbb{N}}, s_{\mathbb{N}}(y))$$

$$\neg \text{Prim}_{\mathbb{N}}(4_{\mathbb{N}}) \wedge \exists y \leq_{\mathbb{N}} (17_{\mathbb{N}}, s_{\mathbb{N}}(y))$$

$$\neg \text{Prim}_{\mathbb{N}}(4_{\mathbb{N}}) \wedge \leq_{\mathbb{N}} (17_{\mathbb{N}}, s_{\mathbb{N}}(20_{\mathbb{N}}))$$

Beispiel 4.40

$$\varphi = \neg P(x) \wedge \exists y R(c, f(y))$$

$$\Delta^{\mathcal{I}} = \mathbb{N}$$

$$\mu = \{x \mapsto 4_{\mathbb{N}}\}$$

$$P^{\mathcal{I}} = \text{Prim}_{\mathbb{N}}$$

$$R^{\mathcal{I}} = \leq_{\mathbb{N}}$$

$$f^{\mathcal{I}} = s_{\mathbb{N}}$$

$$c^{\mathcal{I}} = 17_{\mathbb{N}}$$

Bestimmung von $\varphi^{\mathcal{I}}$:

$$\varphi^{\mathcal{I}} =$$

μ auswerten

Funktionssymbole

Prädikate

Quantor: Wähle $y = 20_{\mathbb{N}}$

Funktion auswerten

$$\neg P^{\mathcal{I}}(\mu(x)) \wedge \exists y R^{\mathcal{I}}(c^{\mathcal{I}}, f^{\mathcal{I}}(y))$$

$$\neg P^{\mathcal{I}}(4_{\mathbb{N}}) \wedge \exists y R^{\mathcal{I}}(c^{\mathcal{I}}, f^{\mathcal{I}}(y))$$

$$\neg P^{\mathcal{I}}(4_{\mathbb{N}}) \wedge \exists y R^{\mathcal{I}}(17_{\mathbb{N}}, s_{\mathbb{N}}(y))$$

$$\neg \text{Prim}_{\mathbb{N}}(4_{\mathbb{N}}) \wedge \exists y \leq_{\mathbb{N}} (17_{\mathbb{N}}, s_{\mathbb{N}}(y))$$

$$\neg \text{Prim}_{\mathbb{N}}(4_{\mathbb{N}}) \wedge \leq_{\mathbb{N}} (17_{\mathbb{N}}, s_{\mathbb{N}}(20_{\mathbb{N}}))$$

$$\neg \text{Prim}_{\mathbb{N}}(4_{\mathbb{N}}) \wedge \leq_{\mathbb{N}} (17_{\mathbb{N}}, 21_{\mathbb{N}})$$

Beispiel 4.40

$$\varphi = \neg P(x) \wedge \exists y R(c, f(y))$$

$$\begin{aligned}\Delta^{\mathcal{I}} &= \mathbb{N} \\ \mu &= \{x \mapsto 4_{\mathbb{N}}\}\end{aligned}$$

$$\begin{aligned}P^{\mathcal{I}} &= \text{Prim}_{\mathbb{N}} \\ R^{\mathcal{I}} &= \leq_{\mathbb{N}}\end{aligned}$$

$$\begin{aligned}f^{\mathcal{I}} &= s_{\mathbb{N}} \\ c^{\mathcal{I}} &= 17_{\mathbb{N}}\end{aligned}$$

Bestimmung von $\varphi^{\mathcal{I}}$:

$$\varphi^{\mathcal{I}} =$$

μ auswerten

Funktionssymbole

Prädikate

Quantor: Wähle $y = 20_{\mathbb{N}}$

Funktion auswerten

Relationen auswerten

$$\neg P^{\mathcal{I}}(\mu(x)) \wedge \exists y R^{\mathcal{I}}(c^{\mathcal{I}}, f^{\mathcal{I}}(y))$$

$$\neg P^{\mathcal{I}}(4_{\mathbb{N}}) \wedge \exists y R^{\mathcal{I}}(c^{\mathcal{I}}, f^{\mathcal{I}}(y))$$

$$\neg P^{\mathcal{I}}(4_{\mathbb{N}}) \wedge \exists y R^{\mathcal{I}}(17_{\mathbb{N}}, s_{\mathbb{N}}(y))$$

$$\neg \text{Prim}_{\mathbb{N}}(4_{\mathbb{N}}) \wedge \exists y \leq_{\mathbb{N}} (17_{\mathbb{N}}, s_{\mathbb{N}}(y))$$

$$\neg \text{Prim}_{\mathbb{N}}(4_{\mathbb{N}}) \wedge \leq_{\mathbb{N}} (17_{\mathbb{N}}, s_{\mathbb{N}}(20_{\mathbb{N}}))$$

$$\neg \text{Prim}_{\mathbb{N}}(4_{\mathbb{N}}) \wedge \leq_{\mathbb{N}} (17_{\mathbb{N}}, 21_{\mathbb{N}})$$

$$\neg 0_{\mathbb{B}} \wedge 1_{\mathbb{B}}$$

Beispiel 4.40

$$\varphi = \neg P(x) \wedge \exists y R(c, f(y))$$

$$\begin{aligned}\Delta^{\mathcal{I}} &= \mathbb{N} \\ \mu &= \{x \mapsto 4_{\mathbb{N}}\}\end{aligned}$$

$$\begin{aligned}P^{\mathcal{I}} &= \text{Prim}_{\mathbb{N}} \\ R^{\mathcal{I}} &= \leq_{\mathbb{N}}\end{aligned}$$

$$\begin{aligned}f^{\mathcal{I}} &= s_{\mathbb{N}} \\ c^{\mathcal{I}} &= 17_{\mathbb{N}}\end{aligned}$$

Bestimmung von $\varphi^{\mathcal{I}}$:

$$\varphi^{\mathcal{I}} =$$

μ auswerten

Funktionssymbole

Prädikate

Quantor: Wähle $y = 20_{\mathbb{N}}$

Funktion auswerten

Relationen auswerten

Junktoren auswerten

$$\neg P^{\mathcal{I}}(\mu(x)) \wedge \exists y R^{\mathcal{I}}(c^{\mathcal{I}}, f^{\mathcal{I}}(y))$$

$$\neg P^{\mathcal{I}}(4_{\mathbb{N}}) \wedge \exists y R^{\mathcal{I}}(c^{\mathcal{I}}, f^{\mathcal{I}}(y))$$

$$\neg P^{\mathcal{I}}(4_{\mathbb{N}}) \wedge \exists y R^{\mathcal{I}}(17_{\mathbb{N}}, s_{\mathbb{N}}(y))$$

$$\neg \text{Prim}_{\mathbb{N}}(4_{\mathbb{N}}) \wedge \exists y \leq_{\mathbb{N}} (17_{\mathbb{N}}, s_{\mathbb{N}}(y))$$

$$\neg \text{Prim}_{\mathbb{N}}(4_{\mathbb{N}}) \wedge \leq_{\mathbb{N}} (17_{\mathbb{N}}, s_{\mathbb{N}}(20_{\mathbb{N}}))$$

$$\neg \text{Prim}_{\mathbb{N}}(4_{\mathbb{N}}) \wedge \leq_{\mathbb{N}} (17_{\mathbb{N}}, 21_{\mathbb{N}})$$

$$\neg 0_{\mathbb{B}} \wedge 1_{\mathbb{B}}$$

$$1_{\mathbb{B}}$$

Definition 4.41 (Modell)

Eine Interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}, \mu)$ **erfüllt** eine Formel φ (ist ein **Modell** für φ , $\mathcal{I} \models \varphi$), wenn

Definition 4.41 (Modell)

Eine Interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}, \mu)$ **erfüllt** eine Formel φ (ist ein **Modell** für φ , $\mathcal{I} \models \varphi$), wenn

- $\varphi = P(t_1, \dots, t_n)$ ein **Atom** ist und $((\mu(t_1))^{\mathcal{I}}, \dots, (\mu(t_n))^{\mathcal{I}}) \in P^{\mathcal{I}}$ gilt;

Definition 4.41 (Modell)

Eine Interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}, \mu)$ **erfüllt** eine Formel φ (ist ein **Modell** für φ , $\mathcal{I} \models \varphi$), wenn

- $\varphi = P(t_1, \dots, t_n)$ ein **Atom** ist und $((\mu(t_1))^{\mathcal{I}}, \dots, (\mu(t_n))^{\mathcal{I}}) \in P^{\mathcal{I}}$ gilt;
- $\varphi = \forall x \psi(x)$ gilt und für jedes $x \in \Delta^{\mathcal{I}}$ $\psi(x)$ gilt;

Definition 4.41 (Modell)

Eine Interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}, \mu)$ **erfüllt** eine Formel φ (ist ein **Modell** für φ , $\mathcal{I} \models \varphi$), wenn

- $\varphi = P(t_1, \dots, t_n)$ ein **Atom** ist und $((\mu(t_1))^{\mathcal{I}}, \dots, (\mu(t_n))^{\mathcal{I}}) \in P^{\mathcal{I}}$ gilt;
- $\varphi = \forall x \psi(x)$ gilt und für jedes $x \in \Delta^{\mathcal{I}}$ $\psi(x)$ gilt;
- $\varphi = \exists x \psi(x)$ gilt und für ein $x \in \Delta^{\mathcal{I}}$ $\psi(x)$ gilt;

Definition 4.41 (Modell)

Eine Interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}, \mu)$ **erfüllt** eine Formel φ (ist ein **Modell** für φ , $\mathcal{I} \models \varphi$), wenn

- $\varphi = P(t_1, \dots, t_n)$ ein **Atom** ist und $((\mu(t_1))^{\mathcal{I}}, \dots, (\mu(t_n))^{\mathcal{I}}) \in P^{\mathcal{I}}$ gilt;
- $\varphi = \forall x \psi(x)$ gilt und für jedes $x \in \Delta^{\mathcal{I}}$ $\psi(x)$ gilt;
- $\varphi = \exists x \psi(x)$ gilt und für ein $x \in \Delta^{\mathcal{I}}$ $\psi(x)$ gilt;
- φ einen Junktore enthält (z. B., $\varphi = \neg \varphi_1$, $\varphi = \varphi_1 \wedge \varphi_2$) und entsprechend der Wahrheitstabelle als wahr ausgewertet wird.

Definition 4.41 (Modell)

Eine Interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}, \mu)$ **erfüllt** eine Formel φ (ist ein **Modell** für φ , $\mathcal{I} \models \varphi$), wenn

- $\varphi = P(t_1, \dots, t_n)$ ein **Atom** ist und $((\mu(t_1))^{\mathcal{I}}, \dots, (\mu(t_n))^{\mathcal{I}}) \in P^{\mathcal{I}}$ gilt;
- $\varphi = \forall x \psi(x)$ gilt und für jedes $x \in \Delta^{\mathcal{I}}$ $\psi(x)$ gilt;
- $\varphi = \exists x \psi(x)$ gilt und für ein $x \in \Delta^{\mathcal{I}}$ $\psi(x)$ gilt;
- φ einen Junktore enthält (z. B., $\varphi = \neg \varphi_1$, $\varphi = \varphi_1 \wedge \varphi_2$) und entsprechend der Wahrheitstabelle als wahr ausgewertet wird.

Definition 4.42 (Logische Implikation)

Eine Formel φ **impliziert logisch** eine Formel ψ ($\varphi \models \psi$), wenn jedes Modell von φ auch Modell von ψ ist.

Übung 4.43

Gegeben sei die Signatur $(\{f^{(1)}, c^{(0)}\}, \{R^{(2)}, =^{(2)}\})$.

Finden Sie für jeden der folgenden Sätze eine Interpretation, die ihn wahr macht, und eine Interpretation, die ihn falsch macht.

Hierbei soll das Prädikat „ $=$ “ als Gleichheit von Elementen der Domänen interpretiert werden. Verwenden Sie als Domänen Mengen von Zahlen.

1 $\forall x \forall y (R(x, y) \rightarrow \exists z (R(x, z) \wedge R(z, y)))$

2 $\neg \exists x (f(x) = c)$

3 $\forall x \forall y (f(x) = f(y) \rightarrow x = y)$

Beispiel 4.44 (Körperaxiome)

Abschluss unter Addition

$$\forall x \forall y \exists z (x + y = z)$$

Beispiel 4.44 (Körperaxiome)

Abschluss unter Addition

$$\forall x \forall y \exists z (x + y = z)$$

Assoziativität der Addition

$$\forall x \forall y \forall z ((x + (y + z)) = ((x + y) + z))$$

Beispiel 4.44 (Körperaxiome)

Abschluss unter Addition

$$\forall x \forall y \exists z (x + y = z)$$

Assoziativität der Addition

$$\forall x \forall y \forall z ((x + (y + z)) = ((x + y) + z))$$

Kommutativität der Addition

$$\forall x \forall y (x + y = y + x)$$

Beispiel 4.44 (Körperaxiome)

Abschluss unter Addition

$$\forall x \forall y \exists z (x + y = z)$$

Assoziativität der Addition

$$\forall x \forall y \forall z ((x + (y + z)) = ((x + y) + z))$$

Kommutativität der Addition

$$\forall x \forall y (x + y = y + x)$$

Neutrales Element der Addition

$$\forall x (x + 0 = x)$$

Beispiel 4.44 (Körperaxiome)

Abschluss unter Addition

$$\forall x \forall y \exists z (x + y = z)$$

Assoziativität der Addition

$$\forall x \forall y \forall z ((x + (y + z)) = ((x + y) + z))$$

Kommutativität der Addition

$$\forall x \forall y (x + y = y + x)$$

Neutrales Element der Addition

$$\forall x (x + 0 = x)$$

Inverses Element der Addition

$$\forall x \exists y (x + y = 0)$$

Beispiel 4.44 (Körperaxiome)

Abschluss unter Addition	$\forall x \forall y \exists z (x + y = z)$
Assoziativität der Addition	$\forall x \forall y \forall z ((x + (y + z)) = ((x + y) + z))$
Kommutativität der Addition	$\forall x \forall y (x + y = y + x)$
Neutrales Element der Addition	$\forall x (x + 0 = x)$
Inverses Element der Addition	$\forall x \exists y (x + y = 0)$
Abschluss unter Multiplikation	$\forall x \forall y \exists z (x \cdot y = z)$
Assoziativität der Multiplikation	$\forall x \forall y \forall z ((x \cdot (y \cdot z)) = ((x \cdot y) \cdot z))$
Kommutativität der Multiplikation	$\forall x \forall y (x \cdot y = y \cdot x)$
Neutrales Element der Multiplikation	$\forall x (x \cdot 1 = x)$
Inverses Element der Multiplikation	$\forall x (x = 0 \vee \exists y (x \cdot y = 1))$

Beispiel 4.44 (Körperaxiome)

Abschluss unter Addition	$\forall x \forall y \exists z (x + y = z)$
Assoziativität der Addition	$\forall x \forall y \forall z ((x + (y + z)) = ((x + y) + z))$
Kommutativität der Addition	$\forall x \forall y (x + y = y + x)$
Neutrales Element der Addition	$\forall x (x + 0 = x)$
Inverses Element der Addition	$\forall x \exists y (x + y = 0)$
Abschluss unter Multiplikation	$\forall x \forall y \exists z (x \cdot y = z)$
Assoziativität der Multiplikation	$\forall x \forall y \forall z ((x \cdot (y \cdot z)) = ((x \cdot y) \cdot z))$
Kommutativität der Multiplikation	$\forall x \forall y (x \cdot y = y \cdot x)$
Neutrales Element der Multiplikation	$\forall x (x \cdot 1 = x)$
Inverses Element der Multiplikation	$\forall x (x = 0 \vee \exists y (x \cdot y = 1))$
Distributivität	$\forall x \forall y \forall z (x \cdot (y + z) = (x \cdot y) + (x \cdot z))$

Beispiel 4.44 (Körperaxiome)

Abschluss unter Addition	$\forall x \forall y \exists z (x + y = z)$
Assoziativität der Addition	$\forall x \forall y \forall z ((x + (y + z)) = ((x + y) + z))$
Kommutativität der Addition	$\forall x \forall y (x + y = y + x)$
Neutrales Element der Addition	$\forall x (x + 0 = x)$
Inverses Element der Addition	$\forall x \exists y (x + y = 0)$
Abschluss unter Multiplikation	$\forall x \forall y \exists z (x \cdot y = z)$
Assoziativität der Multiplikation	$\forall x \forall y \forall z ((x \cdot (y \cdot z)) = ((x \cdot y) \cdot z))$
Kommutativität der Multiplikation	$\forall x \forall y (x \cdot y = y \cdot x)$
Neutrales Element der Multiplikation	$\forall x (x \cdot 1 = x)$
Inverses Element der Multiplikation	$\forall x (x = 0 \vee \exists y (x \cdot y = 1))$
Distributivität	$\forall x \forall y \forall z (x \cdot (y + z) = (x \cdot y) + (x \cdot z))$

Warum „PrädikatenLogik erster Stufe“?

PL erster Stufe Quantifizierung von Elementen ($\exists x P(x)$)

Warum „PrädikatenLogik erster Stufe“?

PL erster Stufe Quantifizierung von Elementen ($\exists x P(x)$)

PL zweiter Stufe Quantifizierung von Prädikaten ($\exists P \forall x P(x)$)

Warum „PrädikatenLogik erster Stufe“?

PL erster Stufe Quantifizierung von Elementen ($\exists x P(x)$)

PL zweiter Stufe Quantifizierung von Prädikaten ($\exists P \forall x P(x)$)

PL dritter Stufe Quantifizierung von Prädikaten über Prädikaten ($\exists R \forall P (R(P))$)

Warum „PrädikatenLogik erster Stufe“?

- PL erster Stufe Quantifizierung von Elementen ($\exists x P(x)$)
- PL zweiter Stufe Quantifizierung von Prädikaten ($\exists P \forall x P(x)$)
- PL dritter Stufe Quantifizierung von Prädikaten über Prädikaten ($\exists R \forall P (R(P))$)
- ...

Warum „PrädikatenLogik erster Stufe“?

- PL erster Stufe Quantifizierung von Elementen ($\exists x P(x)$)
- PL zweiter Stufe Quantifizierung von Prädikaten ($\exists P \forall x P(x)$)
- PL dritter Stufe Quantifizierung von Prädikaten über Prädikaten ($\exists R \forall P (R(P))$)
- ...

Warum „Prädikatenlogik erster Stufe“?

PL erster Stufe Quantifizierung von Elementen ($\exists x P(x)$)

PL zweiter Stufe Quantifizierung von Prädikaten ($\exists P \forall x P(x)$)

PL dritter Stufe Quantifizierung von Prädikaten über Prädikaten ($\exists R \forall P (R(P))$)

...

- PL1 genügt zur Axiomatisierung der rationalen Zahlen ($\leadsto$ Algebra)
- PL2 ist notwendig für reelle Zahlen ($\leadsto$ Analysis)

Definition 4.45 (Dedekind-Vollständigkeit)

Eine Menge M ist **Dedekind-vollständig**, wenn jede nach oben beschränkte nichtleere Teilmenge $S \subseteq M$ eine kleinste obere Schranke hat.



Richard Dedekind
(1831–1916)

Definition 4.45 (Dedekind-Vollständigkeit)

Eine Menge M ist **Dedekind-vollständig**, wenn jede nach oben beschränkte nichtleere Teilmenge $S \subseteq M$ eine kleinste obere Schranke hat.

Beispiel 4.46 (Irrationale Zahl $\sqrt{2}$)

- $S = \{x \mid x \cdot x \leq 2\}$
- Kleinste obere Schranke von S : $\sqrt{2}$



Richard Dedekind
(1831–1916)

Definition 4.45 (Dedekind-Vollständigkeit)

Eine Menge M ist **Dedekind-vollständig**, wenn jede nach oben beschränkte nichtleere Teilmenge $S \subseteq M$ eine kleinste obere Schranke hat.

Beispiel 4.46 (Irrationale Zahl $\sqrt{2}$)

- $S = \{x \mid x \cdot x \leq 2\}$
- Kleinste obere Schranke von S : $\sqrt{2}$
- $\sqrt{2} \notin \mathbb{Q} \rightsquigarrow$ nicht Dedekind-vollständig



Richard Dedekind
(1831–1916)

Definition 4.45 (Dedekind-Vollständigkeit)

Eine Menge M ist **Dedekind-vollständig**, wenn jede nach oben beschränkte nichtleere Teilmenge $S \subseteq M$ eine kleinste obere Schranke hat.

Beispiel 4.46 (Irrationale Zahl $\sqrt{2}$)

- $S = \{x \mid x \cdot x \leq 2\}$
- Kleinste obere Schranke von S : $\sqrt{2}$
- $\sqrt{2} \notin \mathbb{Q} \rightsquigarrow$ nicht Dedekind-vollständig
- $\sqrt{2} \in \mathbb{R} \rightsquigarrow$ Dedekind-vollständig



Richard Dedekind
(1831–1916)

$$\forall S \langle [\exists x S(x) \wedge \exists x \forall y (S(y) \rightarrow y < x)] \rightarrow \\ \exists x [\forall y (S(y) \rightarrow y \leq x) \wedge \forall z (z < x \rightarrow \exists t (z < t \wedge S(t)))] \rangle$$

$$\forall S \langle [\exists x S(x) \wedge \exists x \forall y (S(y) \rightarrow y < x)] \rightarrow \\ \exists x [\forall y (S(y) \rightarrow y \leq x) \wedge \forall z (z < x \rightarrow \exists t (z < t \wedge S(t)))] \rangle$$

„Für jede Menge S gilt: Wenn S mindestens ein Element enthält und ein x existiert, das größer ist als alle Elemente y von S , dann existiert auch ein x , das größer oder gleich jedem Element y von S ist und für jedes z , das kleiner ist als dieses x , gibt es ein t , das größer als z und in S enthalten ist.“

In der Prädikatenlogik kann man mehr ausdrücken als in der Aussagenlogik ...

Beispiel 4.47 (Pinguine)

- Tux ist ein Pinguin.

$P(t)$

In der Prädikatenlogik kann man mehr ausdrücken als in der Aussagenlogik ...

Beispiel 4.47 (Pinguine)

- Tux ist ein Pinguin.

$$P(t)$$

- Kein Pinguin kann fliegen.

$$\forall x(P(x) \rightarrow \neg F(x))$$

In der Prädikatenlogik kann man mehr ausdrücken als in der Aussagenlogik ...

Beispiel 4.47 (Pinguine)

- Tux ist ein Pinguin.
 $P(t)$
- Kein Pinguin kann fliegen.
 $\forall x(P(x) \rightarrow \neg F(x))$
- Alle Pinguine sind Vögel.
 $\forall x(P(x) \rightarrow V(x))$

In der Prädikatenlogik kann man mehr ausdrücken als in der Aussagenlogik ...

Beispiel 4.47 (Pinguine)

- Tux ist ein Pinguin.
 $P(t)$
- Kein Pinguin kann fliegen.
 $\forall x(P(x) \rightarrow \neg F(x))$
- Alle Pinguine sind Vögel.
 $\forall x(P(x) \rightarrow V(x))$
- Manche Vögel können nicht fliegen.
 $\exists x(V(x) \wedge \neg F(x))$

In der Prädikatenlogik kann man mehr ausdrücken als in der Aussagenlogik ...

Beispiel 4.47 (Pinguine)

- Tux ist ein Pinguin.
 $P(t)$
- Kein Pinguin kann fliegen.
 $\forall x(P(x) \rightarrow \neg F(x))$
- Alle Pinguine sind Vögel.
 $\forall x(P(x) \rightarrow V(x))$
- Manche Vögel können nicht fliegen.
 $\exists x(V(x) \wedge \neg F(x))$

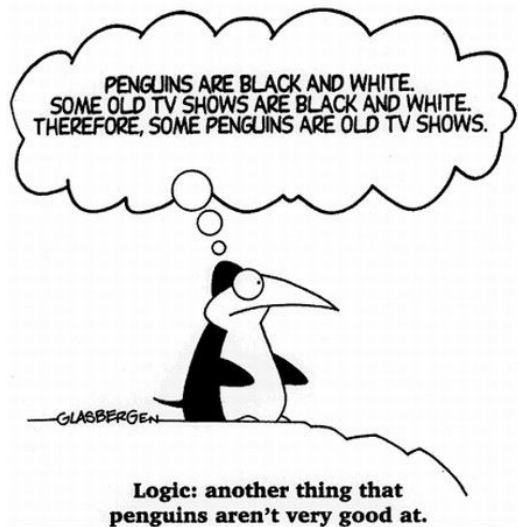
... aber: Schlussfolgern in der Prädikatenlogik ist schwieriger als in der Aussagenlogik.

In der Prädikatenlogik kann man mehr ausdrücken als in der Aussagenlogik ...

Beispiel 4.47 (Pinguine)

- Tux ist ein Pinguin.
 $P(t)$
- Kein Pinguin kann fliegen.
 $\forall x(P(x) \rightarrow \neg F(x))$
- Alle Pinguine sind Vögel.
 $\forall x(P(x) \rightarrow V(x))$
- Manche Vögel können nicht fliegen.
 $\exists x(V(x) \wedge \neg F(x))$

... aber: Schlussfolgern in der Prädikatenlogik ist schwieriger als in der Aussagenlogik.



Eigenname Konstantensymbol (Hans, Stuttgart, Europa)

Eigenname Konstantensymbol (Hans, Stuttgart, Europa)

Substantiv Einstelliges Prädikat (Mensch, Haus, Stadt, Zahl)

Eigenname Konstantensymbol (Hans, Stuttgart, Europa)

Substantiv Einstelliges Prädikat (Mensch, Haus, Stadt, Zahl)

Adjektiv Einstelliges Prädikat (schön, groß, lebendig)

- Eigenname** Konstantensymbol (Hans, Stuttgart, Europa)
- Substantiv** Einstelliges Prädikat (Mensch, Haus, Stadt, Zahl)
- Adjektiv** Einstelliges Prädikat (schön, groß, lebendig)
- Verb** Prädikat oder Funktionssymbol
Stelligkeit abhängig von Anzahl der Objekte

- Eigenname** Konstantensymbol (Hans, Stuttgart, Europa)
- Substantiv** Einstelliges Prädikat (Mensch, Haus, Stadt, Zahl)
- Adjektiv** Einstelliges Prädikat (schön, groß, lebendig)
- Verb** Prädikat oder Funktionssymbol
 - Stelligkeit abhängig von Anzahl der Objekte
 - keins** Einstelliges Prädikat (lebt, existiert)

- Eigenname** Konstantensymbol (Hans, Stuttgart, Europa)
- Substantiv** Einstelliges Prädikat (Mensch, Haus, Stadt, Zahl)
- Adjektiv** Einstelliges Prädikat (schön, groß, lebendig)
- Verb** Prädikat oder Funktionssymbol
 - Stelligkeit abhängig von Anzahl der Objekte
 - keins** Einstelliges Prädikat (lebt, existiert)
 - eins** Zweistelliges Prädikat (kauft, hat, mag)

Eigenname Konstantensymbol (Hans, Stuttgart, Europa)

Substantiv Einstelliges Prädikat (Mensch, Haus, Stadt, Zahl)

Adjektiv Einstelliges Prädikat (schön, groß, lebendig)

Verb Prädikat oder Funktionssymbol

Stelligkeit abhängig von Anzahl der Objekte

keins Einstelliges Prädikat (lebt, existiert)

eins Zweistelliges Prädikat (kauft, hat, mag)

mehrere Mehrstelliges Prädikat (kauft-von, hat-Ausbildungsvertrag-mit)

Eigenname Konstantensymbol (Hans, Stuttgart, Europa)

Substantiv Einstelliges Prädikat (Mensch, Haus, Stadt, Zahl)

Adjektiv Einstelliges Prädikat (schön, groß, lebendig)

Verb Prädikat oder Funktionssymbol

Stelligkeit abhängig von Anzahl der Objekte

keins Einstelliges Prädikat (lebt, existiert)

eins Zweistelliges Prädikat (kauft, hat, mag)

mehrere Mehrstelliges Prädikat (kauft-von, hat-Ausbildungsvertrag-mit)

eindeutig Funktionssymbol (mutter, direkter-nachfolger)

Eigenname Konstantensymbol (Hans, Stuttgart, Europa)

Substantiv Einstelliges Prädikat (Mensch, Haus, Stadt, Zahl)

Adjektiv Einstelliges Prädikat (schön, groß, lebendig)

Verb Prädikat oder Funktionssymbol

Stelligkeit abhängig von Anzahl der Objekte

keins Einstelliges Prädikat (lebt, existiert)

eins Zweistelliges Prädikat (kauft, hat, mag)

mehrere Mehrstelliges Prädikat (kauft-von, hat-Ausbildungsvertrag-mit)

eindeutig Funktionssymbol (mutter, direkter-nachfolger)

und, oder „Frauen **und** Kinder zuerst“ $\rightsquigarrow F(x) \vee K(x) \rightarrow Z(x)$

Eigenname Konstantensymbol (Hans, Stuttgart, Europa)

Substantiv Einstelliges Prädikat (Mensch, Haus, Stadt, Zahl)

Adjektiv Einstelliges Prädikat (schön, groß, lebendig)

Verb Prädikat oder Funktionssymbol

Stelligkeit abhängig von Anzahl der Objekte

keins Einstelliges Prädikat (lebt, existiert)

eins Zweistelliges Prädikat (kauft, hat, mag)

mehrere Mehrstelliges Prädikat (kauft-von, hat-Ausbildungsvertrag-mit)

eindeutig Funktionssymbol (mutter, direkter-nachfolger)

und, oder „Frauen **und** Kinder zuerst“ $\rightsquigarrow F(x) \vee K(x) \rightarrow Z(x)$

wenn, dann „Wenn A, dann B“ $\rightsquigarrow A \rightarrow B$

„**Nur** wenn A, dann B“ $\rightsquigarrow B \rightarrow A$

Eigenname Konstantensymbol (Hans, Stuttgart, Europa)

Substantiv Einstelliges Prädikat (Mensch, Haus, Stadt, Zahl)

Adjektiv Einstelliges Prädikat (schön, groß, lebendig)

Verb Prädikat oder Funktionssymbol

Stelligkeit abhängig von Anzahl der Objekte

keins Einstelliges Prädikat (lebt, existiert)

eins Zweistelliges Prädikat (kauft, hat, mag)

mehrere Mehrstelliges Prädikat (kauft-von, hat-Ausbildungsvertrag-mit)

eindeutig Funktionssymbol (mutter, direkter-nachfolger)

und, oder „Frauen **und** Kinder zuerst“ $\rightsquigarrow F(x) \vee K(x) \rightarrow Z(x)$

wenn, dann „Wenn A, dann B“ $\rightsquigarrow A \rightarrow B$

„**Nur** wenn A, dann B“ $\rightsquigarrow B \rightarrow A$

Eigenname Konstantensymbol (Hans, Stuttgart, Europa)

Substantiv Einstelliges Prädikat (Mensch, Haus, Stadt, Zahl)

Adjektiv Einstelliges Prädikat (schön, groß, lebendig)

Verb Prädikat oder Funktionssymbol

Stelligkeit abhängig von Anzahl der Objekte

keins Einstelliges Prädikat (lebt, existiert)

eins Zweistelliges Prädikat (kauft, hat, mag)

mehrere Mehrstelliges Prädikat (kauft-von, hat-Ausbildungsvertrag-mit)

eindeutig Funktionssymbol (mutter, direkter-nachfolger)

und, oder „Frauen **und** Kinder zuerst“ $\rightsquigarrow F(x) \vee K(x) \rightarrow Z(x)$

wenn, dann „Wenn A, dann B“ $\rightsquigarrow A \rightarrow B$

„**Nur** wenn A, dann B“ $\rightsquigarrow B \rightarrow A$

Häufige Kombinationen:

$\forall$ mit $\rightarrow$ Jeder Mensch ist ein Lebewesen.

$\exists$ mit $\wedge$ Manche Lebewesen können schwimmen.

Übung 4.48

Formalisieren Sie die folgenden Sätze in Prädikatenlogik.

Verwenden Sie hierzu

- die einstelligen Prädikate Student, Professor, Vorlesung,
- die zweistelligen Prädikate Mag, Betreut, Hält,
- das zweistellige Prädikat $=$ (für Gleichheit von Elementen).

- 1 Jeder Professor mag alle Studenten.
- 2 Jeder Student mag (mindestens) einen Professor.
- 3 Es gibt einen Studenten, der genau einen Professor mag.
- 4 Studenten werden nur von Professoren betreut.
- 5 Jeder Professor hält mindestens zwei Vorlesungen.
- 6 Jeder Professor betreut nur Studenten, die ihn mögen.

- 1. Einführung
- 2. Mengen
- 3. Aussagenlogik
- 4. Prädikatenlogik
 - 4.1 Relationen und Funktionen
 - 4.2 Syntax
 - 4.3 Semantik
 - 4.4 Schlussfolgerungsverfahren
- 5. Prolog

1. Einführung

2. Mengen

3. Aussagenlogik

4. Prädikatenlogik

4.1 Relationen und Funktionen

4.2 Syntax

4.3 Semantik

4.4 Schlussfolgerungsverfahren

4.4.1 Resolution

4.4.2 Tableaus

5. Prolog

Wie kann das Resolutionsprinzip auf PL übertragen werden?

Beispiel 4.49 (naive Resolution in PL)

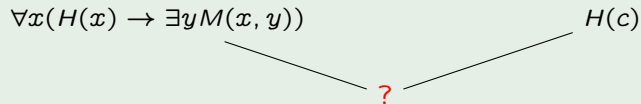
$$\forall x(H(x) \rightarrow \exists y M(x, y))$$

$$H(c)$$

?

Wie kann das Resolutionsprinzip auf PL übertragen werden?

Beispiel 4.49 (naive Resolution in PL)

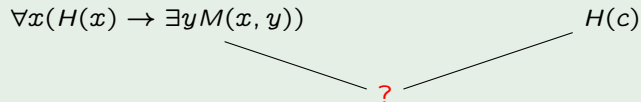


Probleme:

- Interaktion von Quantoren: $\forall x\varphi, \forall x\neg\varphi$ vs. $\exists x\varphi, \exists x\neg\varphi$

Wie kann das Resolutionsprinzip auf PL übertragen werden?

Beispiel 4.49 (naive Resolution in PL)

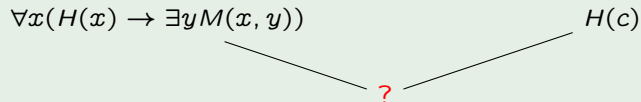


Probleme:

- Interaktion von Quantoren: $\forall x\varphi, \forall x\neg\varphi$ vs. $\exists x\varphi, \exists x\neg\varphi$
- Interaktion von Variablen und Konstanten

Wie kann das Resolutionsprinzip auf PL übertragen werden?

Beispiel 4.49 (naive Resolution in PL)

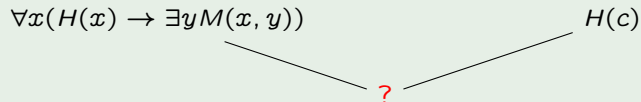


Probleme:

- Interaktion von Quantoren: $\forall x\varphi, \forall x\neg\varphi$ vs. $\exists x\varphi, \exists x\neg\varphi$
- Interaktion von Variablen und Konstanten
- Erzeugung von KNF in Formeln mit Quantoren

Wie kann das Resolutionsprinzip auf PL übertragen werden?

Beispiel 4.49 (naive Resolution in PL)

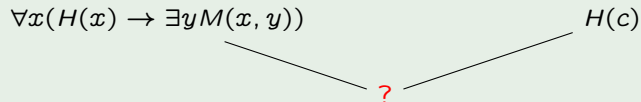


Probleme:

- Interaktion von Quantoren: $\forall x\varphi, \forall x\neg\varphi$ vs. $\exists x\varphi, \exists x\neg\varphi$
- Interaktion von Variablen und Konstanten
- Erzeugung von KNF in Formeln mit Quantoren

Wie kann das Resolutionsprinzip auf PL übertragen werden?

Beispiel 4.49 (naive Resolution in PL)



Probleme:

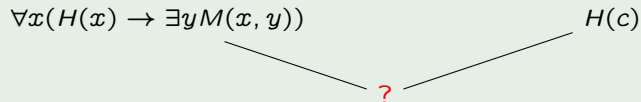
- Interaktion von Quantoren: $\forall x\varphi, \forall x\neg\varphi$ vs. $\exists x\varphi, \exists x\neg\varphi$
- Interaktion von Variablen und Konstanten
- Erzeugung von KNF in Formeln mit Quantoren

Ansatz:

- Elimination des Existenz-Quantors

Wie kann das Resolutionsprinzip auf PL übertragen werden?

Beispiel 4.49 (naive Resolution in PL)



Probleme:

- Interaktion von Quantoren: $\forall x\varphi, \forall x\neg\varphi$ vs. $\exists x\varphi, \exists x\neg\varphi$
- Interaktion von Variablen und Konstanten
- Erzeugung von KNF in Formeln mit Quantoren

Ansatz:

- Elimination des Existenz-Quantors
- Ersetzung von Variablen durch Terme, um Atome gleich zu machen

Schritt 1: Negations-Normalform

Vorgehen wie in AL

1 Elimination von $\rightarrow$ und $\leftrightarrow$

Vorgehen wie in AL

1 Elimination von $\rightarrow$ und $\leftrightarrow$

$$\blacksquare \varphi \leftrightarrow \psi \quad \rightsquigarrow \quad (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$$

Vorgehen wie in AL

1 Elimination von $\rightarrow$ und $\leftrightarrow$

- $\varphi \leftrightarrow \psi \quad \rightsquigarrow \quad (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$
- $\varphi \rightarrow \psi \quad \rightsquigarrow \quad \neg\varphi \vee \psi$

Schritt 1: Negations-Normalform

Vorgehen wie in AL

1 Elimination von $\rightarrow$ und $\leftrightarrow$

$$\blacksquare \varphi \leftrightarrow \psi \quad \rightsquigarrow \quad (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$$

$$\blacksquare \varphi \rightarrow \psi \quad \rightsquigarrow \quad \neg\varphi \vee \psi$$

2 Anwendung der de-Morgan-Gesetze und ihrer Entsprechungen für Quantoren

Vorgehen wie in AL

1 Elimination von $\rightarrow$ und $\leftrightarrow$

$$\blacksquare \varphi \leftrightarrow \psi \quad \rightsquigarrow \quad (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$$

$$\blacksquare \varphi \rightarrow \psi \quad \rightsquigarrow \quad \neg\varphi \vee \psi$$

2 Anwendung der de-Morgan-Gesetze und ihrer Entsprechungen für Quantoren

$$\blacksquare \neg(\varphi \wedge \psi) \quad \rightsquigarrow \quad \neg\varphi \vee \neg\psi$$

Vorgehen wie in AL

1 Elimination von $\rightarrow$ und $\leftrightarrow$

- $\varphi \leftrightarrow \psi \quad \rightsquigarrow \quad (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$
- $\varphi \rightarrow \psi \quad \rightsquigarrow \quad \neg\varphi \vee \psi$

2 Anwendung der de-Morgan-Gesetze und ihrer Entsprechungen für Quantoren

- $\neg(\varphi \wedge \psi) \quad \rightsquigarrow \quad \neg\varphi \vee \neg\psi$
- $\neg(\varphi \vee \psi) \quad \rightsquigarrow \quad \neg\varphi \wedge \neg\psi$

Vorgehen wie in AL

1 Elimination von $\rightarrow$ und $\leftrightarrow$

- $\varphi \leftrightarrow \psi \rightsquigarrow (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$
- $\varphi \rightarrow \psi \rightsquigarrow \neg\varphi \vee \psi$

2 Anwendung der de-Morgan-Gesetze und ihrer Entsprechungen für Quantoren

- $\neg(\varphi \wedge \psi) \rightsquigarrow \neg\varphi \vee \neg\psi$
- $\neg(\varphi \vee \psi) \rightsquigarrow \neg\varphi \wedge \neg\psi$
- $\neg\forall x\varphi \rightsquigarrow \exists x\neg\varphi$

Vorgehen wie in AL

1 Elimination von $\rightarrow$ und $\leftrightarrow$

- $\varphi \leftrightarrow \psi \rightsquigarrow (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$
- $\varphi \rightarrow \psi \rightsquigarrow \neg\varphi \vee \psi$

2 Anwendung der de-Morgan-Gesetze und ihrer Entsprechungen für Quantoren

- $\neg(\varphi \wedge \psi) \rightsquigarrow \neg\varphi \vee \neg\psi$
- $\neg(\varphi \vee \psi) \rightsquigarrow \neg\varphi \wedge \neg\psi$
- $\neg\forall x\varphi \rightsquigarrow \exists x\neg\varphi$
- $\neg\exists x\varphi \rightsquigarrow \forall x\neg\varphi$

Vorgehen wie in AL

1 Elimination von $\rightarrow$ und $\leftrightarrow$

- $\varphi \leftrightarrow \psi \rightsquigarrow (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$
- $\varphi \rightarrow \psi \rightsquigarrow \neg\varphi \vee \psi$

2 Anwendung der de-Morgan-Gesetze und ihrer Entsprechungen für Quantoren

- $\neg(\varphi \wedge \psi) \rightsquigarrow \neg\varphi \vee \neg\psi$
- $\neg(\varphi \vee \psi) \rightsquigarrow \neg\varphi \wedge \neg\psi$
- $\neg\forall x\varphi \rightsquigarrow \exists x\neg\varphi$
- $\neg\exists x\varphi \rightsquigarrow \forall x\neg\varphi$

3 Elimination von doppelter Negation und negierten Boole'schen Konstanten

Vorgehen wie in AL

1 Elimination von $\rightarrow$ und $\leftrightarrow$

- $\varphi \leftrightarrow \psi \rightsquigarrow (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$
- $\varphi \rightarrow \psi \rightsquigarrow \neg\varphi \vee \psi$

2 Anwendung der de-Morgan-Gesetze und ihrer Entsprechungen für Quantoren

- $\neg(\varphi \wedge \psi) \rightsquigarrow \neg\varphi \vee \neg\psi$
- $\neg(\varphi \vee \psi) \rightsquigarrow \neg\varphi \wedge \neg\psi$
- $\neg\forall x\varphi \rightsquigarrow \exists x\neg\varphi$
- $\neg\exists x\varphi \rightsquigarrow \forall x\neg\varphi$

3 Elimination von doppelter Negation und negierten Boole'schen Konstanten

- $\neg\neg\varphi \rightsquigarrow \varphi$

Vorgehen wie in AL

1 Elimination von $\rightarrow$ und $\leftrightarrow$

- $\varphi \leftrightarrow \psi \rightsquigarrow (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$
- $\varphi \rightarrow \psi \rightsquigarrow \neg\varphi \vee \psi$

2 Anwendung der de-Morgan-Gesetze und ihrer Entsprechungen für Quantoren

- $\neg(\varphi \wedge \psi) \rightsquigarrow \neg\varphi \vee \neg\psi$
- $\neg(\varphi \vee \psi) \rightsquigarrow \neg\varphi \wedge \neg\psi$
- $\neg\forall x\varphi \rightsquigarrow \exists x\neg\varphi$
- $\neg\exists x\varphi \rightsquigarrow \forall x\neg\varphi$

3 Elimination von doppelter Negation und negierten Boole'schen Konstanten

- $\neg\neg\varphi \rightsquigarrow \varphi$
- $\neg\mathbf{W} \rightsquigarrow \mathbf{F}$

Vorgehen wie in AL

1 Elimination von $\rightarrow$ und $\leftrightarrow$

- $\varphi \leftrightarrow \psi \rightsquigarrow (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$
- $\varphi \rightarrow \psi \rightsquigarrow \neg\varphi \vee \psi$

2 Anwendung der de-Morgan-Gesetze und **ihrer Entsprechungen für Quantoren**

- $\neg(\varphi \wedge \psi) \rightsquigarrow \neg\varphi \vee \neg\psi$
- $\neg(\varphi \vee \psi) \rightsquigarrow \neg\varphi \wedge \neg\psi$
- $\neg\forall x\varphi \rightsquigarrow \exists x\neg\varphi$
- $\neg\exists x\varphi \rightsquigarrow \forall x\neg\varphi$

3 Elimination von doppelter Negation und negierten Boole'schen Konstanten

- $\neg\neg\varphi \rightsquigarrow \varphi$
- $\neg\mathbf{W} \rightsquigarrow \mathbf{F}$
- $\neg\mathbf{F} \rightsquigarrow \mathbf{W}$

Vorgehen wie in AL

1 Elimination von $\rightarrow$ und $\leftrightarrow$

- $\varphi \leftrightarrow \psi \rightsquigarrow (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$
- $\varphi \rightarrow \psi \rightsquigarrow \neg\varphi \vee \psi$

2 Anwendung der de-Morgan-Gesetze und **ihrer Entsprechungen für Quantoren**

- $\neg(\varphi \wedge \psi) \rightsquigarrow \neg\varphi \vee \neg\psi$
- $\neg(\varphi \vee \psi) \rightsquigarrow \neg\varphi \wedge \neg\psi$
- $\neg\forall x\varphi \rightsquigarrow \exists x\neg\varphi$
- $\neg\exists x\varphi \rightsquigarrow \forall x\neg\varphi$

3 Elimination von doppelter Negation und negierten Boole'schen Konstanten

- $\neg\neg\varphi \rightsquigarrow \varphi$
- $\neg\mathbf{W} \rightsquigarrow \mathbf{F}$
- $\neg\mathbf{F} \rightsquigarrow \mathbf{W}$

Vorgehen wie in AL

1 Elimination von $\rightarrow$ und $\leftrightarrow$

- $\varphi \leftrightarrow \psi \rightsquigarrow (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$
- $\varphi \rightarrow \psi \rightsquigarrow \neg\varphi \vee \psi$

2 Anwendung der de-Morgan-Gesetze und **ihrer Entsprechungen für Quantoren**

- $\neg(\varphi \wedge \psi) \rightsquigarrow \neg\varphi \vee \neg\psi$
- $\neg(\varphi \vee \psi) \rightsquigarrow \neg\varphi \wedge \neg\psi$
- $\neg\forall x\varphi \rightsquigarrow \exists x\neg\varphi$
- $\neg\exists x\varphi \rightsquigarrow \forall x\neg\varphi$

3 Elimination von doppelter Negation und negierten Boole'schen Konstanten

- $\neg\neg\varphi \rightsquigarrow \varphi$
- $\neg\mathbf{W} \rightsquigarrow \mathbf{F}$
- $\neg\mathbf{F} \rightsquigarrow \mathbf{W}$

- **Ergebnis:** Formel nur mit Junktoren $\wedge$, $\vee$ und $\neg$ direkt vor Atomen

Definition 4.50 (Skolem-Form)

Eine prädikatenlogische Formel φ ist in **Skolem-Form (SF)**, wenn sie in NNF ist und keine Existenzquantoren enthält.



Thoralf Skolem
(1887–1963)

Definition 4.50 (Skolem-Form)

Eine prädikatenlogische Formel φ ist in **Skolem-Form (SF)**, wenn sie in NNF ist und keine Existenzquantoren enthält.

Transformation in SF:

Ersetze existentiell quantifizierte Variablen durch Terme



Thoralf Skolem
(1887–1963)

Definition 4.50 (Skolem-Form)

Eine prädikatenlogische Formel φ ist in **Skolem-Form (SF)**, wenn sie in NNF ist und keine Existenzquantoren enthält.

Transformation in SF:

Ersetze existentiell quantifizierte Variablen durch Terme



Thoralf Skolem
(1887–1963)

Beispiel 4.51 (Skolemisierung)

„Es gibt einen Präsidenten.“

$\exists x P(x)$

Definition 4.50 (Skolem-Form)

Eine prädikatenlogische Formel φ ist in **Skolem-Form (SF)**, wenn sie in NNF ist und keine Existenzquantoren enthält.

Transformation in SF:

Ersetze existentiell quantifizierte Variablen durch Terme



Thoralf Skolem
(1887–1963)

Beispiel 4.51 (Skolemisierung)

„Es gibt einen Präsidenten.“

$$\exists x P(x) \rightsquigarrow P(c)$$

c ist neu!

Definition 4.50 (Skolem-Form)

Eine prädikatenlogische Formel φ ist in **Skolem-Form (SF)**, wenn sie in NNF ist und keine Existenzquantoren enthält.

Transformation in SF:

Ersetze existentiell quantifizierte Variablen durch Terme



Thoralf Skolem
(1887–1963)

Beispiel 4.51 (Skolemisierung)

„Es gibt einen Präsidenten.“

$$\exists x P(x) \rightsquigarrow P(c)$$

c ist neu!

„Jeder Mensch hat eine Mutter.“

$$\forall x \exists y (H(x) \rightarrow M(x, y))$$

Definition 4.50 (Skolem-Form)

Eine prädikatenlogische Formel φ ist in **Skolem-Form (SF)**, wenn sie in NNF ist und keine Existenzquantoren enthält.

Transformation in SF:

Ersetze existentiell quantifizierte Variablen durch Terme



Thoralf Skolem
(1887–1963)

Beispiel 4.51 (Skolemisierung)

„Es gibt einen Präsidenten.“

$$\exists x P(x) \rightsquigarrow P(c)$$

c ist neu!

„Jeder Mensch hat eine Mutter.“

$$\forall x \exists y (H(x) \rightarrow M(x, y)) \rightsquigarrow \forall x (H(x) \rightarrow M(x, c))$$

Definition 4.50 (Skolem-Form)

Eine prädikatenlogische Formel φ ist in **Skolem-Form (SF)**, wenn sie in NNF ist und keine Existenzquantoren enthält.

Transformation in SF:

Ersetze existentiell quantifizierte Variablen durch Terme



Thoralf Skolem
(1887–1963)

Beispiel 4.51 (Skolemisierung)

„Es gibt einen Präsidenten.“

$$\exists x P(x) \rightsquigarrow P(c)$$

c ist neu!

„Jeder Mensch hat eine Mutter.“

$$\forall x \exists y (H(x) \rightarrow M(x, y)) \rightsquigarrow \forall x (H(x) \rightarrow M(x, f(x)))$$

Definition 4.50 (Skolem-Form)

Eine prädikatenlogische Formel φ ist in **Skolem-Form (SF)**, wenn sie in NNF ist und keine Existenzquantoren enthält.

Transformation in SF:

Ersetze existentiell quantifizierte Variablen durch Terme



Thoralf Skolem
(1887–1963)

Beispiel 4.51 (Skolemisierung)

„Es gibt einen Präsidenten.“

$$\exists x P(x) \rightsquigarrow P(c)$$

c ist neu!

„Jeder Mensch hat eine Mutter.“

$$\forall x \exists y (H(x) \rightarrow M(x, y)) \rightsquigarrow \forall x (H(x) \rightarrow M(x, f(x)))$$

f ist neu!

Eingabe: Formel φ in NNF

Ausgabe: skolemisierte Formel $\text{sf}(\varphi)$

Eingabe: Formel φ in NNF

Ausgabe: skolemisierte Formel $\text{sf}(\varphi)$

- 1: **while** φ enthält existentiell quantifizierte Variablen **do**
- 2: sei y erste existentiell quantifizierte Variable
- 3: sei $\{\forall x_1, \dots, \forall x_n\}$ die Menge der Allquantoren, in deren Scope $\exists y$ liegt
- 4: erzeuge neues n -stelliges Funktionssymbol f
- 5: ersetze alle Vorkommen von y im Scope von $\exists y$ durch $f(x_1, \dots, x_n)$
- 6: **return** φ

Eingabe: Formel φ in NNF

Ausgabe: skolemisierte Formel $\text{sf}(\varphi)$

- 1: **while** φ enthält existentiell quantifizierte Variablen **do**
- 2: sei y erste existentiell quantifizierte Variable
- 3: sei $\{\forall x_1, \dots, \forall x_n\}$ die Menge der Allquantoren, in deren Scope $\exists y$ liegt
- 4: erzeuge neues n -stelliges Funktionssymbol f
- 5: ersetze alle Vorkommen von y im Scope von $\exists y$ durch $f(x_1, \dots, x_n)$
- 6: **return** φ

Beachte:

- Bearbeite Existenzquantoren von vorne nach hinten

Eingabe: Formel φ in NNF

Ausgabe: skolemisierte Formel $\text{sf}(\varphi)$

- 1: **while** φ enthält existentiell quantifizierte Variablen **do**
- 2: sei y erste existentiell quantifizierte Variable
- 3: sei $\{\forall x_1, \dots, \forall x_n\}$ die Menge der Allquantoren, in deren Scope $\exists y$ liegt
- 4: erzeuge neues n -stelliges Funktionssymbol f
- 5: ersetze alle Vorkommen von y im Scope von $\exists y$ durch $f(x_1, \dots, x_n)$
- 6: **return** φ

Beachte:

- Bearbeite Existenzquantoren von vorne nach hinten
- Das neue Funktionssymbol f heißt **Skolem-Funktionssymbol**

Eingabe: Formel φ in NNF

Ausgabe: skolemisierte Formel $\text{sf}(\varphi)$

- 1: **while** φ enthält existentiell quantifizierte Variablen **do**
- 2: sei y erste existentiell quantifizierte Variable
- 3: sei $\{\forall x_1, \dots, \forall x_n\}$ die Menge der Allquantoren, in deren Scope $\exists y$ liegt
- 4: erzeuge neues n -stelliges Funktionssymbol f
- 5: ersetze alle Vorkommen von y im Scope von $\exists y$ durch $f(x_1, \dots, x_n)$
- 6: **return** φ

Beachte:

- Bearbeite Existenzquantoren von vorne nach hinten
- Das neue Funktionssymbol f heißt **Skolem-Funktionssymbol**
- Existentiell quantifizierte Variablen, die nicht im Scope eines Allquantors stehen, werden durch **Konstanten** ersetzt.

Eingabe: Formel φ in NNF

Ausgabe: skolemisierte Formel $\text{sf}(\varphi)$

- 1: **while** φ enthält existentiell quantifizierte Variablen **do**
- 2: sei y erste existentiell quantifizierte Variable
- 3: sei $\{\forall x_1, \dots, \forall x_n\}$ die Menge der Allquantoren, in deren Scope $\exists y$ liegt
- 4: erzeuge neues n -stelliges Funktionssymbol f
- 5: ersetze alle Vorkommen von y im Scope von $\exists y$ durch $f(x_1, \dots, x_n)$
- 6: **return** φ

Beachte:

- Bearbeite Existenzquantoren von vorne nach hinten
- Das neue Funktionssymbol f heißt **Skolem-Funktionssymbol**
- Existentiell quantifizierte Variablen, die nicht im Scope eines Allquantors stehen, werden durch **Konstanten** ersetzt.
- φ und $\text{sf}(\varphi)$ sind nicht äquivalent, aber **erfüllbarkeitsäquivalent**.

Eingabe: Formel φ in NNF

Ausgabe: skolemisierte Formel $\text{sf}(\varphi)$

- 1: **while** φ enthält existentiell quantifizierte Variablen **do**
- 2: sei y erste existentiell quantifizierte Variable
- 3: sei $\{\forall x_1, \dots, \forall x_n\}$ die Menge der Allquantoren, in deren Scope $\exists y$ liegt
- 4: erzeuge neues n -stelliges Funktionssymbol f
- 5: ersetze alle Vorkommen von y im Scope von $\exists y$ durch $f(x_1, \dots, x_n)$
- 6: **return** φ

Beachte:

- Bearbeite Existenzquantoren von vorne nach hinten
- Das neue Funktionssymbol f heißt **Skolem-Funktionssymbol**
- Existentiell quantifizierte Variablen, die nicht im Scope eines Allquantors stehen, werden durch **Konstanten** ersetzt.
- φ und $\text{sf}(\varphi)$ sind nicht äquivalent, aber **erfüllbarkeitsäquivalent**.
- **Ergebnis:** Formel ohne Existenzquantor

Übung 4.52

Skolemisieren Sie die folgenden Formeln:

1 $\forall x(\exists y R(x, y) \wedge \exists y \neg R(x, y)) \wedge \exists y \exists z \neg R(y, z)$

2 $\exists x \forall y (R(x, y) \vee R(y, x)) \wedge \neg \forall y \neg (R(y, y) \wedge \neg R(y, f(y))) \vee \forall x \neg (\exists y R(x, y) \rightarrow \forall y R(x, y))$

Problem: Unnötig großer Scope von Allquantoren

Beispiel 4.53 (unnötig komplexe Skolem-Funktion)

$$\forall x \exists y (P(x) \wedge Q(y)) \quad \rightsquigarrow \quad \forall x (P(x) \wedge Q(f(x)))$$

Problem: Unnötig großer Scope von Allquantoren

Beispiel 4.53 (unnötig komplexe Skolem-Funktion)

$$\forall x \exists y (P(x) \wedge Q(y)) \quad \rightsquigarrow \quad \forall x (P(x) \wedge Q(f(x)))$$

Da y nicht von x abhängt, könnte y durch eine Konstante ersetzt werden.

Problem: Unnötig großer Scope von Allquantoren

Beispiel 4.53 (unnötig komplexe Skolem-Funktion)

$$\forall x \exists y (P(x) \wedge Q(y)) \quad \rightsquigarrow \quad \forall x (P(x) \wedge Q(f(x)))$$

Da y nicht von x abhängt, könnte y durch eine Konstante ersetzt werden.

Lösung: Zusätzliche Vorverarbeitung für einfachere Skolem-Funktionen

1 Negations-Normalform

Problem: Unnötig großer Scope von Allquantoren

Beispiel 4.53 (unnötig komplexe Skolem-Funktion)

$$\forall x \exists y (P(x) \wedge Q(y)) \quad \rightsquigarrow \quad \forall x (P(x) \wedge Q(f(x)))$$

Da y nicht von x abhängt, könnte y durch eine Konstante ersetzt werden.

Lösung: Zusätzliche Vorverarbeitung für einfachere Skolem-Funktionen

- 1 Negations-Normalform
- 2 Skolem-Form

Problem: Unnötig großer Scope von Allquantoren

Beispiel 4.53 (unnötig komplexe Skolem-Funktion)

$$\forall x \exists y (P(x) \wedge Q(y)) \quad \rightsquigarrow \quad \forall x (P(x) \wedge Q(f(x)))$$

Da y nicht von x abhängt, könnte y durch eine Konstante ersetzt werden.

Lösung: Zusätzliche Vorverarbeitung für einfachere Skolem-Funktionen

- 1 Negations-Normalform

- 2 Skolem-Form

- 1 Miniscoping

Problem: Unnötig großer Scope von Allquantoren

Beispiel 4.53 (unnötig komplexe Skolem-Funktion)

$$\forall x \exists y (P(x) \wedge Q(y)) \quad \rightsquigarrow \quad \forall x (P(x) \wedge Q(f(x)))$$

Da y nicht von x abhängt, könnte y durch eine Konstante ersetzt werden.

Lösung: Zusätzliche Vorverarbeitung für einfachere Skolem-Funktionen

1 Negations-Normalform

2 Skolem-Form

1 Miniscoping

2 Skolemisierung

Ziel: Äquivalenzumformung zur Minimierung des Scopes von Quantoren

Ziel: Äquivalenzumformung zur Minimierung des Scopes von Quantoren

Annahme: x kommt frei in φ und ψ vor, aber nicht in χ

- $\forall x(\varphi \wedge \chi) \rightsquigarrow \forall x\varphi \wedge \chi$
- $\forall x(\varphi \vee \chi) \rightsquigarrow \forall x\varphi \vee \chi$
- $\forall x(\varphi \wedge \psi) \rightsquigarrow \forall x\varphi \wedge \forall x\psi$

Ziel: Äquivalenzumformung zur Minimierung des Scopes von Quantoren

Annahme: x kommt frei in φ und ψ vor, aber nicht in χ

$$\blacksquare \forall x(\varphi \wedge \chi) \rightsquigarrow \forall x\varphi \wedge \chi$$

$$\blacksquare \exists x(\varphi \wedge \chi) \rightsquigarrow \exists x\varphi \wedge \chi$$

$$\blacksquare \forall x(\varphi \vee \chi) \rightsquigarrow \forall x\varphi \vee \chi$$

$$\blacksquare \forall x(\varphi \wedge \psi) \rightsquigarrow \forall x\varphi \wedge \forall x\psi$$

Ziel: Äquivalenzumformung zur Minimierung des Scopes von Quantoren

Annahme: x kommt frei in φ und ψ vor, aber nicht in χ

$$\blacksquare \forall x(\varphi \wedge \chi) \rightsquigarrow \forall x\varphi \wedge \chi$$

$$\blacksquare \exists x(\varphi \wedge \chi) \rightsquigarrow \exists x\varphi \wedge \chi$$

$$\blacksquare \forall x(\varphi \vee \chi) \rightsquigarrow \forall x\varphi \vee \chi$$

$$\blacksquare \exists x(\varphi \vee \chi) \rightsquigarrow \exists x\varphi \vee \chi$$

$$\blacksquare \forall x(\varphi \wedge \psi) \rightsquigarrow \forall x\varphi \wedge \forall x\psi$$

Ziel: Äquivalenzumformung zur Minimierung des Scopes von Quantoren

Annahme: x kommt frei in φ und ψ vor, aber nicht in χ

$$\blacksquare \forall x(\varphi \wedge \chi) \rightsquigarrow \forall x\varphi \wedge \chi$$

$$\blacksquare \forall x(\varphi \vee \chi) \rightsquigarrow \forall x\varphi \vee \chi$$

$$\blacksquare \forall x(\varphi \wedge \psi) \rightsquigarrow \forall x\varphi \wedge \forall x\psi$$

$$\blacksquare \exists x(\varphi \wedge \chi) \rightsquigarrow \exists x\varphi \wedge \chi$$

$$\blacksquare \exists x(\varphi \vee \chi) \rightsquigarrow \exists x\varphi \vee \chi$$

$$\blacksquare \exists x(\varphi \vee \psi) \rightsquigarrow \exists x\varphi \vee \exists x\psi$$

Ziel: Äquivalenzumformung zur Minimierung des Scopes von Quantoren

Annahme: x kommt frei in φ und ψ vor, aber nicht in χ

$$\blacksquare \forall x(\varphi \wedge \chi) \rightsquigarrow \forall x\varphi \wedge \chi$$

$$\blacksquare \forall x(\varphi \vee \chi) \rightsquigarrow \forall x\varphi \vee \chi$$

$$\blacksquare \forall x(\varphi \wedge \psi) \rightsquigarrow \forall x\varphi \wedge \forall x\psi$$

$$\blacksquare \exists x(\varphi \wedge \chi) \rightsquigarrow \exists x\varphi \wedge \chi$$

$$\blacksquare \exists x(\varphi \vee \chi) \rightsquigarrow \exists x\varphi \vee \chi$$

$$\blacksquare \exists x(\varphi \vee \psi) \rightsquigarrow \exists x\varphi \vee \exists x\psi$$

Ziel: Äquivalenzumformung zur Minimierung des Scopes von Quantoren

Annahme: x kommt frei in φ und ψ vor, aber nicht in χ

$$\blacksquare \forall x(\varphi \wedge \chi) \rightsquigarrow \forall x\varphi \wedge \chi$$

$$\blacksquare \exists x(\varphi \wedge \chi) \rightsquigarrow \exists x\varphi \wedge \chi$$

$$\blacksquare \forall x(\varphi \vee \chi) \rightsquigarrow \forall x\varphi \vee \chi$$

$$\blacksquare \exists x(\varphi \vee \chi) \rightsquigarrow \exists x\varphi \vee \chi$$

$$\blacksquare \forall x(\varphi \wedge \psi) \rightsquigarrow \forall x\varphi \wedge \forall x\psi$$

$$\blacksquare \exists x(\varphi \vee \psi) \rightsquigarrow \exists x\varphi \vee \exists x\psi$$

Dabei: Quantoren von innen nach außen bearbeiten.

Ziel: Äquivalenzumformung zur Minimierung des Scopes von Quantoren

Annahme: x kommt frei in φ und ψ vor, aber nicht in χ

$$\blacksquare \forall x(\varphi \wedge \chi) \rightsquigarrow \forall x\varphi \wedge \chi$$

$$\blacksquare \exists x(\varphi \wedge \chi) \rightsquigarrow \exists x\varphi \wedge \chi$$

$$\blacksquare \forall x(\varphi \vee \chi) \rightsquigarrow \forall x\varphi \vee \chi$$

$$\blacksquare \exists x(\varphi \vee \chi) \rightsquigarrow \exists x\varphi \vee \chi$$

$$\blacksquare \forall x(\varphi \wedge \psi) \rightsquigarrow \forall x\varphi \wedge \forall x\psi$$

$$\blacksquare \exists x(\varphi \vee \psi) \rightsquigarrow \exists x\varphi \vee \exists x\psi$$

Dabei: Quantoren von innen nach außen bearbeiten.

Beispiel 4.54 (Skolem-Funktionen mit Miniscoping)

Miniscoping:

$$\forall x\exists y(P(x) \wedge Q(y)) \rightsquigarrow \forall x(P(x) \wedge \exists yQ(y)) \rightsquigarrow \forall xP(x) \wedge \exists yQ(y)$$

Ziel: Äquivalenzumformung zur Minimierung des Scopes von Quantoren

Annahme: x kommt frei in φ und ψ vor, aber nicht in χ

$$\blacksquare \forall x(\varphi \wedge \chi) \rightsquigarrow \forall x\varphi \wedge \chi$$

$$\blacksquare \exists x(\varphi \wedge \chi) \rightsquigarrow \exists x\varphi \wedge \chi$$

$$\blacksquare \forall x(\varphi \vee \chi) \rightsquigarrow \forall x\varphi \vee \chi$$

$$\blacksquare \exists x(\varphi \vee \chi) \rightsquigarrow \exists x\varphi \vee \chi$$

$$\blacksquare \forall x(\varphi \wedge \psi) \rightsquigarrow \forall x\varphi \wedge \forall x\psi$$

$$\blacksquare \exists x(\varphi \vee \psi) \rightsquigarrow \exists x\varphi \vee \exists x\psi$$

Dabei: Quantoren von innen nach außen bearbeiten.

Beispiel 4.54 (Skolem-Funktionen mit Miniscoping)

Miniscoping:

$$\forall x\exists y(P(x) \wedge Q(y)) \rightsquigarrow \forall x(P(x) \wedge \exists yQ(y)) \rightsquigarrow \forall xP(x) \wedge \exists yQ(y)$$

Skolemisierung:

$$\forall xP(x) \wedge \exists yQ(y) \rightsquigarrow \forall xP(x) \wedge Q(c)$$

Übung 4.55

$$\varphi = \forall x \exists y ((P(x) \vee S(x, y) \vee Q(y)) \wedge Q(x))$$

Skolemisieren Sie die Formel φ

- 1 mit dem Standardverfahren;
- 2 mit dem Miniscoping-Verfahren.

Schritt 3: Allquantoren entfernen

- 1 Negations-Normalform
- 2 Skolem-Form
 - 1 Miniscoping
 - 2 Skolemisierung

Schritt 3: Allquantoren entfernen

- 1 Negations-Normalform
- 2 Skolem-Form
 - 1 Miniscoping
 - 2 Skolemisierung
- 3 Allquantoren entfernen

Schritt 3: Allquantoren entfernen

1 Negations-Normalform

2 Skolem-Form

1 Miniscoping

2 Skolemisierung

3 Allquantoren entfernen

- Alle verbleibenden Variablen universell quantifiziert

Schritt 3: Allquantoren entfernen

1 Negations-Normalform

2 Skolem-Form

1 Miniscoping

2 Skolemisierung

3 Allquantoren entfernen

- Alle verbleibenden Variablen universell quantifiziert
- Quantor $\forall$ nicht mehr notwendig

Schritt 3: Allquantoren entfernen

1 Negations-Normalform

2 Skolem-Form

1 Miniscoping

2 Skolemisierung

3 Allquantoren entfernen

- Alle verbleibenden Variablen universell quantifiziert
- Quantor $\forall$ nicht mehr notwendig
- Zuvor: Mehrfach quantifizierte Variablen umbenennen

1 Negations-Normalform

2 Skolem-Form

1 Miniscoping

2 Skolemisierung

3 Allquantoren entfernen

- Alle verbleibenden Variablen universell quantifiziert
- Quantor $\forall$ nicht mehr notwendig
- Zuvor: Mehrfach quantifizierte Variablen umbenennen
- **Ergebnis:** Formel ohne Quantoren

1 Negations-Normalform

2 Skolem-Form

1 Miniscoping

2 Skolemisierung

3 Allquantoren entfernen

- Alle verbleibenden Variablen universell quantifiziert
- Quantor $\forall$ nicht mehr notwendig
- Zuvor: Mehrfach quantifizierte Variablen umbenennen
- **Ergebnis:** Formel ohne Quantoren

Schritt 3: Allquantoren entfernen

1 Negations-Normalform

2 Skolem-Form

1 Miniscoping

2 Skolemisierung

3 Allquantoren entfernen

- Alle verbleibenden Variablen universell quantifiziert
- Quantor $\forall$ nicht mehr notwendig
- Zuvor: Mehrfach quantifizierte Variablen umbenennen
- **Ergebnis:** Formel ohne Quantoren

Beispiel 4.56 (Entfernen des Allquantors)

$$\forall x(P(x, c) \vee Q(x)) \vee \forall xP(x, x)$$

Schritt 3: Allquantoren entfernen

1 Negations-Normalform

2 Skolem-Form

1 Miniscoping

2 Skolemisierung

3 Allquantoren entfernen

- Alle verbleibenden Variablen universell quantifiziert
- Quantor $\forall$ nicht mehr notwendig
- Zuvor: Mehrfach quantifizierte Variablen umbenennen
- **Ergebnis:** Formel ohne Quantoren

Beispiel 4.56 (Entfernen des Allquantors)

$$\begin{aligned} & \forall x (P(x, c) \vee Q(x)) \vee \forall x P(x, x) \\ \rightsquigarrow & \forall x (P(x, c) \vee Q(x)) \vee \forall y P(y, y) \end{aligned}$$

Schritt 3: Allquantoren entfernen

1 Negations-Normalform

2 Skolem-Form

1 Miniscoping

2 Skolemisierung

3 Allquantoren entfernen

- Alle verbleibenden Variablen universell quantifiziert
- Quantor $\forall$ nicht mehr notwendig
- Zuvor: Mehrfach quantifizierte Variablen umbenennen
- **Ergebnis:** Formel ohne Quantoren

Beispiel 4.56 (Entfernen des Allquantors)

$$\begin{aligned} & \forall x (P(x, c) \vee Q(x)) \vee \forall x P(x, x) \\ \rightsquigarrow & \forall x (P(x, c) \vee Q(x)) \vee \forall y P(y, y) \\ \rightsquigarrow & P(x, c) \vee Q(x) \vee P(y, y) \end{aligned}$$

Schritt 3: Allquantoren entfernen

1 Negations-Normalform

2 Skolem-Form

1 Miniscoping

2 Skolemisierung

3 Allquantoren entfernen

- Alle verbleibenden Variablen universell quantifiziert
- Quantor $\forall$ nicht mehr notwendig
- Zuvor: Mehrfach quantifizierte Variablen umbenennen
- **Ergebnis:** Formel ohne Quantoren

Beispiel 4.56 (Entfernen des Allquantors)

$$\begin{aligned} & \forall x (P(x, c) \vee Q(x)) \vee \forall x P(x, x) \\ \rightsquigarrow & \forall x (P(x, c) \vee Q(x)) \vee \forall y P(y, y) \\ \rightsquigarrow & P(x, c) \vee Q(x) \vee P(y, y) \\ \text{nicht} & P(x, c) \vee Q(x) \vee P(x, x) \end{aligned}$$

- 1 Negations-Normalform
- 2 Skolem-Form
 - 1 Miniscoping
 - 2 Skolemisierung
- 3 Entfernung des Allquantors
 - 1 Variablen-Umbenennung
 - 2 $\forall$ weglassen
- 4 Konjunktive Normalform

- 1 Negations-Normalform
- 2 Skolem-Form
 - 1 Miniscoping
 - 2 Skolemisierung
- 3 Entfernung des Allquantors
 - 1 Variablen-Umbenennung
 - 2 $\forall$ weglassen
- 4 Konjunktive Normalform
 - Wie in AL: Anwendung des Distributivgesetzes

- 1 Negations-Normalform
- 2 Skolem-Form
 - 1 Miniscoping
 - 2 Skolemisierung
- 3 Entfernung des Allquantors
 - 1 Variablen-Umbenennung
 - 2 $\forall$ weglassen
- 4 Konjunktive Normalform
 - Wie in AL: Anwendung des Distributivgesetzes
 - Atome statt Aussagenvariablen

- 1 Negations-Normalform
- 2 Skolem-Form
 - 1 Miniscoping
 - 2 Skolemisierung
- 3 Entfernung des Allquantors
 - 1 Variablen-Umbenennung
 - 2 $\forall$ weglassen
- 4 Konjunktive Normalform
 - Wie in AL: Anwendung des Distributivgesetzes
 - Atome statt Aussagenvariablen
 - Ergebnis: Konjunktion von Disjunktionen

Beispiel 4.57 (PL-Resolution)

Ausgangsformel $\exists z H(z) \wedge \forall x (H(x) \rightarrow \exists y M(x, y))$

Beispiel 4.57 (PL-Resolution)

Ausgangsformel $\exists z H(z) \wedge \forall x (H(x) \rightarrow \exists y M(x, y))$

NNF $\exists z H(z) \wedge \forall x (\neg H(x) \vee \exists y M(x, y))$

→ eliminiert

Beispiel 4.57 (PL-Resolution)

Ausgangsformel $\exists z H(z) \wedge \forall x (H(x) \rightarrow \exists y M(x, y))$

NNF $\exists z H(z) \wedge \forall x (\neg H(x) \vee \exists y M(x, y))$

$\rightarrow$ eliminiert

Skolem-Form $H(c) \wedge \forall x (\neg H(x) \vee M(x, f(x)))$

$\exists$ eliminiert

Beispiel 4.57 (PL-Resolution)

Ausgangsformel $\exists z H(z) \wedge \forall x (H(x) \rightarrow \exists y M(x, y))$

NNF $\exists z H(z) \wedge \forall x (\neg H(x) \vee \exists y M(x, y))$

$\rightarrow$ eliminiert

Skolem-Form $H(c) \wedge \forall x (\neg H(x) \vee M(x, f(x)))$

$\exists$ eliminiert

KNF $H(c) \wedge (\neg H(x) \vee M(x, f(x)))$

PL-Atome statt Aussagenvariablen

Beispiel 4.57 (PL-Resolution)

Ausgangsformel $\exists z H(z) \wedge \forall x (H(x) \rightarrow \exists y M(x, y))$

NNF $\exists z H(z) \wedge \forall x (\neg H(x) \vee \exists y M(x, y))$

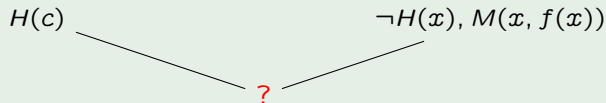
$\rightarrow$ eliminiert

Skolem-Form $H(c) \wedge \forall x (\neg H(x) \vee M(x, f(x)))$

$\exists$ eliminiert

KNF $H(c) \wedge (\neg H(x) \vee M(x, f(x)))$

PL-Atome statt Aussagenvariablen



Beispiel 4.57 (PL-Resolution)

Ausgangsformel $\exists z H(z) \wedge \forall x (H(x) \rightarrow \exists y M(x, y))$

NNF $\exists z H(z) \wedge \forall x (\neg H(x) \vee \exists y M(x, y))$

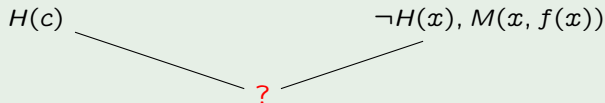
$\rightarrow$ eliminiert

Skolem-Form $H(c) \wedge \forall x (\neg H(x) \vee M(x, f(x)))$

$\exists$ eliminiert

KNF $H(c) \wedge (\neg H(x) \vee M(x, f(x)))$

PL-Atome statt Aussagenvariablen



- Finde Variablenbelegung, die zwei Atome syntaktisch gleich macht

Beispiel 4.57 (PL-Resolution)

Ausgangsformel $\exists z H(z) \wedge \forall x (H(x) \rightarrow \exists y M(x, y))$

NNF $\exists z H(z) \wedge \forall x (\neg H(x) \vee \exists y M(x, y))$

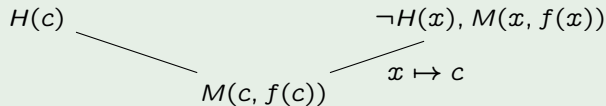
→ eliminiert

Skolem-Form $H(c) \wedge \forall x(\neg H(x) \vee M(x, f(x)))$

$\exists$ eliminiert

$$\text{KNF } H(c) \wedge (\neg H(x) \vee M(x, f(x)))$$

PL-Atome statt Aussagenvariablen



- Finde Variablenbelegung, die zwei Atome syntaktisch gleich macht
- Ersetze Variablen in **gesamter** Klausel

Beispiel 4.57 (PL-Resolution)

Ausgangsformel $\exists z H(z) \wedge \forall x (H(x) \rightarrow \exists y M(x, y))$

NNF $\exists z H(z) \wedge \forall x (\neg H(x) \vee \exists y M(x, y))$

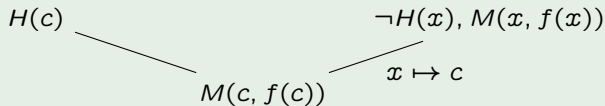
$\rightarrow$ eliminiert

Skolem-Form $H(c) \wedge \forall x (\neg H(x) \vee M(x, f(x)))$

$\exists$ eliminiert

KNF $H(c) \wedge (\neg H(x) \vee M(x, f(x)))$

PL-Atome statt Aussagenvariablen



- Finde Variablenbelegung, die zwei Atome syntaktisch gleich macht
- Ersetze Variablen in **gesamter** Klausel
- Erzeuge Resolvente

Definition 4.58 (Substitution)

Eine **Substitution** ist eine Funktion, die Variablen auf Terme abbildet.

Definition 4.58 (Substitution)

Eine **Substitution** ist eine Funktion, die Variablen auf Terme abbildet.

Beispiel 4.59 (Substitution)

Die Substitution

$$\sigma = \{x \mapsto f(z), y \mapsto g(c, d)\}$$

bildet x auf $f(z)$ und y auf $g(c, d)$ ab.

Die Anwendung einer Substitution auf einen Term oder eine Formel ersetzt alle Variablen des Definitionsbereichs durch ihre Funktionswerte:

$$\sigma(g(x, f(y))) = g(f(z), f(g(c, d)))$$

Definition 4.60 (Unifikator)

Ein **Unifikator** ist eine Substitution, die zwei Formeln auf dieselbe Formel abbildet.

Definition 4.60 (Unifikator)

Ein **Unifikator** ist eine Substitution, die zwei Formeln auf dieselbe Formel abbildet.

Beispiel 4.61 (Unifikator von $R(a, y)$ und $R(x, f(a))$)

$$\varphi = R(a, y) \quad \psi = R(x, f(a)) \quad \sigma = \{x \mapsto a, y \mapsto f(a)\}$$

Definition 4.60 (Unifikator)

Ein **Unifikator** ist eine Substitution, die zwei Formeln auf dieselbe Formel abbildet.

Beispiel 4.61 (Unifikator von $R(a, y)$ und $R(x, f(a))$)

$$\begin{aligned} \varphi &= R(a, y) & \psi &= R(x, f(a)) & \sigma &= \{x \mapsto a, y \mapsto f(a)\} \\ \sigma(\varphi) &= R(a, f(a)) = \sigma(\psi) \end{aligned}$$

Definition 4.60 (Unifikator)

Ein **Unifikator** ist eine Substitution, die zwei Formeln auf dieselbe Formel abbildet.

Beispiel 4.61 (Unifikator von $R(a, y)$ und $R(x, f(a))$)

$$\begin{aligned} \varphi &= R(a, y) & \psi &= R(x, f(a)) & \sigma &= \{x \mapsto a, y \mapsto f(a)\} \\ \sigma(\varphi) &= R(a, f(a)) = \sigma(\psi) \end{aligned}$$

Resolution von C_1 und C_2 ist möglich, wenn es Atome φ, ψ und Substitution σ gibt mit

- $\varphi \in C_1$ und $\neg\psi \in C_2$
- σ ist Unifikator von φ und ψ .

Beispiel 4.62 (gleiche Variable in mehreren Klauseln)

Problem Unifikator für $P(x, c)$ und $P(d, x)$ wird nicht gefunden

Beispiel 4.62 (gleiche Variable in mehreren Klauseln)

Problem Unifikator für $P(x, c)$ und $P(d, x)$ wird nicht gefunden

Lösung Disjunkte Umbenennung der Variablen $P(x, c), P(d, y), \sigma = \{x \mapsto d, y \mapsto c\}$

Beispiel 4.62 (gleiche Variable in mehreren Klauseln)

Problem Unifikator für $P(x, c)$ und $P(d, x)$ wird nicht gefunden

Lösung Disjunkte Umbenennung der Variablen $P(x, c), P(d, y), \sigma = \{x \mapsto d, y \mapsto c\}$

Beispiel 4.63 (Abbildung von x auf $f(x)$)

Problem Unifikation von x und $f(x)$ resultiert in $x \mapsto f(x)$

Beispiel 4.62 (gleiche Variable in mehreren Klauseln)

Problem Unifikator für $P(x, c)$ und $P(d, x)$ wird nicht gefunden

Lösung Disjunkte Umbenennung der Variablen $P(x, c), P(d, y), \sigma = \{x \mapsto d, y \mapsto c\}$

Beispiel 4.63 (Abbildung von x auf $f(x)$)

Problem Unifikation von x und $f(x)$ resultiert in $x \mapsto f(x)$

Lösung Occurs-Check: x kann nicht auf Term abgebildet werden, der x enthält

Übung 4.64

Seien

- u, w, x, y, z Variablen,
- c, d Konstantensymbole,
- f ein einstelliges und g ein zweistelliges Funktionssymbol,
- N und P einstellige, R ein zweistelliges und S ein dreistelliges Prädikat.

Versuchen Sie, Unifikatoren für die folgenden Paare von Atomen zu finden.

- 1 $P(x)$ und $P(f(g(y, z)))$
- 2 $P(x)$ und $N(f(y))$
- 3 $R(x, f(x))$ und $R(f(y), z)$
- 4 $R(x, f(x))$ und $R(f(y), y)$
- 5 $R(x, f(x))$ und $R(f(c), d)$
- 6 $S(x, f(g(x, y)), g(x, f(d)))$ und $S(g(c, d), f(z), g(g(c, u), w))$

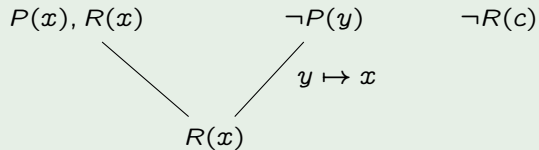
Beispiel 4.65 (Mehrere mögliche Unifikatoren)

$P(x), R(x)$

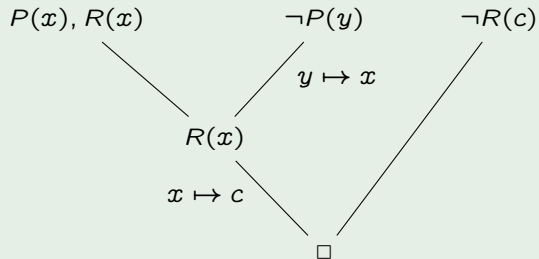
$\neg P(y)$

$\neg R(c)$

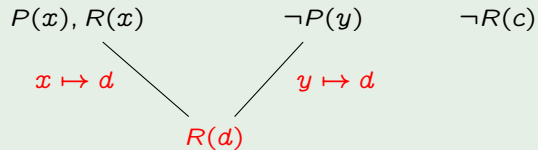
Beispiel 4.65 (Mehrere mögliche Unifikatoren)



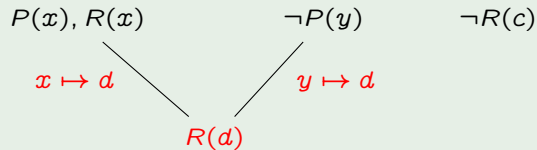
Beispiel 4.65 (Mehrere mögliche Unifikatoren)



Beispiel 4.65 (Mehrere mögliche Unifikatoren)

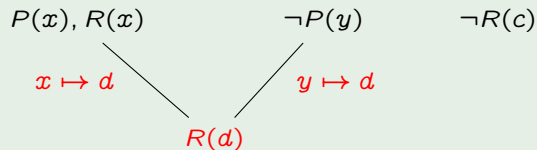


Beispiel 4.65 (Mehrere mögliche Unifikatoren)



Problem manche Unifikatoren verhindern Resolution

Beispiel 4.65 (Mehrere mögliche Unifikatoren)



Problem manche Unifikatoren verhindern Resolution

Lösung **Allgemeinster** Unifikator (MGU): ersetze so wenig wie möglich

Eingabe: Atome φ, ψ

Ausgabe: $\text{au}(\varphi, \psi)$, wenn φ und ψ unifizierbar sind; „nicht unifizierbar“, sonst

Eingabe: Atome φ, ψ

Ausgabe: $\text{au}(\varphi, \psi)$, wenn φ und ψ unifizierbar sind; „nicht unifizierbar“, sonst

1: $\sigma := \{\}$

Eingabe: Atome φ, ψ

Ausgabe: $\text{au}(\varphi, \psi)$, wenn φ und ψ unifizierbar sind; „nicht unifizierbar“, sonst

- 1: $\sigma := \{\}$
- 2: **while** $\sigma(\varphi) \neq \sigma(\psi)$ **do**
- 3: sei i die erste Position, an der sich $\sigma(\varphi)$ und $\sigma(\psi)$ unterscheiden

Eingabe: Atome φ, ψ

Ausgabe: $\text{au}(\varphi, \psi)$, wenn φ und ψ unifizierbar sind; „nicht unifizierbar“, sonst

```
1:  $\sigma := \{\}$ 
2: while  $\sigma(\varphi) \neq \sigma(\psi)$  do
3:     sei  $i$  die erste Position, an der sich  $\sigma(\varphi)$  und  $\sigma(\psi)$  unterscheiden
4:     if  $\sigma(\varphi)|_i$  oder  $\sigma(\psi)|_i$  ist Prädikat then
5:         return „nicht unifizierbar“
```

Eingabe: Atome φ, ψ

Ausgabe: $\text{au}(\varphi, \psi)$, wenn φ und ψ unifizierbar sind; „nicht unifizierbar“, sonst

```
1:  $\sigma := \{\}$ 
2: while  $\sigma(\varphi) \neq \sigma(\psi)$  do
3:   sei  $i$  die erste Position, an der sich  $\sigma(\varphi)$  und  $\sigma(\psi)$  unterscheiden
4:   if  $\sigma(\varphi)|_i$  oder  $\sigma(\psi)|_i$  ist Prädikat then
5:     return „nicht unifizierbar“
6:   else if weder  $\sigma(\varphi)|_i$  noch  $\sigma(\psi)|_i$  ist Variable then
7:     return „nicht unifizierbar“
```


Eingabe: Atome φ, ψ

Ausgabe: $\text{au}(\varphi, \psi)$, wenn φ und ψ unifizierbar sind; „nicht unifizierbar“, sonst

```
1:  $\sigma := \{\}$ 
2: while  $\sigma(\varphi) \neq \sigma(\psi)$  do
3:     sei  $i$  die erste Position, an der sich  $\sigma(\varphi)$  und  $\sigma(\psi)$  unterscheiden
4:     if  $\sigma(\varphi)|_i$  oder  $\sigma(\psi)|_i$  ist Prädikat then
5:         return „nicht unifizierbar“
6:     else if weder  $\sigma(\varphi)|_i$  noch  $\sigma(\psi)|_i$  ist Variable then
7:         return „nicht unifizierbar“
8:     else
9:         sei  $x$  die Variable,  $t$  der andere Term
```

Eingabe: Atome φ, ψ

Ausgabe: $\text{au}(\varphi, \psi)$, wenn φ und ψ unifizierbar sind; „nicht unifizierbar“, sonst

```
1:  $\sigma := \{\}$ 
2: while  $\sigma(\varphi) \neq \sigma(\psi)$  do
3:   sei  $i$  die erste Position, an der sich  $\sigma(\varphi)$  und  $\sigma(\psi)$  unterscheiden
4:   if  $\sigma(\varphi)|_i$  oder  $\sigma(\psi)|_i$  ist Prädikat then
5:     return „nicht unifizierbar“
6:   else if weder  $\sigma(\varphi)|_i$  noch  $\sigma(\psi)|_i$  ist Variable then
7:     return „nicht unifizierbar“
8:   else
9:     sei  $x$  die Variable,  $t$  der andere Term
10:    if  $x$  ist echter Subterm von  $t$  then
11:      return „nicht unifizierbar“
```

Eingabe: Atome φ, ψ

Ausgabe: $\text{au}(\varphi, \psi)$, wenn φ und ψ unifizierbar sind; „nicht unifizierbar“, sonst

```
1:  $\sigma := \{\}$ 
2: while  $\sigma(\varphi) \neq \sigma(\psi)$  do
3:   sei  $i$  die erste Position, an der sich  $\sigma(\varphi)$  und  $\sigma(\psi)$  unterscheiden
4:   if  $\sigma(\varphi)|_i$  oder  $\sigma(\psi)|_i$  ist Prädikat then
5:     return „nicht unifizierbar“
6:   else if weder  $\sigma(\varphi)|_i$  noch  $\sigma(\psi)|_i$  ist Variable then
7:     return „nicht unifizierbar“
8:   else
9:     sei  $x$  die Variable,  $t$  der andere Term
10:    if  $x$  ist echter Subterm von  $t$  then
11:      return „nicht unifizierbar“
12:    else
13:      Ersetze jedes Vorkommen von  $x$  in  $\sigma$  durch  $t$ 
14:       $\sigma := \sigma \cup \{x \mapsto t\}$ 
```

Eingabe: Atome φ, ψ

Ausgabe: $\text{au}(\varphi, \psi)$, wenn φ und ψ unifizierbar sind; „nicht unifizierbar“, sonst

```
1:  $\sigma := \{\}$ 
2: while  $\sigma(\varphi) \neq \sigma(\psi)$  do
3:   sei  $i$  die erste Position, an der sich  $\sigma(\varphi)$  und  $\sigma(\psi)$  unterscheiden
4:   if  $\sigma(\varphi)|_i$  oder  $\sigma(\psi)|_i$  ist Prädikat then
5:     return „nicht unifizierbar“
6:   else if weder  $\sigma(\varphi)|_i$  noch  $\sigma(\psi)|_i$  ist Variable then
7:     return „nicht unifizierbar“
8:   else
9:     sei  $x$  die Variable,  $t$  der andere Term
10:    if  $x$  ist echter Subterm von  $t$  then
11:      return „nicht unifizierbar“
12:    else
13:      Ersetze jedes Vorkommen von  $x$  in  $\sigma$  durch  $t$ 
14:       $\sigma := \sigma \cup \{x \mapsto t\}$ 
15: return  $\sigma$ 
```

Beispiel 4.66

$$\varphi = R(x, z, g(x, f(y)))$$

$$\psi = R(f(c), x, g(y, f(g(x, y))))$$

Beispiel 4.66

$$\varphi = R(x, z, g(x, f(y)))$$

$$\psi = R(f(c), \textcolor{red}{x}, g(\textcolor{red}{y}, f(g(\textcolor{red}{x}, \textcolor{red}{y}))))$$

Beispiel 4.66

$$\varphi = R(x, z, g(x, f(y)))$$

$$\psi = R(f(c), u, g(\textcolor{red}{y}, f(g(u, \textcolor{red}{y}))))$$

Beispiel 4.66

$$\varphi = R(x, z, g(x, f(y)))$$

$$\psi = R(f(c), u, g(w, f(g(u, w))))$$

Beispiel 4.66

$$\varphi = R(x, z, g(x, f(y)))$$

$$\psi = R(f(c), u, g(w, f(g(u, w))))$$

Beispiel 4.66

$$\varphi = R(\mathbf{x}, z, g(x, f(y)))$$

$$\psi = R(\mathbf{f(c)}, u, g(w, f(g(u, w))))$$

$$x \mapsto f(c) \quad \text{Occurs-Check}$$

Beispiel 4.66

$$\varphi = R(\textcolor{red}{x}, z, g(\textcolor{red}{x}, f(y)))$$

$$\psi = R(f(c), u, g(w, f(g(u, w))))$$

$$x \mapsto f(c) \quad \text{Occurs-Check}$$

Beispiel 4.66

$$\varphi = R(f(c), z, g(f(c), f(y)))$$

$$\psi = R(f(c), u, g(w, f(g(u, w))))$$

$$x \mapsto f(c) \quad \text{Occurs-Check}$$

Beispiel 4.66

$$\varphi = R(f(c), \textcolor{red}{z}, g(f(c), f(y)))$$

$$\psi = R(f(c), \textcolor{red}{u}, g(w, f(g(\textcolor{red}{u}, w))))$$

x	$\mapsto$	$f(c)$	Occurs-Check allgemeinster Unifikator
u	$\mapsto$	z	

Beispiel 4.66

$$\varphi = R(f(c), z, g(f(c), f(y)))$$

$$\psi = R(f(c), z, g(w, f(g(z, w))))$$

x	$\mapsto$	$f(c)$	Occurs-Check allgemeinster Unifikator
u	$\mapsto$	z	

Beispiel 4.66

$$\varphi = R(f(c), z, \textcolor{red}{g}(f(c), f(y)))$$

$$\psi = R(f(c), z, \textcolor{red}{g}(w, f(g(z, w))))$$

x	$\mapsto$	$f(c)$	Occurs-Check allgemeinster Unifikator
u	$\mapsto$	z	

Beispiel 4.66

$$\varphi = R(f(c), z, g(\textcolor{red}{f}(c), f(y)))$$

$$\psi = R(f(c), z, g(\textcolor{red}{w}, f(g(z, \textcolor{red}{w}))))$$

x	$\mapsto$	$f(c)$	Occurs-Check allgemeinster Unifikator
u	$\mapsto$	z	
w	$\mapsto$	$f(c)$	

Beispiel 4.66

$$\varphi = R(f(c), z, g(f(c), f(y)))$$

$$\psi = R(f(c), z, g(f(c), f(g(z, f(c)))))$$

x	$\mapsto$	$f(c)$	Occurs-Check allgemeinster Unifikator
u	$\mapsto$	z	
w	$\mapsto$	$f(c)$	

Beispiel 4.66

$$\varphi = R(f(c), z, g(f(c), f(y)))$$

$$\psi = R(f(c), z, g(f(c), f(g(z, f(c)))))$$

x	$\mapsto$	$f(c)$	Occurs-Check allgemeinster Unifikator
u	$\mapsto$	z	
w	$\mapsto$	$f(c)$	

Beispiel 4.66

$$\varphi = R(f(c), z, g(f(c), f(y)))$$

$$\psi = R(f(c), z, g(f(c), f(g(z, f(c)))))$$

x	$\mapsto$	$f(c)$	Occurs-Check allgemeinster Unifikator
u	$\mapsto$	z	
w	$\mapsto$	$f(c)$	
y	$\mapsto$	$g(z, f(c))$	

Beispiel 4.66

$$\varphi = R(f(c), z, g(f(c), f(g(z, f(c)))))$$

$$\psi = R(f(c), z, g(f(c), f(g(z, f(c)))))$$

x	$\mapsto$	$f(c)$	Occurs-Check allgemeinster Unifikator
u	$\mapsto$	z	
w	$\mapsto$	$f(c)$	
y	$\mapsto$	$g(z, f(c))$	

Übung 4.67

Seien

- x, y, z Variablen,
- c, d Konstanten,
- f einstelliges, g ein zweistelliges und h ein dreistelliges Funktionssymbol,
- S und T dreistellige Prädikate.

Wenden Sie den Unifikations-Algorithmus an, um allgemeinste Unifikatoren für die folgenden Paare von Termen zu finden. Beachten Sie die disjunkte Umbenennung der Variablen.

1 $\varphi_1 = S(x, f(y), g(z, d))$ und $\psi_1 = S(c, f(x), g(f(z), z))$

2 $\varphi_2 = T(x, f(x), h(f(y), z, z))$ und $\psi_2 = T(c, f(y), h(z, f(x), c))$

Prämissen:

- 1 Bob kauft einen Kürbis.

Prämissen:

- 1 Bob kauft einen Kürbis.

$$\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))$$

Prämissen:

- 1 Bob kauft einen Kürbis.

$$\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))$$

- 2 Wer einen Kürbis kauft, isst ihn oder schnitzt ihn.

Prämissen:

- 1 Bob kauft einen Kürbis.

$$\exists x(\text{Kauft}(\text{b}, x) \wedge \text{Kürbis}(x))$$

- 2 Wer einen Kürbis kauft, isst ihn oder schnitzt ihn.

$$\forall x \forall y (\text{Kauft}(x, y) \wedge \text{Kürbis}(y) \rightarrow \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))$$

Prämissen:

- 1 Bob kauft einen Kürbis.

$$\exists x(\text{Kauft}(\text{b}, x) \wedge \text{Kürbis}(x))$$

- 2 Wer einen Kürbis kauft, isst ihn oder schnitzt ihn.

$$\forall x \forall y (\text{Kauft}(x, y) \wedge \text{Kürbis}(y) \rightarrow \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))$$

- 3 Kinder essen keine Kürbisse.

Prämissen:

- 1 Bob kauft einen Kürbis.

$$\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))$$

- 2 Wer einen Kürbis kauft, isst ihn oder schnitzt ihn.

$$\forall x \forall y (\text{Kauft}(x, y) \wedge \text{Kürbis}(y) \rightarrow \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))$$

- 3 Kinder essen keine Kürbisse.

$$\forall x (\text{Kind}(x) \rightarrow \forall y (\text{Kürbis}(y) \rightarrow \neg \text{Isst}(x, y)))$$

Prämissen:

- 1 Bob kauft einen Kürbis.

$$\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))$$

- 2 Wer einen Kürbis kauft, isst ihn oder schnitzt ihn.

$$\forall x \forall y (\text{Kauft}(x, y) \wedge \text{Kürbis}(y) \rightarrow \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))$$

- 3 Kinder essen keine Kürbisse.

$$\forall x (\text{Kind}(x) \rightarrow \forall y (\text{Kürbis}(y) \rightarrow \neg \text{Isst}(x, y)))$$

Prämissen:

- 1 Bob kauft einen Kürbis.

$$\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))$$

- 2 Wer einen Kürbis kauft, isst ihn oder schnitzt ihn.

$$\forall x \forall y (\text{Kauft}(x, y) \wedge \text{Kürbis}(y) \rightarrow \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))$$

- 3 Kinder essen keine Kürbisse.

$$\forall x (\text{Kind}(x) \rightarrow \forall y (\text{Kürbis}(y) \rightarrow \neg \text{Isst}(x, y)))$$

Konklusion:

- 4 Wenn Bob ein Kind ist, schnitzt er etwas.

$$\text{Kind}(b) \rightarrow \exists x \text{Schnitzt}(b, x)$$

0 Konjunktion von Prämissen und Negation der Konklusion

$$\begin{aligned} & (\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))) \quad \wedge \\ & (\forall x \forall y(\text{Kauft}(x, y) \wedge \text{Kürbis}(y) \rightarrow \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))) \quad \wedge \\ & (\forall x(\text{Kind}(x) \rightarrow \forall y(\text{Kürbis}(y) \rightarrow \neg \text{Isst}(x, y)))) \quad \wedge \\ & \neg(\text{Kind}(b) \rightarrow \exists x \text{Schnitzt}(b, x)) \end{aligned}$$

0 Konjunktion von Prämissen und Negation der Konklusion

1 Negations-Normalform

1 Ersetzung von $\varphi \rightarrow \psi$ durch $\neg\varphi \vee \psi$

$$\begin{aligned} & (\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))) \quad \wedge \\ & (\forall x \forall y(\text{Kauft}(x, y) \wedge \text{Kürbis}(y) \rightarrow \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))) \quad \wedge \\ & (\forall x(\text{Kind}(x) \rightarrow \forall y(\text{Kürbis}(y) \rightarrow \neg \text{Isst}(x, y)))) \quad \wedge \\ & \neg(\text{Kind}(b) \rightarrow \exists x \text{Schnitzt}(b, x)) \end{aligned}$$

0 Konjunktion von Prämissen und Negation der Konklusion

1 Negations-Normalform

1 Ersetzung von $\varphi \rightarrow \psi$ durch $\neg\varphi \vee \psi$

$$\begin{aligned} & (\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))) \quad \wedge \\ & (\forall x \forall y (\neg(\text{Kauft}(x, y) \wedge \text{Kürbis}(y)) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))) \quad \wedge \\ & (\forall x(\text{Kind}(x) \rightarrow \forall y(\text{Kürbis}(y) \rightarrow \neg \text{Isst}(x, y)))) \quad \wedge \\ & \neg(\text{Kind}(b) \rightarrow \exists x \text{Schnitzt}(b, x)) \end{aligned}$$

0 Konjunktion von Prämissen und Negation der Konklusion

1 Negations-Normalform

1 Ersetzung von $\varphi \rightarrow \psi$ durch $\neg\varphi \vee \psi$

$$\begin{aligned} & (\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))) \quad \wedge \\ & (\forall x \forall y (\neg(\text{Kauft}(x, y) \wedge \text{Kürbis}(y)) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))) \quad \wedge \\ & (\forall x (\neg \text{Kind}(x) \vee \forall y (\text{Kürbis}(y) \rightarrow \neg \text{Isst}(x, y)))) \quad \wedge \\ & \neg(\text{Kind}(b) \rightarrow \exists x \text{Schnitzt}(b, x)) \end{aligned}$$

0 Konjunktion von Prämissen und Negation der Konklusion

1 Negations-Normalform

1 Ersetzung von $\varphi \rightarrow \psi$ durch $\neg\varphi \vee \psi$

$$\begin{aligned} & (\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))) \quad \wedge \\ & (\forall x \forall y (\neg(\text{Kauft}(x, y) \wedge \text{Kürbis}(y)) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))) \quad \wedge \\ & (\forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y)))) \quad \wedge \\ & \neg(\text{Kind}(b) \rightarrow \exists x \text{Schnitzt}(b, x)) \end{aligned}$$

0 Konjunktion von Prämissen und Negation der Konklusion

1 Negations-Normalform

1 Ersetzung von $\varphi \rightarrow \psi$ durch $\neg\varphi \vee \psi$

$$\begin{aligned} & (\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))) \quad \wedge \\ & (\forall x \forall y (\neg(\text{Kauft}(x, y) \wedge \text{Kürbis}(y)) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))) \quad \wedge \\ & (\forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y)))) \quad \wedge \\ & \neg(\neg \text{Kind}(b) \vee \exists x \text{Schnitzt}(b, x)) \end{aligned}$$

0 Konjunktion von Prämissen und Negation der Konklusion

1 Negations-Normalform

1 Ersetzung von $\varphi \rightarrow \psi$ durch $\neg\varphi \vee \psi$

2 Gesetze von De Morgan

$$\begin{aligned} & (\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))) \quad \wedge \\ & (\forall x \forall y (\neg(\text{Kauft}(x, y) \wedge \text{Kürbis}(y)) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))) \quad \wedge \\ & (\forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y)))) \quad \wedge \\ & \neg(\neg \text{Kind}(b) \vee \exists x \text{Schnitzt}(b, x)) \end{aligned}$$

0 Konjunktion von Prämissen und Negation der Konklusion

1 Negations-Normalform

1 Ersetzung von $\varphi \rightarrow \psi$ durch $\neg\varphi \vee \psi$

2 Gesetze von De Morgan

$$\begin{aligned} & (\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))) \quad \wedge \\ & (\forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))) \quad \wedge \\ & (\forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y)))) \quad \wedge \\ & \neg(\neg \text{Kind}(b) \vee \exists x \text{Schnitzt}(b, x)) \end{aligned}$$

0 Konjunktion von Prämissen und Negation der Konklusion

1 Negations-Normalform

1 Ersetzung von $\varphi \rightarrow \psi$ durch $\neg\varphi \vee \psi$

2 Gesetze von De Morgan

$$\begin{aligned} & (\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))) \quad \wedge \\ & (\forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))) \quad \wedge \\ & (\forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y)))) \quad \wedge \\ & (\neg \neg \text{Kind}(b) \wedge \neg \exists x \text{Schnitzt}(b, x)) \end{aligned}$$

0 Konjunktion von Prämissen und Negation der Konklusion

1 Negations-Normalform

1 Ersetzung von $\varphi \rightarrow \psi$ durch $\neg\varphi \vee \psi$

2 Gesetze von De Morgan

$$\begin{aligned} & (\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))) \quad \wedge \\ & (\forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))) \quad \wedge \\ & (\forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y)))) \quad \wedge \\ & (\neg \neg \text{Kind}(b) \wedge \forall x \neg \text{Schnitzt}(b, x)) \end{aligned}$$

0 Konjunktion von Prämissen und Negation der Konklusion

1 Negations-Normalform

- 1 Ersetzung von $\varphi \rightarrow \psi$ durch $\neg\varphi \vee \psi$
- 2 Gesetze von De Morgan
- 3 Entfernung doppelter Negation

$$\begin{aligned} & (\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))) \quad \wedge \\ & (\forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))) \quad \wedge \\ & (\forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y)))) \quad \wedge \\ & (\neg \neg \text{Kind}(b) \wedge \forall x \neg \text{Schnitzt}(b, x)) \end{aligned}$$

0 Konjunktion von Prämissen und Negation der Konklusion

1 Negations-Normalform

1 Ersetzung von $\varphi \rightarrow \psi$ durch $\neg\varphi \vee \psi$

2 Gesetze von De Morgan

3 Entfernung doppelter Negation

$$\begin{aligned} & (\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))) \quad \wedge \\ & (\forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))) \quad \wedge \\ & (\forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y)))) \quad \wedge \\ & (\text{Kind}(b) \wedge \forall x \neg \text{Schnitzt}(b, x)) \end{aligned}$$

0 Konjunktion von Prämissen und Negation der Konklusion

1 Negations-Normalform

- 1 Ersetzung von $\varphi \rightarrow \psi$ durch $\neg\varphi \vee \psi$
- 2 Gesetze von De Morgan
- 3 Entfernung doppelter Negation

2 Skolemisierung

$$\begin{aligned} & (\exists x (\text{Kauft}(b, x) \wedge \text{Kürbis}(x))) \quad \wedge \\ & (\forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))) \quad \wedge \\ & (\forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y)))) \quad \wedge \\ & (\text{Kind}(b) \wedge \forall x \neg \text{Schnitzt}(b, x)) \end{aligned}$$

0 Konjunktion von Prämissen und Negation der Konklusion

1 Negations-Normalform

- 1 Ersetzung von $\varphi \rightarrow \psi$ durch $\neg\varphi \vee \psi$
- 2 Gesetze von De Morgan
- 3 Entfernung doppelter Negation

2 Skolemisierung

$$\begin{aligned} & (\text{Kauft}(b, k) \wedge \text{Kürbis}(k)) \quad \wedge \\ & (\forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))) \quad \wedge \\ & (\forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y)))) \quad \wedge \\ & (\text{Kind}(b) \wedge \forall x \neg \text{Schnitzt}(b, x)) \end{aligned}$$

0 Konjunktion von Prämissen und Negation der Konklusion

1 Negations-Normalform

- 1 Ersetzung von $\varphi \rightarrow \psi$ durch $\neg\varphi \vee \psi$
- 2 Gesetze von De Morgan
- 3 Entfernung doppelter Negation

2 Skolemisierung

3 Allquantoren

- 1 Variablen-Umbenennung

$$\begin{aligned} & (\text{Kauft}(b, k) \wedge \text{Kürbis}(k)) \quad \wedge \\ & (\forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))) \quad \wedge \\ & (\forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y)))) \quad \wedge \\ & (\text{Kind}(b) \wedge \forall x \neg \text{Schnitzt}(b, x)) \end{aligned}$$

0 Konjunktion von Prämissen und Negation der Konklusion

1 Negations-Normalform

- 1 Ersetzung von $\varphi \rightarrow \psi$ durch $\neg\varphi \vee \psi$
- 2 Gesetze von De Morgan
- 3 Entfernung doppelter Negation

2 Skolemisierung

3 Allquantoren

- 1 Variablen-Umbenennung

$$\begin{aligned} & (\text{Kauft}(b, k) \wedge \text{Kürbis}(k)) \quad \wedge \\ & (\forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))) \quad \wedge \\ & (\forall u (\neg \text{Kind}(u) \vee \forall w (\neg \text{Kürbis}(w) \vee \neg \text{Isst}(u, w)))) \quad \wedge \\ & (\text{Kind}(b) \wedge \forall x \neg \text{Schnitzt}(b, x)) \end{aligned}$$

0 Konjunktion von Prämissen und Negation der Konklusion

1 Negations-Normalform

- 1 Ersetzung von $\varphi \rightarrow \psi$ durch $\neg\varphi \vee \psi$
- 2 Gesetze von De Morgan
- 3 Entfernung doppelter Negation

2 Skolemisierung

3 Allquantoren

- 1 Variablen-Umbenennung

$$\begin{aligned} & (\text{Kauft}(b, k) \wedge \text{Kürbis}(k)) \quad \wedge \\ & (\forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))) \quad \wedge \\ & (\forall u (\neg \text{Kind}(u) \vee \forall w (\neg \text{Kürbis}(w) \vee \neg \text{Isst}(u, w)))) \quad \wedge \\ & (\text{Kind}(b) \wedge \forall z \neg \text{Schnitzt}(b, z)) \end{aligned}$$

0 Konjunktion von Prämissen und Negation der Konklusion

1 Negations-Normalform

- 1 Ersetzung von $\varphi \rightarrow \psi$ durch $\neg\varphi \vee \psi$
- 2 Gesetze von De Morgan
- 3 Entfernung doppelter Negation

2 Skolemisierung

3 Allquantoren

- 1 Variablen-Umbenennung
- 2 $\forall$ weglassen

$$\begin{aligned} & (\text{Kauft}(b, k) \wedge \text{Kürbis}(k)) \quad \wedge \\ & (\forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))) \quad \wedge \\ & (\forall u (\neg \text{Kind}(u) \vee \forall w (\neg \text{Kürbis}(w) \vee \neg \text{Isst}(u, w)))) \quad \wedge \\ & (\text{Kind}(b) \wedge \forall z \neg \text{Schnitzt}(b, z)) \end{aligned}$$

0 Konjunktion von Prämissen und Negation der Konklusion

1 Negations-Normalform

- 1 Ersetzung von $\varphi \rightarrow \psi$ durch $\neg\varphi \vee \psi$
- 2 Gesetze von De Morgan
- 3 Entfernung doppelter Negation

2 Skolemisierung

3 Allquantoren

- 1 Variablen-Umbenennung
- 2 $\forall$ weglassen

$$\begin{aligned} &(\text{Kauft}(b, k) \wedge \text{Kürbis}(k)) \quad \wedge \\ &(\neg\text{Kauft}(x, y) \vee \neg\text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y)) \quad \wedge \\ &(\neg\text{Kind}(u) \vee \neg\text{Kürbis}(w) \vee \neg\text{Isst}(u, w)) \quad \wedge \\ &(\text{Kind}(b) \wedge \neg\text{Schnitzt}(b, z)) \end{aligned}$$

0 Konjunktion von Prämissen und Negation der Konklusion

1 Negations-Normalform

- 1 Ersetzung von $\varphi \rightarrow \psi$ durch $\neg\varphi \vee \psi$
- 2 Gesetze von De Morgan
- 3 Entfernung doppelter Negation

2 Skolemisierung

3 Allquantoren

- 1 Variablen-Umbenennung
- 2 $\forall$ weglassen

4 Konjunktive Normalform

$(\text{Kauft}(b, k) \wedge \text{Kürbis}(k)) \quad \wedge$
 $(\neg\text{Kauft}(x, y) \vee \neg\text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y)) \quad \wedge$
 $(\neg\text{Kind}(u) \vee \neg\text{Kürbis}(w) \vee \neg\text{Isst}(u, w)) \quad \wedge$
 $(\text{Kind}(b) \wedge \neg\text{Schnitzt}(b, z))$

0 Konjunktion von Prämissen und Negation der Konklusion

1 Negations-Normalform

- 1 Ersetzung von $\varphi \rightarrow \psi$ durch $\neg\varphi \vee \psi$
- 2 Gesetze von De Morgan
- 3 Entfernung doppelter Negation

2 Skolemisierung

3 Allquantoren

- 1 Variablen-Umbenennung
- 2 $\forall$ weglassen

4 Konjunktive Normalform

$\text{Kauft}(b, k) \wedge$
 $\text{Kürbis}(k) \wedge$
 $(\neg\text{Kauft}(x, y) \vee \neg\text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y)) \wedge$
 $(\neg\text{Kind}(u) \vee \neg\text{Kürbis}(w) \vee \neg\text{Isst}(u, w)) \wedge$
 $(\text{Kind}(b) \wedge \neg\text{Schnitzt}(b, z))$

0 Konjunktion von Prämissen und Negation der Konklusion

1 Negations-Normalform

- 1 Ersetzung von $\varphi \rightarrow \psi$ durch $\neg\varphi \vee \psi$
- 2 Gesetze von De Morgan
- 3 Entfernung doppelter Negation

2 Skolemisierung

3 Allquantoren

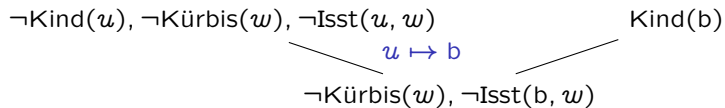
- 1 Variablen-Umbenennung
- 2 $\forall$ weglassen

4 Konjunktive Normalform

$\text{Kauft}(b, k) \quad \wedge$
 $\text{Kürbis}(k) \quad \wedge$
 $(\neg\text{Kauft}(x, y) \vee \neg\text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y)) \quad \wedge$
 $(\neg\text{Kind}(u) \vee \neg\text{Kürbis}(w) \vee \neg\text{Isst}(u, w)) \quad \wedge$
 $\text{Kind}(b) \quad \wedge$
 $\neg\text{Schnitzt}(b, z)$

$\neg \text{Kind}(u), \neg \text{Kürbis}(w), \neg \text{Isst}(u, w)$

$\text{Kind}(b)$



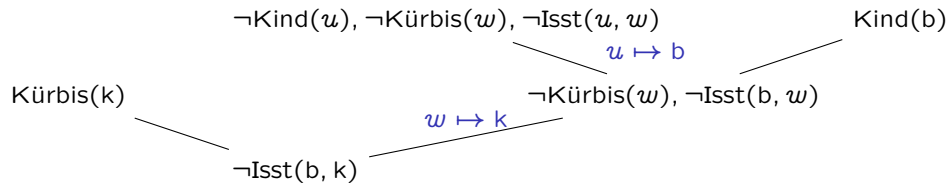
$\neg \text{Kind}(u), \neg \text{Kürbis}(w), \neg \text{Isst}(u, w)$

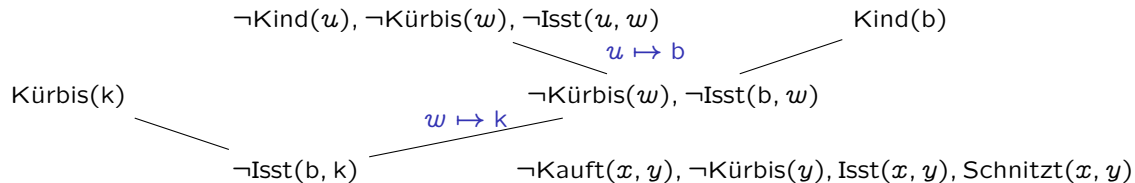
$\text{Kind}(b)$

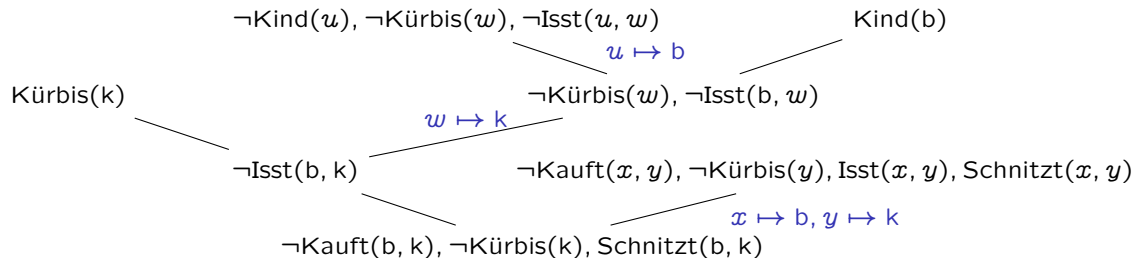
$u \mapsto b$

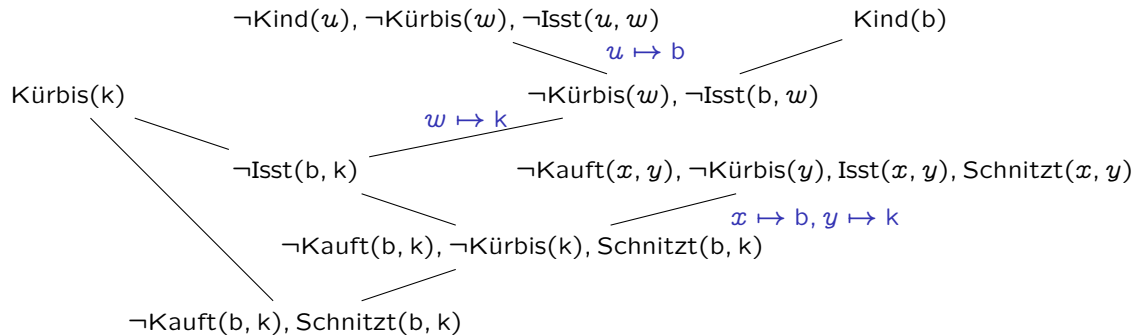
$\text{Kürbis}(k)$

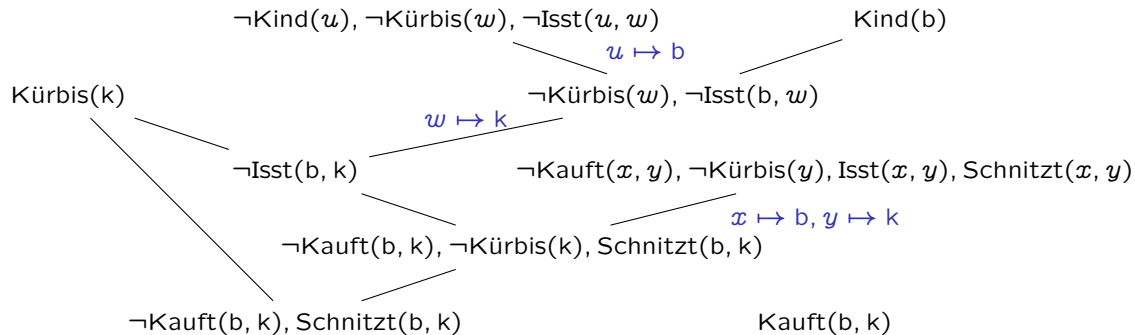
$\neg \text{Kürbis}(w), \neg \text{Isst}(b, w)$

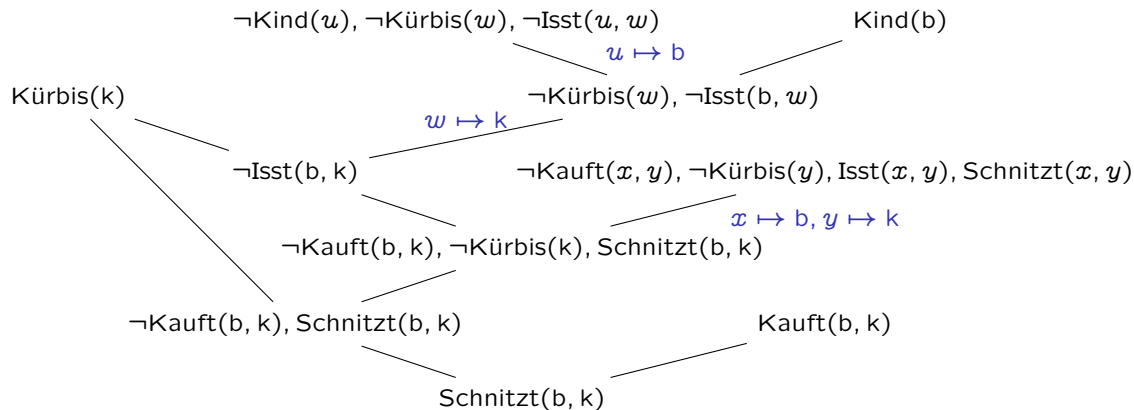


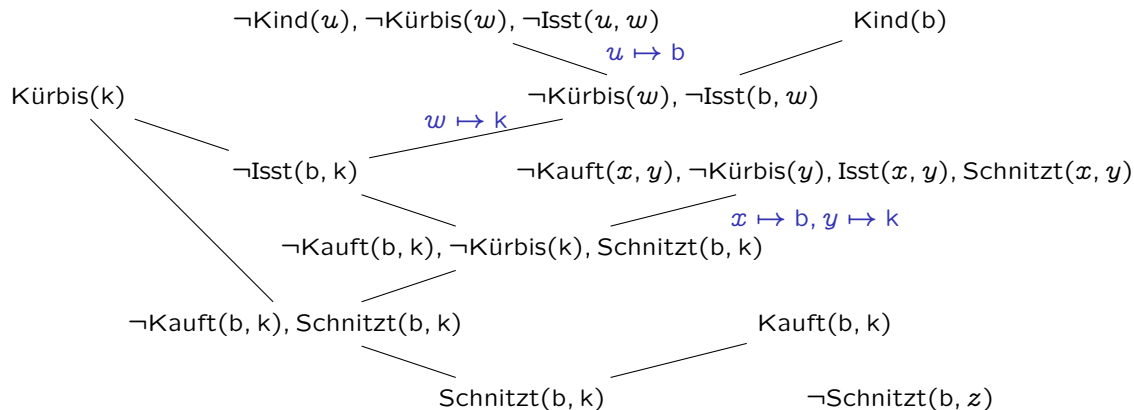


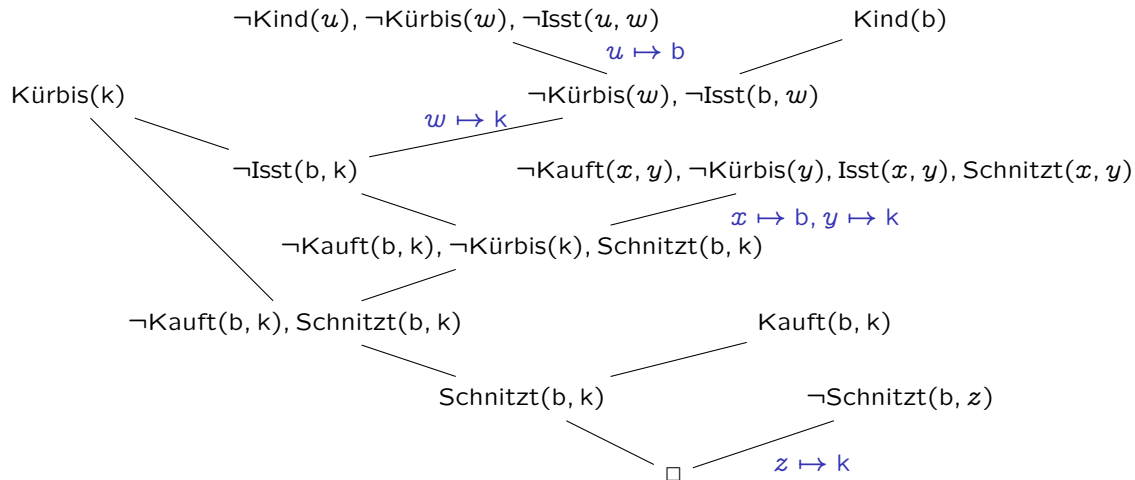












Übung 4.68

Zeigen Sie mittels Resolution der folgenden Folgerung:

Prämissen:

- 1 Alle Hunde bellen.
- 2 Wer Katzen hat, hat keine Mäuse.
- 3 Wer leicht aufwacht, hat nichts, das bellt.
- 4 Anna hat einen Hund oder eine Katze.

Konklusion:

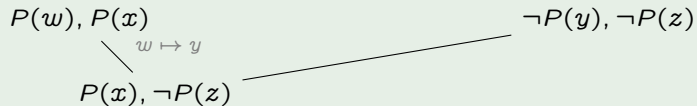
- 5 Wenn Anna leicht aufwacht, hat sie keine Mäuse.

Beispiel 4.69 (Resolution von $\{P(w), P(x)\}$ und $\{(\neg P(y), \neg P(z))\}$)

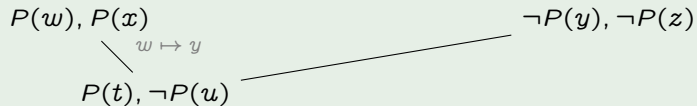
$P(w), P(x)$

$\neg P(y), \neg P(z)$

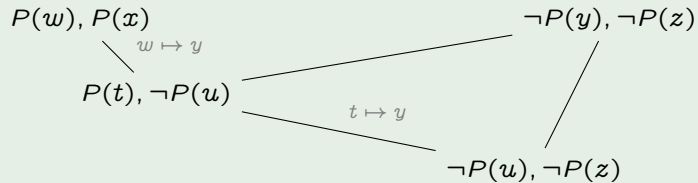
Beispiel 4.69 (Resolution von $\{P(w), P(x)\}$ und $\{\neg P(y), \neg P(z)\}$)



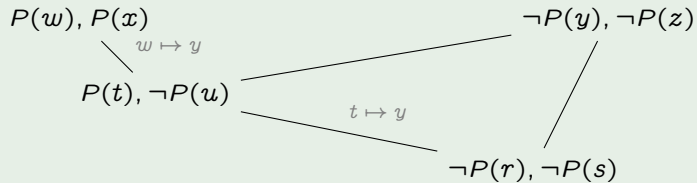
Beispiel 4.69 (Resolution von $\{P(w), P(x)\}$ und $\{\neg P(y), \neg P(z)\}$)



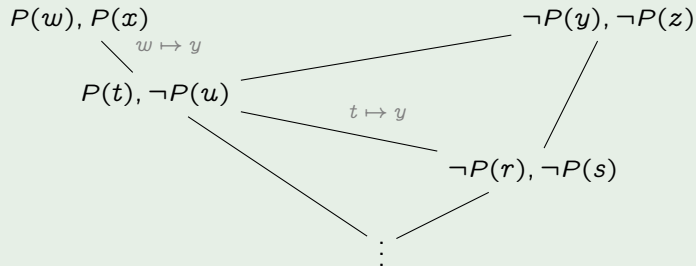
Beispiel 4.69 (Resolution von $\{P(w), P(x)\}$ und $\{(\neg P(y), \neg P(z))\}$)



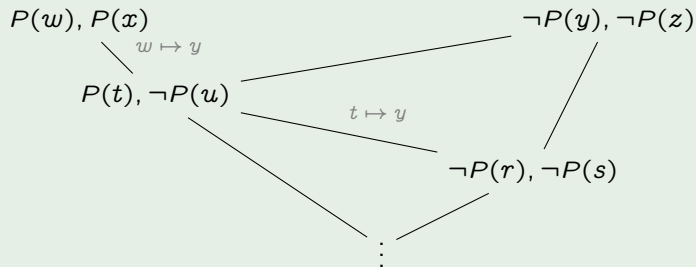
Beispiel 4.69 (Resolution von $\{P(w), P(x)\}$ und $\{(\neg P(y), \neg P(z))\}$)



Beispiel 4.69 (Resolution von $\{P(w), P(x)\}$ und $\{(\neg P(y), \neg P(z))\}$)



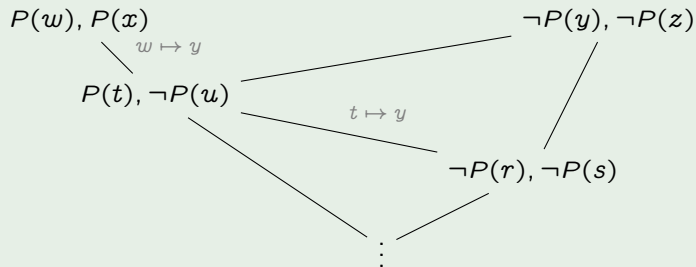
Beispiel 4.69 (Resolution von $\{P(w), P(x)\}$ und $\{(\neg P(y), \neg P(z))\}$)



Problem

- redundante Literale
- $\forall w \forall x (P(w) \vee P(x)) \quad \equiv \quad \forall x P(x)$

Beispiel 4.69 (Resolution von $\{P(w), P(x)\}$ und $\{(\neg P(y), \neg P(z))\}$)



Problem

- redundante Literale
- $\forall w \forall x (P(w) \vee P(x)) \equiv \forall x P(x)$

Lösung

- eliminiere Redundanz
- führe Unifikation **innerhalb** einer Klausel durch

Definition 4.70 (Faktor)

Sei $C = \{L_1, L_2, L_3, \dots\}$ und σ allgemeinsten Unifikator von L_1 und L_2 .
Dann hat C den Faktor $\sigma(C)$.

Definition 4.70 (Faktor)

Sei $C = \{L_1, L_2, L_3, \dots\}$ und σ allgemeinsten Unifikator von L_1 und L_2 .
Dann hat C den Faktor $\sigma(C)$.

Beispiel 4.71 (Faktoren)

- $\{P(w), P(x)\}$ hat Faktor $\{P(x)\}$.

Definition 4.70 (Faktor)

Sei $C = \{L_1, L_2, L_3, \dots\}$ und σ allgemeinsten Unifikator von L_1 und L_2 .
Dann hat C den Faktor $\sigma(C)$.

Beispiel 4.71 (Faktoren)

- $\{P(w), P(x)\}$ hat Faktor $\{P(x)\}$.
- $\{P(x), P(c)\}$ hat Faktor $\{P(c)\}$.

Definition 4.70 (Faktor)

Sei $C = \{L_1, L_2, L_3, \dots\}$ und σ allgemeinsten Unifikator von L_1 und L_2 .
Dann hat C den **Faktor** $\sigma(C)$.

Beispiel 4.71 (Faktoren)

- $\{P(w), P(x)\}$ hat Faktor $\{P(x)\}$.
- $\{P(x), P(c)\}$ hat Faktor $\{P(c)\}$.
- $\{P(x), P(c), R(x, y)\}$ hat Faktor $\{P(c), R(c, y)\}$.

Neue Regel:

Wenn C_1 den Faktor C_2 hat, leite C_2 von C_1 ab ($C_1 \vdash C_2$).

Neue Regel:

Wenn C_1 den Faktor C_2 hat, leite C_2 von C_1 ab ($C_1 \vdash C_2$).

Beispiel 4.72 (Faktorisierung)

$P(w), P(x)$

$\neg P(y), \neg P(z)$

Neue Regel:

Wenn C_1 den Faktor C_2 hat, leite C_2 von C_1 ab ($C_1 \vdash C_2$).

Beispiel 4.72 (Faktorisierung)

$$P(w), P(x)$$
$$\mid w \mapsto x$$
$$P(x)$$
$$\neg P(y), \neg P(z)$$

Neue Regel:

Wenn C_1 den Faktor C_2 hat, leite C_2 von C_1 ab ($C_1 \vdash C_2$).

Beispiel 4.72 (Faktorisierung)

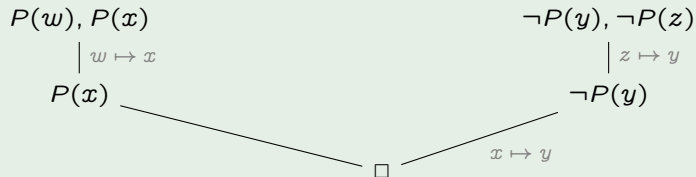
$$\begin{array}{c} P(w), P(x) \\ | \quad w \mapsto x \\ P(x) \end{array}$$

$$\begin{array}{c} \neg P(y), \neg P(z) \\ | \quad z \mapsto y \\ \neg P(y) \end{array}$$

Neue Regel:

Wenn C_1 den Faktor C_2 hat, leite C_2 von C_1 ab ($C_1 \vdash C_2$).

Beispiel 4.72 (Faktorisierung)



Satz 4.73 (Korrektheit der PL-Resolution)

$\mathcal{K}(\varphi) \vdash^* \Box$ impliziert Unerfüllbarkeit von φ .

Satz 4.73 (Korrektheit der PL-Resolution)

$\mathcal{K}(\varphi) \vdash^* \Box$ impliziert Unerfüllbarkeit von φ .

Satz 4.74 (Widerlegungsvollständigkeit der PL-Resolution)

Wenn φ unerfüllbar ist, gilt $\mathcal{K}(\varphi) \vdash^* \Box$.

Satz 4.73 (Korrektheit der PL-Resolution)

$\mathcal{K}(\varphi) \vdash^* \Box$ impliziert Unerfüllbarkeit von φ .

Satz 4.74 (Widerlegungsvollständigkeit der PL-Resolution)

Wenn φ unerfüllbar ist, gilt $\mathcal{K}(\varphi) \vdash^* \Box$.

Terminierung?

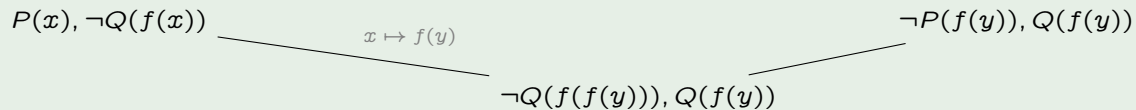
Terminiert der Resolutionsalgorithmus für jede Eingabe?

Beispiel 4.75

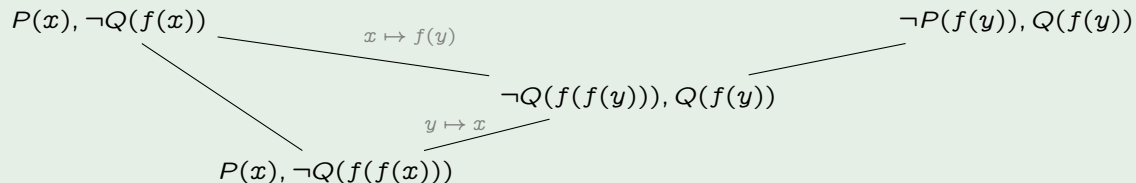
$$P(x), \neg Q(f(x))$$

$$\neg P(f(y)), Q(f(y))$$

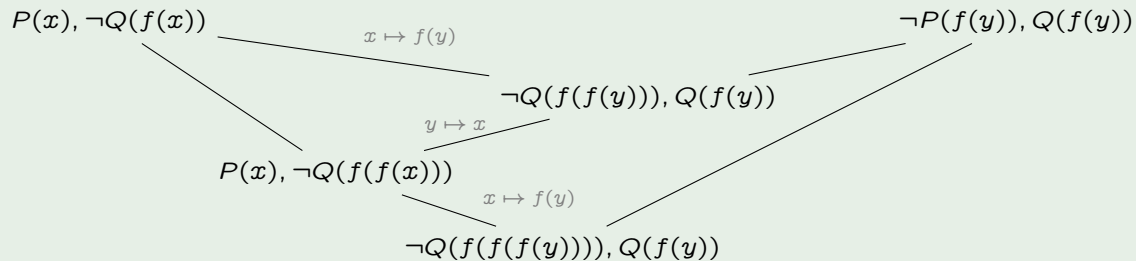
Beispiel 4.75



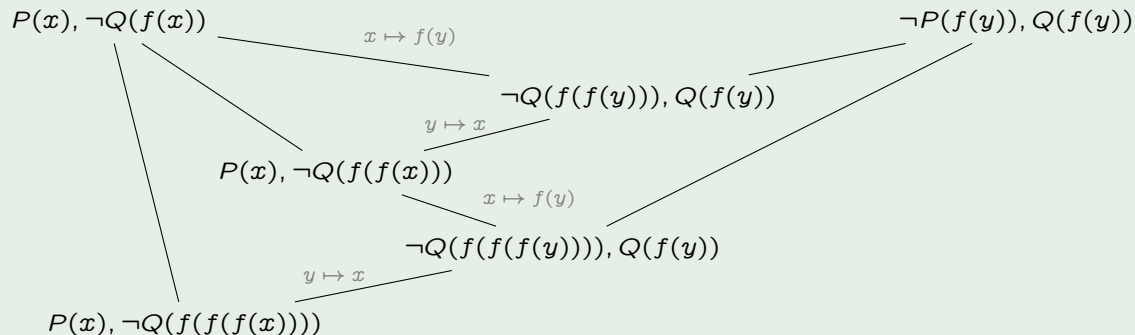
Beispiel 4.75



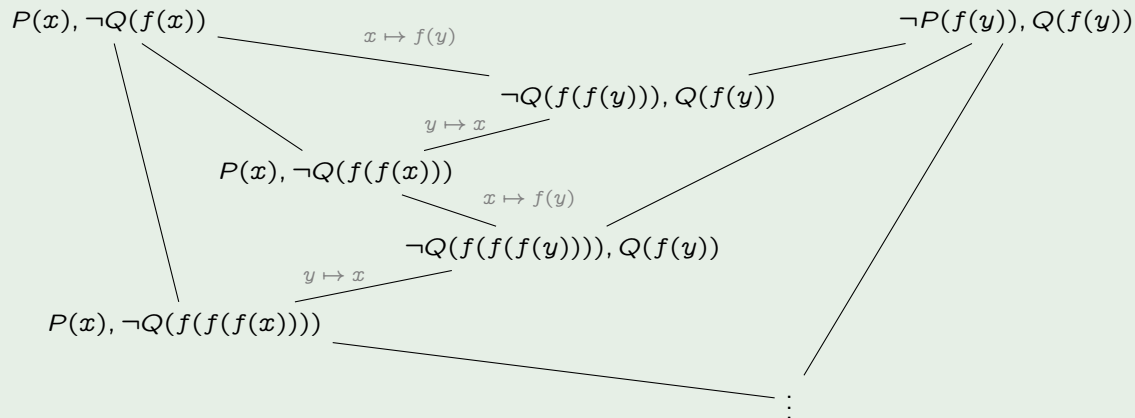
Beispiel 4.75



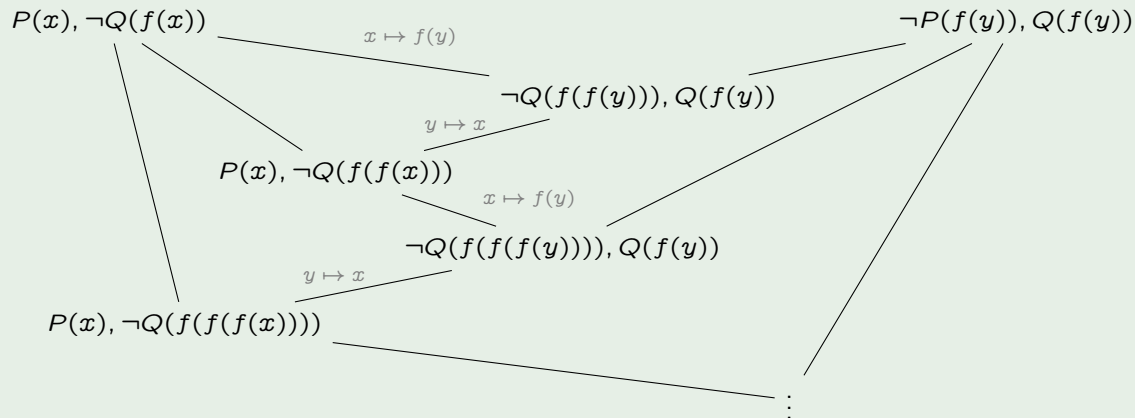
Beispiel 4.75



Beispiel 4.75



Beispiel 4.75



PL-Erfüllbarkeit unentscheidbar $\rightsquigarrow$ Terminierung nicht erreichbar.

1. Einführung

2. Mengen

3. Aussagenlogik

4. Prädikatenlogik

4.1 Relationen und Funktionen

4.2 Syntax

4.3 Semantik

4.4 Schlussfolgerungsverfahren

4.4.1 Resolution

4.4.2 Tableaus

5. Prolog

- Gleiches Grundprinzip

- Gleiches Grundprinzip
 - Regeln

- Gleiches Grundprinzip
 - Regeln
 - Clashes

- Gleiches Grundprinzip
 - Regeln
 - Clashes
- Vorverarbeitung: NNF

- Gleiches Grundprinzip
 - Regeln
 - Clashes
- Vorverarbeitung: NNF
 - Negation nur vor Atomen

- Gleiches Grundprinzip
 - Regeln
 - Clashes
- Vorverarbeitung: NNF
 - Negation nur vor Atomen
- Zusätzliche Regeln für Quantoren

- Gleiches Grundprinzip
 - Regeln
 - Clashes
- Vorverarbeitung: NNF
 - Negation nur vor Atomen
- Zusätzliche Regeln für Quantoren
 - $\forall$ -Regel

- Gleiches Grundprinzip
 - Regeln
 - Clashes
- Vorverarbeitung: NNF
 - Negation nur vor Atomen
- Zusätzliche Regeln für Quantoren
 - $\forall$ -Regel
 - $\exists$ -Regel

- Gleiches Grundprinzip
 - Regeln
 - Clashes
- Vorverarbeitung: NNF
 - Negation nur vor Atomen
- Zusätzliche Regeln für Quantoren
 - $\forall$ -Regel
 - $\exists$ -Regel
- Definition von Clash auf PL-Formeln erweitert

- ∃ Wenn $\exists x\varphi \in S$ gilt
und **keine Substitution** $\sigma = \{x \mapsto t\}$ für einen Term t existiert mit $\sigma(\varphi) \in S$
dann $S := S \cup \sigma(\varphi)$ mit $\sigma = \{x \mapsto c\}$ für ein neues Konstantensymbol c

- Wenn $\exists x\varphi \in S$ gilt
und **keine Substitution** $\sigma = \{x \mapsto t\}$ für einen Term t existiert mit $\sigma(\varphi) \in S$
dann $S := S \cup \sigma(\varphi)$ mit $\sigma = \{x \mapsto c\}$ für ein neues Konstantensymbol c
- eliminiere $\exists$ durch Skolemisierung

- Wenn $\exists x\varphi \in S$ gilt
und **keine Substitution** $\sigma = \{x \mapsto t\}$ für einen Term t existiert mit $\sigma(\varphi) \in S$
dann $S := S \cup \sigma(\varphi)$ mit $\sigma = \{x \mapsto c\}$ für ein neues Konstantensymbol c
- eliminiere $\exists$ durch Skolemisierung
 - Konstanten können immer verwendet werden

- Wenn $\exists x\varphi \in S$ gilt
und **keine Substitution** $\sigma = \{x \mapsto t\}$ für einen Term t existiert mit $\sigma(\varphi) \in S$
dann $S := S \cup \sigma(\varphi)$ mit $\sigma = \{x \mapsto c\}$ für ein neues Konstantensymbol c
- eliminiere $\exists$ durch Skolemisierung
 - Konstanten können immer verwendet werden
 - $\forall$ -Quantoren werden vorher durch $\forall$ -Regel bearbeitet

- Wenn $\exists x\varphi \in S$ gilt
und **keine Substitution** $\sigma = \{x \mapsto t\}$ für einen Term t existiert mit $\sigma(\varphi) \in S$
dann $S := S \cup \sigma(\varphi)$ mit $\sigma = \{x \mapsto c\}$ für ein neues Konstantensymbol c
- eliminiere $\exists$ durch Skolemisierung
 - Konstanten können immer verwendet werden
 - $\forall$ -Quantoren werden vorher durch $\forall$ -Regel bearbeitet
 - keine komplexen Skolem-Funktionen nötig

- Wenn $\exists x\varphi \in S$ gilt
und **keine Substitution** $\sigma = \{x \mapsto t\}$ für einen Term t existiert mit $\sigma(\varphi) \in S$
dann $S := S \cup \sigma(\varphi)$ mit $\sigma = \{x \mapsto c\}$ für ein neues Konstantensymbol c
- eliminiere $\exists$ durch Skolemisierung
 - Konstanten können immer verwendet werden
 - $\forall$ -Quantoren werden vorher durch $\forall$ -Regel bearbeitet
 - keine komplexen Skolem-Funktionen nötig
- Wenn $\forall x\varphi \in S$ gilt
und t ein **beliebiger** Grundterm ist
dann $S := S \cup \sigma(\varphi)$ mit $\sigma = \{x \mapsto t\}$

- Wenn $\exists x\varphi \in S$ gilt
und **keine Substitution** $\sigma = \{x \mapsto t\}$ für einen Term t existiert mit $\sigma(\varphi) \in S$
dann $S := S \cup \sigma(\varphi)$ mit $\sigma = \{x \mapsto c\}$ für ein neues Konstantensymbol c
- eliminiere $\exists$ durch Skolemisierung
 - Konstanten können immer verwendet werden
 - $\forall$ -Quantoren werden vorher durch $\forall$ -Regel bearbeitet
 - keine komplexen Skolem-Funktionen nötig
- Wenn $\forall x\varphi \in S$ gilt
und t ein **beliebiger** Grundterm ist
dann $S := S \cup \sigma(\varphi)$ mit $\sigma = \{x \mapsto t\}$
- t kann auch neues Konstantensymbol sein

- $\exists$ Wenn $\exists x\varphi \in S$ gilt
und **keine Substitution** $\sigma = \{x \mapsto t\}$ für einen Term t existiert mit $\sigma(\varphi) \in S$
dann $S := S \cup \sigma(\varphi)$ mit $\sigma = \{x \mapsto c\}$ für ein neues Konstantensymbol c
- eliminiere $\exists$ durch Skolemisierung
 - Konstanten können immer verwendet werden
 - $\forall$ -Quantoren werden vorher durch $\forall$ -Regel bearbeitet
 - keine komplexen Skolem-Funktionen nötig
- $\forall$ Wenn $\forall x\varphi \in S$ gilt
und t ein **beliebiger** Grundterm ist
dann $S := S \cup \sigma(\varphi)$ mit $\sigma = \{x \mapsto t\}$
- t kann auch neues Konstantensymbol sein
 - anwendbar auf unendliche Menge von Grundtermen

- $\exists$ Wenn $\exists x\varphi \in S$ gilt
und **keine Substitution** $\sigma = \{x \mapsto t\}$ für einen Term t existiert mit $\sigma(\varphi) \in S$
dann $S := S \cup \sigma(\varphi)$ mit $\sigma = \{x \mapsto c\}$ für ein neues Konstantensymbol c
- eliminiere $\exists$ durch Skolemisierung
 - Konstanten können immer verwendet werden
 - $\forall$ -Quantoren werden vorher durch $\forall$ -Regel bearbeitet
 - keine komplexen Skolem-Funktionen nötig
- $\forall$ Wenn $\forall x\varphi \in S$ gilt
und t ein **beliebiger** Grundterm ist
dann $S := S \cup \sigma(\varphi)$ mit $\sigma = \{x \mapsto t\}$
- t kann auch neues Konstantensymbol sein
 - anwendbar auf unendliche Menge von Grundtermen
 - Keine Anwendbarkeitsbedingung $\rightsquigarrow$ **immer anwendbar!**

- $\exists$ Wenn $\exists x\varphi \in S$ gilt
und **keine Substitution** $\sigma = \{x \mapsto t\}$ für einen Term t existiert mit $\sigma(\varphi) \in S$
dann $S := S \cup \sigma(\varphi)$ mit $\sigma = \{x \mapsto c\}$ für ein neues Konstantensymbol c
- eliminiere $\exists$ durch Skolemisierung
 - Konstanten können immer verwendet werden
 - $\forall$ -Quantoren werden vorher durch $\forall$ -Regel bearbeitet
 - keine komplexen Skolem-Funktionen nötig
- $\forall$ Wenn $\forall x\varphi \in S$ gilt
und t ein **beliebiger** Grundterm ist
dann $S := S \cup \sigma(\varphi)$ mit $\sigma = \{x \mapsto t\}$
- t kann auch neues Konstantensymbol sein
 - anwendbar auf unendliche Menge von Grundtermen
 - Keine Anwendbarkeitsbedingung $\rightsquigarrow$ **immer anwendbar!**
 - Terminierung ist nicht garantiert

Beispiel: Halloween-Tableau

1	$\begin{array}{l} \exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x)) \quad \wedge \\ \forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y)) \quad \wedge \\ \forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y))) \quad \wedge \\ \text{Kind}(b) \wedge \forall x \neg \text{Schnitzt}(b, x) \end{array}$

Beispiel: Halloween-Tableau

1	$\begin{aligned} & \exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x)) \quad \wedge \\ & \forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y)) \quad \wedge \\ & \forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y))) \quad \wedge \\ & \text{Kind}(b) \wedge \forall x \neg \text{Schnitzt}(b, x) \end{aligned}$	
2	$\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))$	1: $\wedge$
3	$\forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))$	
4	$\forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y)))$	
5	$\text{Kind}(b)$	
6	$\forall x \neg \text{Schnitzt}(b, x)$	

Beispiel: Halloween-Tableau

1	$\begin{aligned} & \exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x)) \quad \wedge \\ & \forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y)) \quad \wedge \\ & \forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y))) \quad \wedge \\ & \text{Kind}(b) \wedge \forall x \neg \text{Schnitzt}(b, x) \end{aligned}$	
2	$\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))$	1: $\wedge$
3	$\forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))$	
4	$\forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y)))$	
5	$\text{Kind}(b)$	
6	$\forall x \neg \text{Schnitzt}(b, x)$	
7	$\text{Kauft}(b, k) \wedge \text{Kürbis}(k)$	2: $\exists (x \mapsto k)$

Beispiel: Halloween-Tableau

1	$\begin{aligned} & \exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x)) \quad \wedge \\ & \forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y)) \quad \wedge \\ & \forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y))) \quad \wedge \\ & \text{Kind}(b) \wedge \forall x \neg \text{Schnitzt}(b, x) \end{aligned}$	
2	$\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))$	1: $\wedge$
3	$\forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))$	
4	$\forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y)))$	
5	$\text{Kind}(b)$	
6	$\forall x \neg \text{Schnitzt}(b, x)$	
7	$\text{Kauft}(b, k) \wedge \text{Kürbis}(k)$	2: $\exists (x \mapsto k)$
8	$\text{Kauft}(b, k)$	7: $\wedge$
9	$\text{Kürbis}(k)$	

Beispiel: Halloween-Tableau

1	$\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x)) \wedge$ $\forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y)) \wedge$ $\forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y))) \wedge$ $\text{Kind}(b) \wedge \forall x \neg \text{Schnitzt}(b, x)$	
2	$\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))$	1: $\wedge$
3	$\forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))$	
4	$\forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y)))$	
5	$\text{Kind}(b)$	
6	$\forall x \neg \text{Schnitzt}(b, x)$	
7	$\text{Kauft}(b, k) \wedge \text{Kürbis}(k)$	2: $\exists (x \mapsto k)$
8	$\text{Kauft}(b, k)$	7: $\wedge$
9	$\text{Kürbis}(k)$	
10	$\neg \text{Schnitzt}(b, k)$	6: $\forall (x \mapsto k)$

Beispiel: Halloween-Tableau

1	$\begin{aligned} & \exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x)) \quad \wedge \\ & \forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y)) \quad \wedge \\ & \forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y))) \quad \wedge \\ & \text{Kind}(b) \wedge \forall x \neg \text{Schnitzt}(b, x) \end{aligned}$	
2	$\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))$	1: $\wedge$
3	$\forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))$	
4	$\forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y)))$	
5	$\text{Kind}(b)$	
6	$\forall x \neg \text{Schnitzt}(b, x)$	
7	$\text{Kauft}(b, k) \wedge \text{Kürbis}(k)$	2: $\exists(x \mapsto k)$
8	$\text{Kauft}(b, k)$	7: $\wedge$
9	$\text{Kürbis}(k)$	
10	$\neg \text{Schnitzt}(b, k)$	6: $\forall(x \mapsto k)$
11	$\neg \text{Kauft}(b, k) \vee \neg \text{Kürbis}(k) \vee \text{Isst}(b, k) \vee \text{Schnitzt}(b, k)$	3: $\forall(x \mapsto b, y \mapsto k)$

Beispiel: Halloween-Tableau

1	$\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x)) \wedge$ $\forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y)) \wedge$ $\forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y))) \wedge$ $\text{Kind}(b) \wedge \forall x \neg \text{Schnitzt}(b, x)$	
2	$\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))$	1: $\wedge$
3	$\forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))$	
4	$\forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y)))$	
5	$\text{Kind}(b)$	
6	$\forall x \neg \text{Schnitzt}(b, x)$	
7	$\text{Kauft}(b, k) \wedge \text{Kürbis}(k)$	2: $\exists (x \mapsto k)$
8	$\text{Kauft}(b, k)$	7: $\wedge$
9	$\text{Kürbis}(k)$	
10	$\neg \text{Schnitzt}(b, k)$	6: $\forall (x \mapsto k)$
11	$\neg \text{Kauft}(b, k) \vee \neg \text{Kürbis}(k) \vee \text{Isst}(b, k) \vee \text{Schnitzt}(b, k)$	3: $\forall (x \mapsto b, y \mapsto k)$
12	$\neg \text{Kind}(b) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(b, y))$	4: $\forall (x \mapsto b)$

Beispiel: Halloween-Tableau

1	$\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x)) \wedge$ $\forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y)) \wedge$ $\forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y))) \wedge$ $\text{Kind}(b) \wedge \forall x \neg \text{Schnitzt}(b, x)$	
2	$\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))$	1: $\wedge$
3	$\forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))$	
4	$\forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y)))$	
5	$\text{Kind}(b)$	
6	$\forall x \neg \text{Schnitzt}(b, x)$	
7	$\text{Kauft}(b, k) \wedge \text{Kürbis}(k)$	2: $\exists (x \mapsto k)$
8	$\text{Kauft}(b, k)$	7: $\wedge$
9	$\text{Kürbis}(k)$	
10	$\neg \text{Schnitzt}(b, k)$	6: $\forall (x \mapsto k)$
11	$\neg \text{Kauft}(b, k) \vee \neg \text{Kürbis}(k) \vee \text{Isst}(b, k) \vee \text{Schnitzt}(b, k)$	3: $\forall (x \mapsto b, y \mapsto k)$
12	$\neg \text{Kind}(b) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(b, y))$	4: $\forall (x \mapsto b)$
13	$\neg \text{Kind}(b) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(b, y))$	12: $\vee$

Beispiel: Halloween-Tableau

1	$\begin{aligned} & \exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x)) \quad \wedge \\ & \forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y)) \quad \wedge \\ & \forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y))) \quad \wedge \\ & \text{Kind}(b) \wedge \forall x \neg \text{Schnitzt}(b, x) \end{aligned}$		
2	$\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))$		1: $\wedge$
3	$\forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))$		
4	$\forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y)))$		
5	$\text{Kind}(b)$		
6	$\forall x \neg \text{Schnitzt}(b, x)$		
7	$\text{Kauft}(b, k) \wedge \text{Kürbis}(k)$		2: $\exists(x \mapsto k)$
8	$\text{Kauft}(b, k)$		7: $\wedge$
9	$\text{Kürbis}(k)$		
10	$\neg \text{Schnitzt}(b, k)$		6: $\forall(x \mapsto k)$
11	$\neg \text{Kauft}(b, k) \vee \neg \text{Kürbis}(k) \vee \text{Isst}(b, k) \vee \text{Schnitzt}(b, k)$		3: $\forall(x \mapsto b, y \mapsto k)$
12	$\neg \text{Kind}(b) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(b, y))$		4: $\forall(x \mapsto b)$
13	$\neg \text{Kind}(b)$	$\forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(b, y))$	12: $\vee$
14	$\nmid 5$		

Beispiel: Halloween-Tableau

1	$\begin{aligned} & \exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x)) \quad \wedge \\ & \forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y)) \quad \wedge \\ & \forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y))) \quad \wedge \\ & \text{Kind}(b) \wedge \forall x \neg \text{Schnitzt}(b, x) \end{aligned}$		
2	$\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))$		1: $\wedge$
3	$\forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))$		
4	$\forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y)))$		
5	$\text{Kind}(b)$		
6	$\forall x \neg \text{Schnitzt}(b, x)$		
7	$\text{Kauft}(b, k) \wedge \text{Kürbis}(k)$		2: $\exists(x \mapsto k)$
8	$\text{Kauft}(b, k)$		7: $\wedge$
9	$\text{Kürbis}(k)$		
10	$\neg \text{Schnitzt}(b, k)$		6: $\forall(x \mapsto k)$
11	$\neg \text{Kauft}(b, k) \vee \neg \text{Kürbis}(k) \vee \text{Isst}(b, k) \vee \text{Schnitzt}(b, k)$		3: $\forall(x \mapsto b, y \mapsto k)$
12	$\neg \text{Kind}(b) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(b, y))$		4: $\forall(x \mapsto b)$
13	$\neg \text{Kind}(b)$	$\forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(b, y))$	12: $\vee$
14	$\nexists 5$	$\neg \text{Kürbis}(k) \vee \neg \text{Isst}(b, k)$	13: $\forall(y \mapsto k)$

Beispiel: Halloween-Tableau

1	$\begin{aligned} & \exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x)) \quad \wedge \\ & \forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y)) \quad \wedge \\ & \forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y))) \quad \wedge \\ & \text{Kind}(b) \wedge \forall x \neg \text{Schnitzt}(b, x) \end{aligned}$		
2	$\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))$		1: $\wedge$
3	$\forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))$		
4	$\forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y)))$		
5	$\text{Kind}(b)$		
6	$\forall x \neg \text{Schnitzt}(b, x)$		
7	$\text{Kauft}(b, k) \wedge \text{Kürbis}(k)$		2: $\exists(x \mapsto k)$
8	$\text{Kauft}(b, k)$		7: $\wedge$
9	$\text{Kürbis}(k)$		
10	$\neg \text{Schnitzt}(b, k)$		6: $\forall(x \mapsto k)$
11	$\neg \text{Kauft}(b, k) \vee \neg \text{Kürbis}(k) \vee \text{Isst}(b, k) \vee \text{Schnitzt}(b, k)$		3: $\forall(x \mapsto b, y \mapsto k)$
12	$\neg \text{Kind}(b) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(b, y))$		4: $\forall(x \mapsto b)$
13	$\neg \text{Kind}(b)$	$\forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(b, y))$	12: $\vee$
14	$\not\downarrow$ 5	$\neg \text{Kürbis}(k) \vee \neg \text{Isst}(b, k)$	13: $\forall(y \mapsto k)$
15		$\neg \text{Kürbis}(k) \quad \neg \text{Isst}(b, k)$	14: $\vee$

Beispiel: Halloween-Tableau

1	$\begin{aligned} & \exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x)) \quad \wedge \\ & \forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y)) \quad \wedge \\ & \forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y))) \quad \wedge \\ & \text{Kind}(b) \wedge \forall x \neg \text{Schnitzt}(b, x) \end{aligned}$		
2	$\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))$		1: $\wedge$
3	$\forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))$		
4	$\forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y)))$		
5	$\text{Kind}(b)$		
6	$\forall x \neg \text{Schnitzt}(b, x)$		
7	$\text{Kauft}(b, k) \wedge \text{Kürbis}(k)$		2: $\exists(x \mapsto k)$
8	$\text{Kauft}(b, k)$		7: $\wedge$
9	$\text{Kürbis}(k)$		
10	$\neg \text{Schnitzt}(b, k)$		6: $\forall(x \mapsto k)$
11	$\neg \text{Kauft}(b, k) \vee \neg \text{Kürbis}(k) \vee \text{Isst}(b, k) \vee \text{Schnitzt}(b, k)$		3: $\forall(x \mapsto b, y \mapsto k)$
12	$\neg \text{Kind}(b) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(b, y))$		4: $\forall(x \mapsto b)$
13	$\neg \text{Kind}(b)$	$\forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(b, y))$	12: $\vee$
14	$\not\downarrow$ 5	$\neg \text{Kürbis}(k) \vee \neg \text{Isst}(b, k)$	13: $\forall(y \mapsto k)$
15		$\neg \text{Kürbis}(k) \quad \neg \text{Isst}(b, k)$	14: $\vee$
16		$\not\downarrow$ 9	

Beispiel: Halloween-Tableau

1	$\begin{aligned} & \exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x)) \quad \wedge \\ & \forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y)) \quad \wedge \\ & \forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y))) \quad \wedge \\ & \text{Kind}(b) \wedge \forall x \neg \text{Schnitzt}(b, x) \end{aligned}$						
2	$\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))$						1: $\wedge$
3	$\forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))$						
4	$\forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y)))$						
5	$\text{Kind}(b)$						
6	$\forall x \neg \text{Schnitzt}(b, x)$						
7	$\text{Kauft}(b, k) \wedge \text{Kürbis}(k)$						2: $\exists(x \mapsto k)$
8	$\text{Kauft}(b, k)$						7: $\wedge$
9	$\text{Kürbis}(k)$						
10	$\neg \text{Schnitzt}(b, k)$						6: $\forall(x \mapsto k)$
11	$\neg \text{Kauft}(b, k) \vee \neg \text{Kürbis}(k) \vee \text{Isst}(b, k) \vee \text{Schnitzt}(b, k)$						3: $\forall(x \mapsto b, y \mapsto k)$
12	$\neg \text{Kind}(b) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(b, y))$						4: $\forall(x \mapsto b)$
13	$\neg \text{Kind}(b)$	$\forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(b, y))$					12: $\vee$
14	$\not\downarrow$ 5	$\neg \text{Kürbis}(k) \vee \neg \text{Isst}(b, k)$					13: $\forall(y \mapsto k)$
15		$\neg \text{Kürbis}(k)$	$\neg \text{Isst}(b, k)$				14: $\vee$
16		$\not\downarrow$ 9	$\neg \text{Kauft}(b, k)$	$\neg \text{Kürbis}(k)$	$\text{Isst}(b, k)$	$\text{Schnitzt}(b, k)$	11: $\vee$

Beispiel: Halloween-Tableau

1	$\begin{aligned} & \exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x)) \quad \wedge \\ & \forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y)) \quad \wedge \\ & \forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y))) \quad \wedge \\ & \text{Kind}(b) \wedge \forall x \neg \text{Schnitzt}(b, x) \end{aligned}$						
2	$\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))$						1: $\wedge$
3	$\forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))$						
4	$\forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y)))$						
5	$\text{Kind}(b)$						
6	$\forall x \neg \text{Schnitzt}(b, x)$						
7	$\text{Kauft}(b, k) \wedge \text{Kürbis}(k)$						2: $\exists(x \mapsto k)$
8	$\text{Kauft}(b, k)$						7: $\wedge$
9	$\text{Kürbis}(k)$						
10	$\neg \text{Schnitzt}(b, k)$						6: $\forall(x \mapsto k)$
11	$\neg \text{Kauft}(b, k) \vee \neg \text{Kürbis}(k) \vee \text{Isst}(b, k) \vee \text{Schnitzt}(b, k)$						3: $\forall(x \mapsto b, y \mapsto k)$
12	$\neg \text{Kind}(b) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(b, y))$						4: $\forall(x \mapsto b)$
13	$\neg \text{Kind}(b)$	$\forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(b, y))$					12: $\vee$
14	$\downarrow$ 5	$\neg \text{Kürbis}(k) \vee \neg \text{Isst}(b, k)$					13: $\forall(y \mapsto k)$
15		$\neg \text{Kürbis}(k)$	$\neg \text{Isst}(b, k)$				14: $\vee$
16		$\downarrow$ 9	$\neg \text{Kauft}(b, k)$	$\neg \text{Kürbis}(k)$	$\text{Isst}(b, k)$	$\text{Schnitzt}(b, k)$	11: $\vee$
			$\downarrow$ 8				

Beispiel: Halloween-Tableau

1	$\begin{aligned} & \exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x)) \quad \wedge \\ & \forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzzt}(x, y)) \quad \wedge \\ & \forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y))) \quad \wedge \\ & \text{Kind}(b) \wedge \forall x \neg \text{Schnitzzt}(b, x) \end{aligned}$						
2	$\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))$						1: $\wedge$
3	$\forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzzt}(x, y))$						
4	$\forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y)))$						
5	$\text{Kind}(b)$						
6	$\forall x \neg \text{Schnitzzt}(b, x)$						
7	$\text{Kauft}(b, k) \wedge \text{Kürbis}(k)$						2: $\exists(x \mapsto k)$
8	$\text{Kauft}(b, k)$						7: $\wedge$
9	$\text{Kürbis}(k)$						
10	$\neg \text{Schnitzzt}(b, k)$						6: $\forall(x \mapsto k)$
11	$\neg \text{Kauft}(b, k) \vee \neg \text{Kürbis}(k) \vee \text{Isst}(b, k) \vee \text{Schnitzzt}(b, k)$						3: $\forall(x \mapsto b, y \mapsto k)$
12	$\neg \text{Kind}(b) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(b, y))$						4: $\forall(x \mapsto b)$
13	$\neg \text{Kind}(b)$	$\forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(b, y))$					12: $\vee$
14	$\downarrow$ 5	$\neg \text{Kürbis}(k) \vee \neg \text{Isst}(b, k)$					13: $\forall(y \mapsto k)$
15		$\neg \text{Kürbis}(k)$	$\neg \text{Isst}(b, k)$				14: $\vee$
16		$\downarrow$ 9	$\neg \text{Kauft}(b, k)$	$\neg \text{Kürbis}(k)$	$\text{Isst}(b, k)$	$\text{Schnitzzt}(b, k)$	11: $\vee$
			$\downarrow$ 8	$\downarrow$ 9			

Beispiel: Halloween-Tableau

1	$\begin{aligned} & \exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x)) \quad \wedge \\ & \forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y)) \quad \wedge \\ & \forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y))) \quad \wedge \\ & \text{Kind}(b) \wedge \forall x \neg \text{Schnitzt}(b, x) \end{aligned}$						
2	$\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))$						1: $\wedge$
3	$\forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))$						
4	$\forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y)))$						
5	$\text{Kind}(b)$						
6	$\forall x \neg \text{Schnitzt}(b, x)$						
7	$\text{Kauft}(b, k) \wedge \text{Kürbis}(k)$						2: $\exists(x \mapsto k)$
8	$\text{Kauft}(b, k)$						7: $\wedge$
9	$\text{Kürbis}(k)$						
10	$\neg \text{Schnitzt}(b, k)$						6: $\forall(x \mapsto k)$
11	$\neg \text{Kauft}(b, k) \vee \neg \text{Kürbis}(k) \vee \text{Isst}(b, k) \vee \text{Schnitzt}(b, k)$						3: $\forall(x \mapsto b, y \mapsto k)$
12	$\neg \text{Kind}(b) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(b, y))$						4: $\forall(x \mapsto b)$
13	$\neg \text{Kind}(b)$	$\forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(b, y))$					12: $\vee$
14	$\downarrow$ 5	$\neg \text{Kürbis}(k) \vee \neg \text{Isst}(b, k)$					13: $\forall(y \mapsto k)$
15		$\neg \text{Kürbis}(k)$	$\neg \text{Isst}(b, k)$				14: $\vee$
16		$\downarrow$ 9	$\neg \text{Kauft}(b, k)$	$\neg \text{Kürbis}(k)$	$\text{Isst}(b, k)$	$\text{Schnitzt}(b, k)$	11: $\vee$
			$\downarrow$ 8	$\downarrow$ 9	$\downarrow$ 15		

Beispiel: Halloween-Tableau

1	$\begin{aligned} & \exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x)) \quad \wedge \\ & \forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y)) \quad \wedge \\ & \forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y))) \quad \wedge \\ & \text{Kind}(b) \wedge \forall x \neg \text{Schnitzt}(b, x) \end{aligned}$						
2	$\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))$						1: $\wedge$
3	$\forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))$						
4	$\forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y)))$						
5	$\text{Kind}(b)$						
6	$\forall x \neg \text{Schnitzt}(b, x)$						
7	$\text{Kauft}(b, k) \wedge \text{Kürbis}(k)$						2: $\exists(x \mapsto k)$
8	$\text{Kauft}(b, k)$						7: $\wedge$
9	$\text{Kürbis}(k)$						
10	$\neg \text{Schnitzt}(b, k)$						6: $\forall(x \mapsto k)$
11	$\neg \text{Kauft}(b, k) \vee \neg \text{Kürbis}(k) \vee \text{Isst}(b, k) \vee \text{Schnitzt}(b, k)$						3: $\forall(x \mapsto b, y \mapsto k)$
12	$\neg \text{Kind}(b) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(b, y))$						4: $\forall(x \mapsto b)$
13	$\neg \text{Kind}(b)$	$\forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(b, y))$					12: $\vee$
14	$\downarrow$ 5	$\neg \text{Kürbis}(k) \vee \neg \text{Isst}(b, k)$					13: $\forall(y \mapsto k)$
15		$\neg \text{Kürbis}(k)$	$\neg \text{Isst}(b, k)$				14: $\vee$
16		$\downarrow$ 9	$\neg \text{Kauft}(b, k)$	$\neg \text{Kürbis}(k)$	$\text{Isst}(b, k)$	$\text{Schnitzt}(b, k)$	11: $\vee$
			$\downarrow$ 8	$\downarrow$ 9	$\downarrow$ 15	$\downarrow$ 10	

Beispiel: Halloween-Tableau

1	$\begin{aligned} & \exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x)) \quad \wedge \\ & \forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y)) \quad \wedge \\ & \forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y))) \quad \wedge \\ & \text{Kind}(b) \wedge \forall x \neg \text{Schnitzt}(b, x) \end{aligned}$						
2	$\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))$						1: $\wedge$
3	$\forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))$						
4	$\forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y)))$						
5	$\text{Kind}(b)$						
6	$\forall x \neg \text{Schnitzt}(b, x)$						
7	$\text{Kauft}(b, k) \wedge \text{Kürbis}(k)$						2: $\exists(x \mapsto k)$
8	$\text{Kauft}(b, k)$						7: $\wedge$
9	$\text{Kürbis}(k)$						
10	$\neg \text{Schnitzt}(b, k)$						6: $\forall(x \mapsto k)$
11	$\neg \text{Kauft}(b, k) \vee \neg \text{Kürbis}(k) \vee \text{Isst}(b, k) \vee \text{Schnitzt}(b, k)$						3: $\forall(x \mapsto b, y \mapsto k)$
12	$\neg \text{Kind}(b) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(b, y))$						4: $\forall(x \mapsto b)$
13	$\neg \text{Kind}(b)$	$\forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(b, y))$					12: $\vee$
14	$\downarrow 5$	$\neg \text{Kürbis}(k) \vee \neg \text{Isst}(b, k)$					13: $\forall(y \mapsto k)$
15		$\neg \text{Kürbis}(k)$	$\neg \text{Isst}(b, k)$				14: $\vee$
16		$\downarrow 9$	$\neg \text{Kauft}(b, k)$	$\neg \text{Kürbis}(k)$	$\text{Isst}(b, k)$	$\text{Schnitzt}(b, k)$	11: $\vee$
			$\downarrow 8$	$\downarrow 9$	$\downarrow 15$	$\downarrow 10$	

■ φ ist unerfüllbar

Beispiel: Halloween-Tableau

1	$\begin{aligned} & \exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x)) \wedge \\ & \forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y)) \wedge \\ & \forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y))) \wedge \\ & \text{Kind}(b) \wedge \forall x \neg \text{Schnitzt}(b, x) \end{aligned}$						
2	$\exists x(\text{Kauft}(b, x) \wedge \text{Kürbis}(x))$						1: $\wedge$
3	$\forall x \forall y (\neg \text{Kauft}(x, y) \vee \neg \text{Kürbis}(y) \vee \text{Isst}(x, y) \vee \text{Schnitzt}(x, y))$						
4	$\forall x (\neg \text{Kind}(x) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(x, y)))$						
5	$\text{Kind}(b)$						
6	$\forall x \neg \text{Schnitzt}(b, x)$						
7	$\text{Kauft}(b, k) \wedge \text{Kürbis}(k)$						2: $\exists(x \mapsto k)$
8	$\text{Kauft}(b, k)$						7: $\wedge$
9	$\text{Kürbis}(k)$						
10	$\neg \text{Schnitzt}(b, k)$						6: $\forall(x \mapsto k)$
11	$\neg \text{Kauft}(b, k) \vee \neg \text{Kürbis}(k) \vee \text{Isst}(b, k) \vee \text{Schnitzt}(b, k)$						3: $\forall(x \mapsto b, y \mapsto k)$
12	$\neg \text{Kind}(b) \vee \forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(b, y))$						4: $\forall(x \mapsto b)$
13	$\neg \text{Kind}(b)$	$\forall y (\neg \text{Kürbis}(y) \vee \neg \text{Isst}(b, y))$					12: $\vee$
14	$\downarrow$ 5	$\neg \text{Kürbis}(k) \vee \neg \text{Isst}(b, k)$					13: $\forall(y \mapsto k)$
15		$\neg \text{Kürbis}(k)$	$\neg \text{Isst}(b, k)$				14: $\vee$
16		$\downarrow$ 9	$\neg \text{Kauft}(b, k)$	$\neg \text{Kürbis}(k)$	$\text{Isst}(b, k)$	$\text{Schnitzt}(b, k)$	11: $\vee$
			$\downarrow$ 8	$\downarrow$ 9	$\downarrow$ 15	$\downarrow$ 10	

- φ ist unerfüllbar
- Prämissen implizieren Konklusion

Übung 4.76

Verwenden Sie ein Tableau, um zu zeigen, dass Anna keine Mäuse hat.

$$\begin{aligned}\varphi = & \forall x(\neg H(x) \vee B(x)) \quad \wedge \\ & \forall x \forall y \forall z (\neg \text{Hat}(x, y) \vee \neg K(y) \vee \neg \text{Hat}(x, z) \vee \neg M(z)) \quad \wedge \\ & \forall x \forall y (\neg L(x) \vee \neg \text{Hat}(x, y) \vee \neg B(y)) \quad \wedge \\ & \exists x (\text{Hat}(a, x) \wedge (H(x) \vee K(x))) \quad \wedge \\ & L(a) \wedge \exists x (\text{Hat}(a, x) \wedge M(x))\end{aligned}$$

Peano-Axiomensystem der natürlichen Zahlen :

- 1 Null ist eine natürliche Zahl.



Giuseppe Peano
(1858–1932)

Peano-Axiomensystem der natürlichen Zahlen :

- 1 Null ist eine natürliche Zahl.
- 2 Jede natürliche Zahl hat eine natürliche Zahl als Nachfolger.



Giuseppe Peano
(1858–1932)

Peano-Axiomensystem der natürlichen Zahlen :

- 1 Null ist eine natürliche Zahl.
- 2 Jede natürliche Zahl hat eine natürliche Zahl als Nachfolger.
- 3 Null ist nicht der Nachfolger einer natürlichen Zahl.



Giuseppe Peano
(1858–1932)

Peano-Axiomensystem der natürlichen Zahlen :

- 1 Null ist eine natürliche Zahl.
- 2 Jede natürliche Zahl hat eine natürliche Zahl als Nachfolger.
- 3 Null ist nicht der Nachfolger einer natürlichen Zahl.
- 4 Wenn natürliche Zahlen denselben Nachfolger haben, handelt es sich um dieselbe Zahl.



Giuseppe Peano
(1858–1932)

Peano-Axiomensystem der natürlichen Zahlen :

- 1 Null ist eine natürliche Zahl.
- 2 Jede natürliche Zahl hat eine natürliche Zahl als Nachfolger.
- 3 Null ist nicht der Nachfolger einer natürlichen Zahl.
- 4 Wenn natürliche Zahlen denselben Nachfolger haben, handelt es sich um dieselbe Zahl.
- 5 Die natürlichen Zahlen sind die kleinste Menge, die Null und den Nachfolger jedes Elements enthält.

Alternativ:

Jede Menge, die Null und den Nachfolger jedes Elements enthält, ist eine Obermenge der natürlichen Zahlen.



Giuseppe Peano
(1858–1932)

Übung 4.77

1 Formalisieren Sie die Peano-Axiome 1–4 in der Prädikatenlogik.

■ Verwenden Sie die folgende Signatur:

■ Prädikat $N^{(1)}(x)$: „ x ist eine natürliche Zahl“

■ Prädikat $=^{(2)}(x, y)$: „ x ist gleich y “

Für Gleichheit gehen wir davon aus, dass die korrekte Interpretation fest vorgegeben ist.

■ Funktionssymbol $z^{(0)}$: die Konstante „Null“

■ Funktionssymbol $s^{(1)}(x)$: „Nachfolger von x “

■ Axiom 5 kann in der PL 1. Stufe nicht formalisiert werden.

2 Zeigen Sie, dass die Axiome 1–4 **widerspruchsfrei** sind.

■ Zeigen Sie die Erfüllbarkeit der Konjunktion der Axiome.

■ Resolution und Tableaualgorithmus terminieren nicht immer.

3 Zeigen Sie, dass die Axiome 1–4 **unabhängig** sind.

■ Zeigen Sie für jedes Axiom, dass seine Negation zusammen mit den anderen Axiomen erfüllbar ist; z. B. $1 \wedge 2 \wedge \neg 3 \wedge 4$.

■ Verwenden Sie die vom Tableau-Algorithmus erzeugten Modelle, um sich zu verdeutlichen, dass diese Modelle nicht isomorph zu $\mathbb{N}$ sind.

- Problem: Effizienz abhängig von Auswahl der Terme in $\forall$ -Regel

- Problem: Effizienz abhängig von Auswahl der Terme in $\forall$ -Regel
 - in Beispielen: immer sofort „richtiger“ Term

- Problem: Effizienz abhängig von Auswahl der Terme in $\forall$ -Regel
 - in Beispielen: immer sofort „richtiger“ Term
 - Programm muss alle Möglichkeiten testen

- Problem: Effizienz abhängig von Auswahl der Terme in $\forall$ -Regel
 - in Beispielen: immer sofort „richtiger“ Term
 - Programm muss alle Möglichkeiten testen
 - Signatur mit 4 Konstantensymbolen und Formel mit 5 universell quantifizierten Variablen $\rightsquigarrow$
 $4^5 = 1024$ mögliche Regelanwendungen

- Problem: Effizienz abhängig von Auswahl der Terme in $\forall$ -Regel
 - in Beispielen: immer sofort „richtiger“ Term
 - Programm muss alle Möglichkeiten testen
 - Signatur mit 4 Konstantensymbolen und Formel mit 5 universell quantifizierten Variablen $\rightsquigarrow$
 $4^5 = 1024$ mögliche Regelanwendungen
 - Signatur mit Funktionssymbolen $\rightsquigarrow$ unendlich viele Möglichkeiten

- Problem: Effizienz abhängig von Auswahl der Terme in $\forall$ -Regel
 - in Beispielen: immer sofort „richtiger“ Term
 - Programm muss alle Möglichkeiten testen
 - Signatur mit 4 Konstantensymbolen und Formel mit 5 universell quantifizierten Variablen $\rightsquigarrow$
 $4^5 = 1024$ mögliche Regelanwendungen
 - Signatur mit Funktionssymbolen $\rightsquigarrow$ unendlich viele Möglichkeiten
- Abhilfe:

- Problem: Effizienz abhängig von Auswahl der Terme in $\forall$ -Regel
 - in Beispielen: immer sofort „richtiger“ Term
 - Programm muss alle Möglichkeiten testen
 - Signatur mit 4 Konstantensymbolen und Formel mit 5 universell quantifizierten Variablen $\rightsquigarrow$ $4^5 = 1024$ mögliche Regelanwendungen
 - Signatur mit Funktionssymbolen $\rightsquigarrow$ unendlich viele Möglichkeiten
- Abhilfe:
 - Heuristiken: Beginne mit Termen, die bereits vorkommen

- Problem: Effizienz abhängig von Auswahl der Terme in $\forall$ -Regel
 - in Beispielen: immer sofort „richtiger“ Term
 - Programm muss alle Möglichkeiten testen
 - Signatur mit 4 Konstantensymbolen und Formel mit 5 universell quantifizierten Variablen $\rightsquigarrow 4^5 = 1024$ mögliche Regelanwendungen
 - Signatur mit Funktionssymbolen $\rightsquigarrow$ unendlich viele Möglichkeiten
- Abhilfe:
 - Heuristiken: Beginne mit Termen, die bereits vorkommen
 - Nicht in jedem Fall ausreichend: $\forall x P(x) \wedge \forall x \neg P(x)$

- Problem: Effizienz abhängig von Auswahl der Terme in $\forall$ -Regel
 - in Beispielen: immer sofort „richtiger“ Term
 - Programm muss alle Möglichkeiten testen
 - Signatur mit 4 Konstantensymbolen und Formel mit 5 universell quantifizierten Variablen $\rightsquigarrow 4^5 = 1024$ mögliche Regelanwendungen
 - Signatur mit Funktionssymbolen $\rightsquigarrow$ unendlich viele Möglichkeiten
- Abhilfe:
 - Heuristiken: Beginne mit Termen, die bereits vorkommen
 - Nicht in jedem Fall ausreichend: $\forall x P(x) \wedge \forall x \neg P(x)$
 - Modifizierter Algorithmus: Tableau-Algorithmus mit **Unifikation**

- Problem: Effizienz abhängig von Auswahl der Terme in $\forall$ -Regel
 - in Beispielen: immer sofort „richtiger“ Term
 - Programm muss alle Möglichkeiten testen
 - Signatur mit 4 Konstantensymbolen und Formel mit 5 universell quantifizierten Variablen $\rightsquigarrow$ $4^5 = 1024$ mögliche Regelanwendungen
 - Signatur mit Funktionssymbolen $\rightsquigarrow$ unendlich viele Möglichkeiten
- Abhilfe:
 - Heuristiken: Beginne mit Termen, die bereits vorkommen
 - Nicht in jedem Fall ausreichend: $\forall x P(x) \wedge \forall x \neg P(x)$
 - Modifizierter Algorithmus: Tableau-Algorithmus mit **Unifikation**
 - Verzögere Substitution

- Problem: Effizienz abhängig von Auswahl der Terme in $\forall$ -Regel
 - in Beispielen: immer sofort „richtiger“ Term
 - Programm muss alle Möglichkeiten testen
 - Signatur mit 4 Konstantensymbolen und Formel mit 5 universell quantifizierten Variablen $\rightsquigarrow 4^5 = 1024$ mögliche Regelanwendungen
 - Signatur mit Funktionssymbolen $\rightsquigarrow$ unendlich viele Möglichkeiten
- Abhilfe:
 - Heuristiken: Beginne mit Termen, die bereits vorkommen
 - Nicht in jedem Fall ausreichend: $\forall x P(x) \wedge \forall x \neg P(x)$
 - Modifizierter Algorithmus: Tableau-Algorithmus mit **Unifikation**
 - Verzögere Substitution
 - Verwende Unifikation zum Finden einer Substitution, die Clash erzeugt

$\forall^u$ Wenn $\forall x \varphi \in S$
dann $S := S \cup \sigma(\varphi)$ mit $\sigma = \{x \mapsto y\}$ für eine **neue** Variable y

- $\forall^u$ Wenn $\forall x\varphi \in S$
dann $S := S \cup \sigma(\varphi)$ mit $\sigma = \{x \mapsto y\}$ für eine **neue** Variable y
■ y ist **freie** Variable!

- $\forall^u$ Wenn $\forall x\varphi \in S$
dann $S := S \cup \sigma(\varphi)$ mit $\sigma = \{x \mapsto y\}$ für eine **neue** Variable y
- y ist **freie** Variable!
 - immer anwendbar

$\forall^u$ Wenn $\forall x\varphi \in S$

dann $S := S \cup \sigma(\varphi)$ mit $\sigma = \{x \mapsto y\}$ für eine **neue** Variable y

- y ist **freie** Variable!
- immer anwendbar

$\exists^u$ Wenn $\exists x\varphi \in S$

und es keine Substitution $\sigma = \{x \mapsto t\}$ für einen Term t gibt mit $\sigma(\varphi) \in S$

und φ enthält die **freien** Variablen $y_1, \dots, y_n$

dann $S := S \cup \sigma(\varphi)$ mit $\sigma = \{x \mapsto f(y_1, \dots, y_n)\}$ für ein **neues** n -stelliges f

$\forall^u$ Wenn $\forall x\varphi \in S$

dann $S := S \cup \sigma(\varphi)$ mit $\sigma = \{x \mapsto y\}$ für eine **neue** Variable y

- y ist **freie** Variable!
- immer anwendbar

$\exists^u$ Wenn $\exists x\varphi \in S$

und es keine Substitution $\sigma = \{x \mapsto t\}$ für einen Term t gibt mit $\sigma(\varphi) \in S$

und φ enthält die **freien** Variablen $y_1, \dots, y_n$

dann $S := S \cup \sigma(\varphi)$ mit $\sigma = \{x \mapsto f(y_1, \dots, y_n)\}$ für ein **neues** n -stelliges f

- Nachteil: Komplexe Skolem-Funktionen nötig

- $\forall^u$ Wenn $\forall x\varphi \in S$
dann $S := S \cup \sigma(\varphi)$ mit $\sigma = \{x \mapsto y\}$ für eine **neue** Variable y
- y ist **freie** Variable!
 - immer anwendbar
- $\exists^u$ Wenn $\exists x\varphi \in S$
und es keine Substitution $\sigma = \{x \mapsto t\}$ für einen Term t gibt mit $\sigma(\varphi) \in S$
und φ enthält die **freien** Variablen $y_1, \dots, y_n$
dann $S := S \cup \sigma(\varphi)$ mit $\sigma = \{x \mapsto f(y_1, \dots, y_n)\}$ für ein **neues** n -stelliges f
- Nachteil: Komplexe Skolem-Funktionen nötig
- S** Wenn es in einer Spalte Formeln φ und ψ gibt
und einen Unifikator σ für die **freien** Variablen mit $\sigma(\varphi) = \neg\sigma(\psi)$
dann wende σ auf gesamtes Tableau an

$\forall^u$ Wenn $\forall x \varphi \in S$

dann $S := S \cup \sigma(\varphi)$ mit $\sigma = \{x \mapsto y\}$ für eine **neue** Variable y

- y ist **freie** Variable!
- immer anwendbar

$\exists^u$ Wenn $\exists x \varphi \in S$

und es keine Substitution $\sigma = \{x \mapsto t\}$ für einen Term t gibt mit $\sigma(\varphi) \in S$

und φ enthält die **freien** Variablen $y_1, \dots, y_n$

dann $S := S \cup \sigma(\varphi)$ mit $\sigma = \{x \mapsto f(y_1, \dots, y_n)\}$ für ein **neues** n -stelliges f

- Nachteil: Komplexe Skolem-Funktionen nötig

S Wenn es in einer Spalte Formeln φ und ψ gibt

und einen Unifikator σ für die **freien** Variablen mit $\sigma(\varphi) = \neg \sigma(\psi)$

dann wende σ auf gesamtes Tableau an

- **nicht nur auf aktuelle Spalte!**

$\forall^u$ Wenn $\forall x \varphi \in S$

dann $S := S \cup \sigma(\varphi)$ mit $\sigma = \{x \mapsto y\}$ für eine **neue** Variable y

- y ist **freie** Variable!
- immer anwendbar

$\exists^u$ Wenn $\exists x \varphi \in S$

und es keine Substitution $\sigma = \{x \mapsto t\}$ für einen Term t gibt mit $\sigma(\varphi) \in S$

und φ enthält die **freien** Variablen $y_1, \dots, y_n$

dann $S := S \cup \sigma(\varphi)$ mit $\sigma = \{x \mapsto f(y_1, \dots, y_n)\}$ für ein **neues** n -stelliges f

- Nachteil: Komplexe Skolem-Funktionen nötig

S Wenn es in einer Spalte Formeln φ und ψ gibt

und einen Unifikator σ für die **freien** Variablen mit $\sigma(\varphi) = \neg \sigma(\psi)$

dann wende σ auf gesamtes Tableau an

- **nicht nur auf aktuelle Spalte!**
- gleiches Ergebnis, wie wenn man σ von Anfang an verwendet hätte

$\forall^u$ Wenn $\forall x \varphi \in S$

dann $S := S \cup \sigma(\varphi)$ mit $\sigma = \{x \mapsto y\}$ für eine **neue** Variable y

- y ist **freie** Variable!
- immer anwendbar

$\exists^u$ Wenn $\exists x \varphi \in S$

und es keine Substitution $\sigma = \{x \mapsto t\}$ für einen Term t gibt mit $\sigma(\varphi) \in S$

und φ enthält die **freien** Variablen $y_1, \dots, y_n$

dann $S := S \cup \sigma(\varphi)$ mit $\sigma = \{x \mapsto f(y_1, \dots, y_n)\}$ für ein **neues** n -stelliges f

- Nachteil: Komplexe Skolem-Funktionen nötig

S Wenn es in einer Spalte Formeln φ und ψ gibt

und einen Unifikator σ für die **freien** Variablen mit $\sigma(\varphi) = \neg \sigma(\psi)$

dann wende σ auf gesamtes Tableau an

- **nicht nur auf aktuelle Spalte!**
- gleiches Ergebnis, wie wenn man σ von Anfang an verwendet hätte
- freie Variablen müssen überall ersetzt werden

Beispiel 4.78

$$1 \quad || \quad \forall x \exists y (M(x, y) \vee P(x, y)) \wedge \forall z (\neg M(c, z) \wedge \neg P(c, z)) \quad ||$$

Beispiel 4.78

1	$\forall x \exists y (M(x, y) \vee P(x, y)) \wedge \forall z (\neg M(c, z) \wedge \neg P(c, z))$	
2	$\forall x \exists y (M(x, y) \vee P(x, y))$	1: $\wedge$

Beispiel 4.78

1	$\forall x \exists y (M(x, y) \vee P(x, y)) \wedge \forall z (\neg M(c, z) \wedge \neg P(c, z))$	
2	$\forall x \exists y (M(x, y) \vee P(x, y))$	1: $\wedge$
3	$\forall z (\neg M(c, z) \wedge \neg P(c, z))$	

Beispiel 4.78

1	$\forall x \exists y (M(x, y) \vee P(x, y)) \wedge \forall z (\neg M(c, z) \wedge \neg P(c, z))$	
2	$\forall x \exists y (M(x, y) \vee P(x, y))$	1: $\wedge$
3	$\forall z (\neg M(c, z) \wedge \neg P(c, z))$	
4	$\exists y (M(\textcolor{teal}{r}, y) \vee P(\textcolor{teal}{r}, y))$	2: $\forall^u (x \mapsto r)$

Beispiel 4.78

1	$\forall x \exists y (M(x, y) \vee P(x, y)) \wedge \forall z (\neg M(c, z) \wedge \neg P(c, z))$	
2	$\forall x \exists y (M(x, y) \vee P(x, y))$	1: $\wedge$
3	$\forall z (\neg M(c, z) \wedge \neg P(c, z))$	
4	$\exists y (M(r, y) \vee P(r, y))$	2: $\forall^u (x \mapsto r)$
5	$M(r, f(r)) \vee P(r, f(r))$	4: $\exists^u (y \mapsto f(r))$

Beispiel 4.78

1	$\forall x \exists y (M(x, y) \vee P(x, y)) \wedge \forall z (\neg M(c, z) \wedge \neg P(c, z))$	
2	$\forall x \exists y (M(x, y) \vee P(x, y))$	1: $\wedge$
3	$\forall z (\neg M(c, z) \wedge \neg P(c, z))$	
4	$\exists y (M(r, y) \vee P(r, y))$	2: $\forall^u (x \mapsto r)$
5	$M(r, f(r)) \vee P(r, f(r))$	4: $\exists^u (y \mapsto f(r))$
6	$\neg M(c, s) \wedge \neg P(c, s)$	3: $\forall^u (z \mapsto s)$

Beispiel 4.78

1	$\forall x \exists y (M(x, y) \vee P(x, y)) \wedge \forall z (\neg M(c, z) \wedge \neg P(c, z))$	
2	$\forall x \exists y (M(x, y) \vee P(x, y))$	1: $\wedge$
3	$\forall z (\neg M(c, z) \wedge \neg P(c, z))$	
4	$\exists y (M(r, y) \vee P(r, y))$	2: $\forall^u (x \mapsto r)$
5	$M(r, f(r)) \vee P(r, f(r))$	4: $\exists^u (y \mapsto f(r))$
6	$\neg M(c, s) \wedge \neg P(c, s)$	3: $\forall^u (z \mapsto s)$
7	$\neg M(c, s)$	6: $\wedge$
8	$\neg P(c, s)$	

Beispiel 4.78

1	$\forall x \exists y (M(x, y) \vee P(x, y)) \wedge \forall z (\neg M(c, z) \wedge \neg P(c, z))$		
2	$\forall x \exists y (M(x, y) \vee P(x, y))$		1: $\wedge$
3	$\forall z (\neg M(c, z) \wedge \neg P(c, z))$		
4	$\exists y (M(r, y) \vee P(r, y))$		2: $\forall^u (x \mapsto r)$
5	$M(r, f(r)) \vee P(r, f(r))$		4: $\exists^u (y \mapsto f(r))$
6	$\neg M(c, s) \wedge \neg P(c, s)$		3: $\forall^u (z \mapsto s)$
7	$\neg M(c, s)$		6: $\wedge$
8	$\neg P(c, s)$		
9	$M(r, f(r))$	$P(r, f(r))$	5: $\vee$
			S $\{r \mapsto c, s \mapsto f(c)\}$

Beispiel 4.78

1	$\forall x \exists y (M(x, y) \vee P(x, y)) \wedge \forall z (\neg M(c, z) \wedge \neg P(c, z))$		
2	$\forall x \exists y (M(x, y) \vee P(x, y))$		1: $\wedge$
3	$\forall z (\neg M(c, z) \wedge \neg P(c, z))$		
4	$\exists y (M(c, y) \vee P(c, y))$		2: $\forall^u (x \mapsto r)$
5	$M(c, f(c)) \vee P(c, f(c))$		4: $\exists^u (y \mapsto f(r))$
6	$\neg M(c, s) \wedge \neg P(c, s)$		3: $\forall^u (z \mapsto s)$
7	$\neg M(c, s)$		6: $\wedge$
8	$\neg P(c, s)$		
9	$M(c, f(c))$	$P(c, f(c))$	5: $\vee$
			S $\{r \mapsto c, s \mapsto f(c)\}$

Beispiel 4.78

1	$\forall x \exists y (M(x, y) \vee P(x, y)) \wedge \forall z (\neg M(c, z) \wedge \neg P(c, z))$		
2	$\forall x \exists y (M(x, y) \vee P(x, y))$		1: $\wedge$
3	$\forall z (\neg M(c, z) \wedge \neg P(c, z))$		
4	$\exists y (M(c, y) \vee P(c, y))$		2: $\forall^u (x \mapsto r)$
5	$M(c, f(c)) \vee P(c, f(c))$		4: $\exists^u (y \mapsto f(r))$
6	$\neg M(c, f(c)) \wedge \neg P(c, f(c))$		3: $\forall^u (z \mapsto s)$
7	$\neg M(c, f(c))$		6: $\wedge$
8	$\neg P(c, f(c))$		
9	$M(c, f(c))$	$P(c, f(c))$	5: $\vee$
			S $\{r \mapsto c, s \mapsto f(c)\}$

Beispiel 4.78

1	$\forall x \exists y (M(x, y) \vee P(x, y)) \wedge \forall z (\neg M(c, z) \wedge \neg P(c, z))$		
2	$\forall x \exists y (M(x, y) \vee P(x, y))$		1: $\wedge$
3	$\forall z (\neg M(c, z) \wedge \neg P(c, z))$		
4	$\exists y (M(c, y) \vee P(c, y))$		2: $\forall^u (x \mapsto r)$
5	$M(c, f(c)) \vee P(c, f(c))$		4: $\exists^u (y \mapsto f(r))$
6	$\neg M(c, f(c)) \wedge \neg P(c, f(c))$		3: $\forall^u (z \mapsto s)$
7	$\neg M(c, f(c))$		6: $\wedge$
8	$\neg P(c, f(c))$		
9	$M(c, f(c))$	$P(c, f(c))$	5: $\vee$
	$\downarrow 7$		S $\{r \mapsto c, s \mapsto f(c)\}$

Beispiel 4.78

1	$\forall x \exists y (M(x, y) \vee P(x, y)) \wedge \forall z (\neg M(c, z) \wedge \neg P(c, z))$		
2	$\forall x \exists y (M(x, y) \vee P(x, y))$		1: $\wedge$
3	$\forall z (\neg M(c, z) \wedge \neg P(c, z))$		
4	$\exists y (M(c, y) \vee P(c, y))$		2: $\forall^u (x \mapsto r)$
5	$M(c, f(c)) \vee P(c, f(c))$		4: $\exists^u (y \mapsto f(r))$
6	$\neg M(c, f(c)) \wedge \neg P(c, f(c))$		3: $\forall^u (z \mapsto s)$
7	$\neg M(c, f(c))$		6: $\wedge$
8	$\neg P(c, f(c))$		
9	$M(c, f(c))$	$P(c, f(c))$	5: $\vee$
	$\not\downarrow 7$	$\not\downarrow 8$	S $\{r \mapsto c, s \mapsto f(c)\}$

Mit den Übungsaufgaben auf der Seite

`https://www.cs.utexas.edu/users/novak/reso.html`

können Sie die Themen

- Formalisierung
- Normalformen
- Resolution
- Tableaus

im Selbststudium vertiefen.

Syntax Variablen, Prädikate, Funktionssymbole, Quantoren
Terme, Atome, komplexe Formeln
Modellierung der inneren Struktur von Aussagen

Syntax Variablen, Prädikate, Funktionssymbole, Quantoren
Terme, Atome, komplexe Formeln
Modellierung der inneren Struktur von Aussagen

Interpretation Domäne, Funktionen, Relationen

Syntax Variablen, Prädikate, Funktionssymbole, Quantoren
Terme, Atome, komplexe Formeln
Modellierung der inneren Struktur von Aussagen

Interpretation Domäne, Funktionen, Relationen

Schlussfolgerung korrekt, vollständig
nicht terminierend $\rightsquigarrow$ unentscheidbar

Syntax Variablen, Prädikate, Funktionssymbole, Quantoren
Terme, Atome, komplexe Formeln
Modellierung der inneren Struktur von Aussagen

Interpretation Domäne, Funktionen, Relationen

Schlussfolgerung korrekt, vollständig
nicht terminierend $\rightsquigarrow$ unentscheidbar

Resolution NNF, SF, Unifikation, Faktorisierung

Syntax Variablen, Prädikate, Funktionssymbole, Quantoren
Terme, Atome, komplexe Formeln
Modellierung der inneren Struktur von Aussagen

Interpretation Domäne, Funktionen, Relationen

Schlussfolgerung korrekt, vollständig
nicht terminierend $\rightsquigarrow$ unentscheidbar

Resolution NNF, SF, Unifikation, Faktorisierung

Tableau $\exists/\forall$ -Regel, Anwendbarkeit

1. Einführung

2. Mengen

3. Aussagenlogik

4. Prädikatenlogik

5. Prolog

5.1 Terme

5.2 Fakten, Regeln und Anfragen

5.3 Unifikation, Termgleichheit und Arithmetik

5.4 Horn-Resolution und Suchbäume

5.5 Rekursion

5.6 Listen

5.7 Der Cut

- entwickelt 1972 in Frankreich von Alain Colmerauer



Alain Colmerauer
(1941–2017)

- entwickelt 1972 in Frankreich von Alain Colmerauer
- Programmation en **logique**



Alain Colmerauer
(1941–2017)

- entwickelt 1972 in Frankreich von Alain Colmerauer
- **P**rogrammation en **l**ogique
- ISO-Standard 1995



Alain Colmerauer
(1941–2017)

- entwickelt 1972 in Frankreich von Alain Colmerauer
- **P**rogrammation en **l**ogique
- ISO-Standard 1995
- **D**eklarative Programmiersprache



Alain Colmerauer
(1941–2017)

- entwickelt 1972 in Frankreich von Alain Colmerauer
- **P**rogrammation en **l**ogique
- ISO-Standard 1995
- **D**eklarative Programmiersprache
 - „Was gilt?“



Alain Colmerauer
(1941–2017)

- entwickelt 1972 in Frankreich von Alain Colmerauer
- **P**rogrammation en **l**ogique
- ISO-Standard 1995
- **D**eklarative Programmiersprache
 - „Was gilt?“
 - Beschreibung der Domäne



Alain Colmerauer
(1941–2017)

- entwickelt 1972 in Frankreich von Alain Colmerauer
- Programmation en **logique**
- ISO-Standard 1995
- **Deklarative** Programmiersprache
 - „Was gilt?“
 - Beschreibung der Domäne
 - Beantwortung von Anfragen



Alain Colmerauer
(1941–2017)

- entwickelt 1972 in Frankreich von Alain Colmerauer
- **P**rogrammation en **l**ogique
- ISO-Standard 1995
- **D**eklarative Programmiersprache
 - „Was gilt?“
 - Beschreibung der Domäne
 - Beantwortung von Anfragen
 - Verwendung logischer Schlussfolgerungsverfahren



Alain Colmerauer
(1941–2017)

- entwickelt 1972 in Frankreich von Alain Colmerauer
- **P**rogrammation en **l**ogique
- ISO-Standard 1995
- **D**eklarative Programmiersprache
 - „Was gilt?“
 - Beschreibung der Domäne
 - Beantwortung von Anfragen
 - Verwendung logischer Schlussfolgerungsverfahren
- Gegenbegriff: **I**mperative Programmiersprache



Alain Colmerauer
(1941–2017)

- entwickelt 1972 in Frankreich von Alain Colmerauer
- **P**rogrammation en **l**ogique
- ISO-Standard 1995
- **D**eklarative Programmiersprache
 - „Was gilt?“
 - Beschreibung der Domäne
 - Beantwortung von Anfragen
 - Verwendung logischer Schlussfolgerungsverfahren
- Gegenbegriff: **I**mperative Programmiersprache
 - „Was ist zu tun?“



Alain Colmerauer
(1941–2017)

- entwickelt 1972 in Frankreich von Alain Colmerauer
- **P**rogrammation en **l**ogique
- ISO-Standard 1995
- **D**eklarative Programmiersprache
 - „Was gilt?“
 - Beschreibung der Domäne
 - Beantwortung von Anfragen
 - Verwendung logischer Schlussfolgerungsverfahren
- Gegenbegriff: **I**mperative Programmiersprache
 - „Was ist zu tun?“
 - Beschreibung eines Vorgehens (Algorithmus)



Alain Colmerauer
(1941–2017)

- entwickelt 1972 in Frankreich von Alain Colmerauer
- **Programmation en logique**
- ISO-Standard 1995
- **Deklarative** Programmiersprache
 - „Was gilt?“
 - Beschreibung der Domäne
 - Beantwortung von Anfragen
 - Verwendung logischer Schlussfolgerungsverfahren
- Gegenbegriff: **Imperative** Programmiersprache
 - „Was ist zu tun?“
 - Beschreibung eines Vorgehens (Algorithmus)
- ermöglicht sehr kompakte Programme



Alain Colmerauer
(1941–2017)

- entwickelt 1972 in Frankreich von Alain Colmerauer
- **Programmation en logique**
- ISO-Standard 1995
- **Deklarative** Programmiersprache
 - „Was gilt?“
 - Beschreibung der Domäne
 - Beantwortung von Anfragen
 - Verwendung logischer Schlussfolgerungsverfahren
- Gegenbegriff: **Imperative** Programmiersprache
 - „Was ist zu tun?“
 - Beschreibung eines Vorgehens (Algorithmus)
- ermöglicht sehr kompakte Programme
- erfordert andere Denkweise



Alain Colmerauer
(1941–2017)

- entwickelt 1972 in Frankreich von Alain Colmerauer
- **Programmation en logique**
- ISO-Standard 1995
- **Deklarative** Programmiersprache
 - „Was gilt?“
 - Beschreibung der Domäne
 - Beantwortung von Anfragen
 - Verwendung logischer Schlussfolgerungsverfahren
- Gegenbegriff: **Imperative** Programmiersprache
 - „Was ist zu tun?“
 - Beschreibung eines Vorgehens (Algorithmus)
- ermöglicht sehr kompakte Programme
- erfordert andere Denkweise
- Anwendungsbereiche:



Alain Colmerauer
(1941–2017)

- entwickelt 1972 in Frankreich von Alain Colmerauer
- **Programmation en logique**
- ISO-Standard 1995
- **Deklarative** Programmiersprache
 - „Was gilt?“
 - Beschreibung der Domäne
 - Beantwortung von Anfragen
 - Verwendung logischer Schlussfolgerungsverfahren
- Gegenbegriff: **Imperative** Programmiersprache
 - „Was ist zu tun?“
 - Beschreibung eines Vorgehens (Algorithmus)
- ermöglicht sehr kompakte Programme
- erfordert andere Denkweise
- Anwendungsbereiche:
 - Künstliche Intelligenz



Alain Colmerauer
(1941–2017)

- entwickelt 1972 in Frankreich von Alain Colmerauer
- **Programmation en logique**
- ISO-Standard 1995
- **Deklarative** Programmiersprache
 - „Was gilt?“
 - Beschreibung der Domäne
 - Beantwortung von Anfragen
 - Verwendung logischer Schlussfolgerungsverfahren
- Gegenbegriff: **Imperative** Programmiersprache
 - „Was ist zu tun?“
 - Beschreibung eines Vorgehens (Algorithmus)
- ermöglicht sehr kompakte Programme
- erfordert andere Denkweise
- Anwendungsbereiche:
 - Künstliche Intelligenz
 - Spracherkennung



Alain Colmerauer
(1941–2017)

Beispiel 5.1 (Imperatives Kochrezept)

- 1 3 Knoblauchzehen schälen und pressen
- 2 1 Chilischote kleinschneiden
- 3 Knoblauch und Chili in 50ml Olivenöl andünsten
- 4 300g Spaghetti 8 Minuten kochen
- 5 Spaghetti und Öl mischen

Beispiel 5.1 (Imperatives Kochrezept)

- 1 3 Knoblauchzehen schälen und pressen
- 2 1 Chilischote kleinschneiden
- 3 Knoblauch und Chili in 50ml Olivenöl andünsten
- 4 300g Spaghetti 8 Minuten kochen
- 5 Spaghetti und Öl mischen

Beispiel 5.2 (Deklaratives Kochrezept)

- Spaghetti aglio, olio e peperoncino besteht aus einer Mischung von 300g gekochten Spaghetti und Aglio-Olio-Peperoncino-Sauce
- Spaghetti sind gekocht, wenn sie 8 Minuten in kochendem Wasser gelegen haben.
- Aglio-Olio-Peperoncino-Sauce entsteht durch das Andünsten von 3 gepressten Knoblauchzehen und 1 geschnittenen Chilischote in 50ml Olivenöl

- Beschreibung der Domäne in der Wissensbank

- Beschreibung der **Domäne** in der **Wissensbank**
 - einzelne Aussagen formuliert als **Klauseln**

- Beschreibung der Domäne in der Wissensbank
 - einzelne Aussagen formuliert als Klauseln
 - Programmier-Paradigma: Rekursion

- Beschreibung der **Domäne** in der **Wissensbank**
 - einzelne Aussagen formuliert als **Klauseln**
 - Programmier-Paradigma: **Rekursion**
 - keine Kontrollstrukturen wie `if`, `while`, `for`

- Beschreibung der **Domäne** in der **Wissensbank**
 - einzelne Aussagen formuliert als **Klauseln**
 - Programmier-Paradigma: **Rekursion**
 - keine Kontrollstrukturen wie `if`, `while`, `for`
 - kein `goto`

- Beschreibung der **Domäne** in der **Wissensbank**
 - einzelne Aussagen formuliert als **Klauseln**
 - Programmier-Paradigma: **Rekursion**
 - keine Kontrollstrukturen wie `if`, `while`, `for`
 - kein `goto`
- **Interpreter** beantwortet **Anfragen** zur Domäne

- Beschreibung der **Domäne** in der **Wissensbank**
 - einzelne Aussagen formuliert als **Klauseln**
 - Programmier-Paradigma: **Rekursion**
 - keine Kontrollstrukturen wie `if`, `while`, `for`
 - kein `goto`
- **Interpreter** beantwortet **Anfragen** zur Domäne
 - Ableitung von neuem Wissen aus Wissensbank

- Beschreibung der Domäne in der Wissensbank
 - einzelne Aussagen formuliert als Klauseln
 - Programmier-Paradigma: Rekursion
 - keine Kontrollstrukturen wie `if`, `while`, `for`
 - kein `goto`
- Interpreter beantwortet Anfragen zur Domäne
 - Ableitung von neuem Wissen aus Wissensbank
 - Methode:

- Beschreibung der Domäne in der Wissensbank
 - einzelne Aussagen formuliert als Klauseln
 - Programmier-Paradigma: Rekursion
 - keine Kontrollstrukturen wie if, while, for
 - kein goto
- Interpreter beantwortet Anfragen zur Domäne
 - Ableitung von neuem Wissen aus Wissensbank
 - Methode:
 - Unifikation von Anfragen und Klauseln der Wissensbank

- Beschreibung der **Domäne** in der **Wissensbank**
 - einzelne Aussagen formuliert als **Klauseln**
 - Programmier-Paradigma: **Rekursion**
 - keine Kontrollstrukturen wie **if**, **while**, **for**
 - kein **goto**
- **Interpreter** beantwortet **Anfragen** zur Domäne
 - Ableitung von neuem Wissen aus Wissensbank
 - Methode:
 - **Unifikation** von Anfragen und Klauseln der Wissensbank
 - **Resolution**

- enthält Beschreibung der Welt

- enthält Beschreibung der Welt
- ist das Prolog-„Programm“

- enthält Beschreibung der Welt
- ist das Prolog-„Programm“
- besteht aus prädikatenlogischen Formeln ([Klauseln](#))

- enthält Beschreibung der Welt
- ist das Prolog-„Programm“
- besteht aus prädikatenlogischen Formeln ([Klauseln](#))

- enthält Beschreibung der Welt
- ist das Prolog-„Programm“
- besteht aus prädikatenlogischen Formeln ([Klauseln](#))

Beispiel 5.3

Prolog	Prädikatenlogik	natürliche Sprache
<code>party.</code>	P	„Es gibt eine Party.“

- enthält Beschreibung der Welt
- ist das Prolog-„Programm“
- besteht aus prädikatenlogischen Formeln ([Klauseln](#))

Beispiel 5.3

Prolog	Prädikatenlogik	natürliche Sprache
<code>party.</code>	P	„Es gibt eine Party.“
<code>woman(mia).</code>	$W(m)$	„Mia ist eine Frau.“

- enthält Beschreibung der Welt
- ist das Prolog-„Programm“
- besteht aus prädikatenlogischen Formeln (**Klauseln**)

Beispiel 5.3

Prolog	Prädikatenlogik	natürliche Sprache
<code>party.</code>	P	„Es gibt eine Party.“
<code>woman(mia).</code>	$W(m)$	„Mia ist eine Frau.“
<code>human(X) :- woman(X).</code>	$\forall x(W(x) \rightarrow H(x))$	„Jede Frau ist ein Mensch.“

- enthält Beschreibung der Welt
- ist das Prolog-„Programm“
- besteht aus prädikatenlogischen Formeln (**Klauseln**)

Beispiel 5.3

Prolog	Prädikatenlogik	natürliche Sprache
<code>party.</code>	P	„Es gibt eine Party.“
<code>woman(mia).</code>	$W(m)$	„Mia ist eine Frau.“
<code>human(X) :- woman(X).</code>	$\forall x(W(x) \rightarrow H(x))$	„Jede Frau ist ein Mensch.“
<code>human(X) :- human(father(X)), human(mother(X)).</code>	$\forall x(H(f(x)) \wedge H(m(x)) \rightarrow H(x))$	„Sind Vater und Mutter eines Individuums Menschen, ist es auch ein Mensch.“

- beantwortet [Anfragen](#)

- beantwortet [Anfragen](#)
- nutzt Wissensbank

- beantwortet **Anfragen**
- nutzt Wissensbank
- zieht **Schlussfolgerungen**

- beantwortet **Anfragen**
- nutzt Wissensbank
- zieht **Schlussfolgerungen**

- beantwortet **Anfragen**
- nutzt Wissensbank
- zieht **Schlussfolgerungen**

Anfragen

- sind Klauseln

- beantwortet **Anfragen**
- nutzt Wissensbank
- zieht **Schlussfolgerungen**

Anfragen

- sind Klauseln
- sind gültig, wenn ihre Negation unerfüllbar ist

- beantwortet **Anfragen**
- nutzt Wissensbank
- zieht **Schlussfolgerungen**

Anfragen

- sind Klauseln
- sind gültig, wenn ihre Negation unerfüllbar ist
- Interpreter testet Erfüllbarkeit der Negation mit Resolution

- beantwortet **Anfragen**
- nutzt Wissensbank
- zieht **Schlussfolgerungen**

Anfragen

- sind Klauseln
- sind gültig, wenn ihre Negation unerfüllbar ist
- Interpreter testet Erfüllbarkeit der Negation mit Resolution

- beantwortet **Anfragen**
- nutzt Wissensbank
- zieht **Schlussfolgerungen**

Anfragen

- sind Klauseln
- sind gültig, wenn ihre Negation unerfüllbar ist
- Interpreter testet Erfüllbarkeit der Negation mit Resolution

Beispiel 5.4

Prolog	Prädikatenlogik	natürliche Sprache
woman(mia).	$W(m)$	„Ist Mia eine Frau?“

- beantwortet **Anfragen**
- nutzt Wissensbank
- zieht **Schlussfolgerungen**

Anfragen

- sind Klauseln
- sind gültig, wenn ihre Negation unerfüllbar ist
- Interpreter testet Erfüllbarkeit der Negation mit Resolution

Beispiel 5.4

Prolog	Prädikatenlogik	natürliche Sprache
woman(mia).	$W(m)$	„Ist Mia eine Frau?“
human(mia).	$H(m)$	„Ist Mia ein Mensch?“

- beantwortet Anfragen
- nutzt Wissensbank
- zieht Schlussfolgerungen

Anfragen

- sind Klauseln
- sind gültig, wenn ihre Negation unerfüllbar ist
- Interpreter testet Erfüllbarkeit der Negation mit Resolution

Beispiel 5.4

Prolog	Prädikatenlogik	natürliche Sprache
woman(mia).	$W(m)$	„Ist Mia eine Frau?“
human(mia).	$H(m)$	„Ist Mia ein Mensch?“
human(X).	$\exists x H(x)$	„Wer ist ein Mensch?“

- beantwortet **Anfragen**
- nutzt Wissensbank
- zieht **Schlussfolgerungen**

Anfragen

- sind Klauseln
- sind gültig, wenn ihre Negation unerfüllbar ist
- Interpreter testet Erfüllbarkeit der Negation mit Resolution

Beispiel 5.4

Prolog	Prädikatenlogik	natürliche Sprache
woman(mia).	$W(m)$	„Ist Mia eine Frau?“
human(mia).	$H(m)$	„Ist Mia ein Mensch?“
human(X).	$\exists x H(x)$	„Wer ist ein Mensch?“
human(X), human(father(X)).	$\exists x (H(x) \wedge H(f(x)))$	„Wer ist ein Mensch und hat einen menschlichen Vater?“

1. Einführung

2. Mengen

3. Aussagenlogik

4. Prädikatenlogik

5. Prolog

5.1 Terme

5.2 Fakten, Regeln und Anfragen

5.3 Unifikation, Termgleichheit und Arithmetik

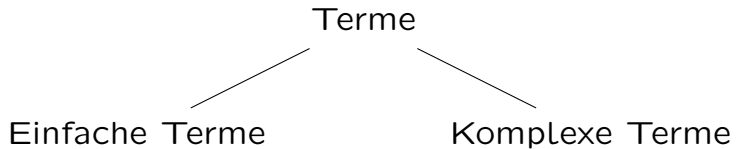
5.4 Horn-Resolution und Suchbäume

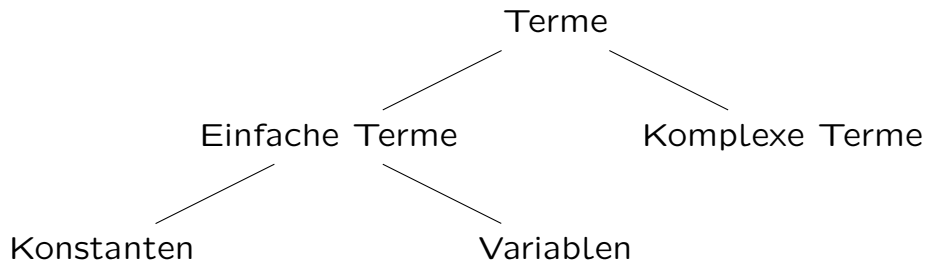
5.5 Rekursion

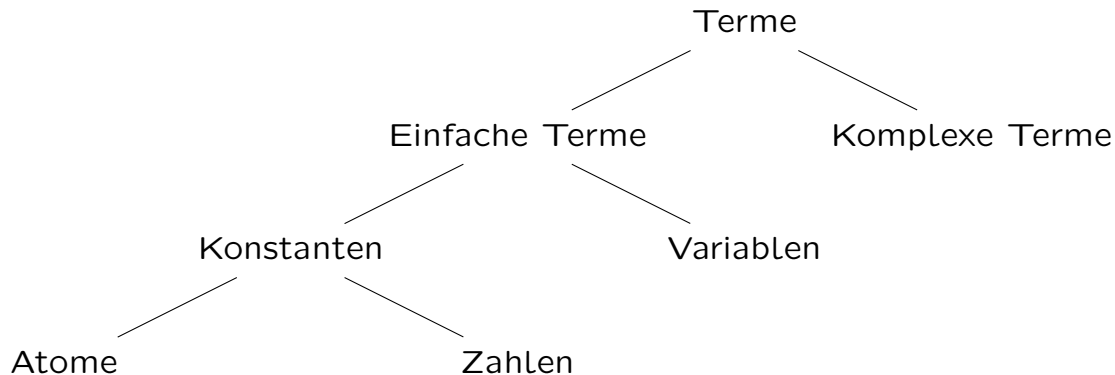
5.6 Listen

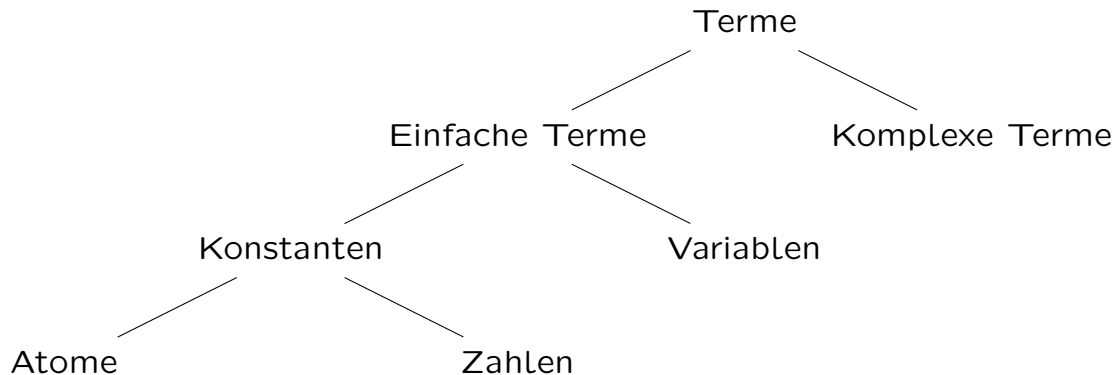
5.7 Der Cut

Terme









Vorsicht

Die Begriffe **Term** und **Atom** haben eine andere Bedeutung als in der Logik!

- Atome** ■ Strings bestehend aus Buchstaben, Ziffern und Unterstrich (_)
beginnend mit **Kleinbuchstaben**,
Z. B. butch, honey_bunny, bigKahunaBurger3

- Atome
- Strings bestehend aus Buchstaben, Ziffern und Unterstrich (`_`) beginnend mit **Kleinbuchstaben**,
Z. B. `butch`, `honey_bunny`, `bigKahunaBurger3`
 - Beliebige Strings in einfachen Anführungsstrichen
Z. B. `'Vincent'`, `'Big Kahuna Burger'`, `'mia@wallace.us'`

- Atome**
- Strings bestehend aus Buchstaben, Ziffern und Unterstrich (`_`) beginnend mit **Kleinbuchstaben**,
Z. B. `butch`, `honey_bunny`, `bigKahunaBurger3`
 - Beliebige Strings in einfachen Anführungsstrichen
Z. B. `'Vincent'`, `'Big Kahuna Burger'`, `'mia@wallace.us'`

- Variablen**
- Strings bestehend aus Buchstaben, Ziffern und Unterstrich beginnend mit **Großbuchstaben oder Unterstrich**
Z. B. `X`, `Y`, `Variable`, `_var1`

- Atome**
 - Strings bestehend aus Buchstaben, Ziffern und Unterstrich (`_`) beginnend mit **Kleinbuchstaben**,
Z. B. `butch`, `honey_bunny`, `bigKahunaBurger3`
 - Beliebige Strings in einfachen Anführungsstrichen
Z. B. `'Vincent'`, `'Big Kahuna Burger'`, `'mia@wallace.us'`
- Variablen**
 - Strings bestehend aus Buchstaben, Ziffern und Unterstrich beginnend mit **Großbuchstaben oder Unterstrich**
Z. B. `X`, `Y`, `Variable`, `_var1`
- Zahlen**
 - Ganzzahlen: Strings bestehend aus Ziffern, ggf. beginnend mit Minus
Z. B. `1`, `65536`, `-28`

Atome

- Strings bestehend aus Buchstaben, Ziffern und Unterstrich (`_`) beginnend mit **Kleinbuchstaben**,
Z. B. `butch`, `honey_bunny`, `bigKahunaBurger3`
- Beliebige Strings in einfachen Anführungsstrichen
Z. B. `'Vincent'`, `'Big Kahuna Burger'`, `'mia@wallace.us'`

Variablen

- Strings bestehend aus Buchstaben, Ziffern und Unterstrich beginnend mit **Großbuchstaben oder Unterstrich**
Z. B. `X`, `Y`, `Variable`, `_var1`

Zahlen

- Ganzzahlen: Strings bestehend aus Ziffern, ggf. beginnend mit Minus
Z. B. `1`, `65536`, `-28`
- Gleitkommazahlen: Zusätzlich Dezimalpunkt und Exponentialschreibweise
Z. B. `2.5`, `-0.000372`, `3.1415e-3`

Übung 5.5

Sind die folgenden Ausdrücke Atome, Variablen oder keins von beidem?

- 1 `vINCENT`
- 2 `Footmassage`
- 3 `variable23`
- 4 `Variable2000`
- 5 `big_kahuna_burger`
- 6 `'big kahuna burger'`
- 7 `big kahuna burger`
- 8 `'Jules'`
- 9 `_Jules`
- 10 `'_Jules'`

Funktor muss Atom sein

Funktor muss Atom sein

Argumente können beliebige Terme sein (auch komplexe)

Funktor muss Atom sein

Argumente können beliebige Terme sein (auch komplexe)

- getrennt durch Komma

Funktor muss Atom sein

Argumente können beliebige Terme sein (auch komplexe)

- getrennt durch Komma
- eingeschlossen in Klammern

Funktor muss Atom sein

Argumente können beliebige Terme sein (auch komplexe)

- getrennt durch Komma
- eingeschlossen in Klammern
- Anzahl der Argumente: **Stelligkeit**

Funktor muss Atom sein

Argumente können beliebige Terme sein (auch komplexe)

- getrennt durch Komma
- eingeschlossen in Klammern
- Anzahl der Argumente: **Stelligkeit**
- **kein** Leerzeichen zwischen Funktor und Klammer

Funktor muss Atom sein

Argumente können beliebige Terme sein (auch komplexe)

- getrennt durch Komma
- eingeschlossen in Klammern
- Anzahl der Argumente: **Stelligkeit**
- **kein** Leerzeichen zwischen Funktor und Klammer

Funktor muss Atom sein

Argumente können beliebige Terme sein (auch komplexe)

- getrennt durch Komma
- eingeschlossen in Klammern
- Anzahl der Argumente: **Stelligkeit**
- **kein** Leerzeichen zwischen Funktor und Klammer

Beispiel 5.6

```
loves(marsellus, mia)
```

```
woman(mia)
```

```
eats(jules, big_kahuna_burger)
```

```
knows(koons, father(butch))
```

```
loves(vincent, wife(marsellus))
```

```
less(zero, s(s(s(zero))))
```

Prädikatenlogik ■ Funktionssymbol $f^{(i)}$ hat feste Stelligkeit i

Prädikatenlogik

- Funktionssymbol $f^{(i)}$ hat feste Stelligkeit i
- $f^{(1)}(x, y)$ ist syntaktisch falsch

Prädikatenlogik

- Funktionssymbol $f^{(i)}$ hat feste Stelligkeit i
- $f^{(1)}(x, y)$ ist syntaktisch falsch

Prolog

- Atom f kann mit unterschiedlichen Stelligkeiten verwendet werden

Prädikatenlogik

- Funktionssymbol $f^{(i)}$ hat feste Stelligkeit i
- $f^{(1)}(x, y)$ ist syntaktisch falsch

Prolog

- Atom f kann mit unterschiedlichen Stelligkeiten verwendet werden
- $f(x)$ und $f(x,y)$ können parallel verwendet werden

Prädikatenlogik

- Funktionssymbol $f^{(i)}$ hat feste Stelligkeit i
- $f^{(1)}(x, y)$ ist syntaktisch falsch

Prolog

- Atom f kann mit unterschiedlichen Stelligkeiten verwendet werden
- $f(x)$ und $f(x, y)$ können parallel verwendet werden
- Notation: $f/1$ bzw. $f/2$

Prädikatenlogik

- Funktionssymbol $f^{(i)}$ hat feste Stelligkeit i
- $f^{(1)}(x, y)$ ist syntaktisch falsch

Prolog

- Atom f kann mit unterschiedlichen Stelligkeiten verwendet werden
- $f(x)$ und $f(x, y)$ können parallel verwendet werden
- Notation: $f/1$ bzw. $f/2$
- Zwischen $f/1$ und $f/2$ besteht kein Zusammenhang

Prädikatenlogik

- Funktionssymbol $f^{(i)}$ hat feste Stelligkeit i
- $f^{(1)}(x, y)$ ist syntaktisch falsch

Prolog

- Atom `f` kann mit unterschiedlichen Stelligkeiten verwendet werden
- `f(x)` und `f(x,y)` können parallel verwendet werden
- Notation: `f/1` bzw. `f/2`
- Zwischen `f/1` und `f/2` besteht kein Zusammenhang
 - aus `mother(john,anna)` folgt nicht `mother(anna)`

Definition 5.7 (Prädikat)

Ein **Prädikat** ist ein Atom, das als äußerster Funktor in einem komplexen Term in der Wissensbank vorkommt.

Definition 5.7 (Prädikat)

Ein **Prädikat** ist ein Atom, das als äußerster Funktor in einem komplexen Term in der Wissensbank vorkommt.

Prädikatenlogik ■ Funktionssymbol $\rightsquigarrow$ Term $\rightsquigarrow$ Domänenelement

Definition 5.7 (Prädikat)

Ein **Prädikat** ist ein Atom, das als äußerster Funktor in einem komplexen Term in der Wissensbank vorkommt.

- Prädikatenlogik**
- Funktionssymbol $\rightsquigarrow$ Term $\rightsquigarrow$ Domänenelement
 - Prädikat $\rightsquigarrow$ Atom $\rightsquigarrow$ Wahrheitswert

Definition 5.7 (Prädikat)

Ein **Prädikat** ist ein Atom, das als äußerster Funktor in einem komplexen Term in der Wissensbank vorkommt.

Prädikatenlogik

- Funktionssymbol $\rightsquigarrow$ Term $\rightsquigarrow$ Domänenelement

- Prädikat $\rightsquigarrow$ Atom $\rightsquigarrow$ Wahrheitswert

Prolog

- keine syntaktische Unterscheidung zwischen Funktionssymbolen und Prädikaten

Definition 5.7 (Prädikat)

Ein **Prädikat** ist ein Atom, das als äußerster Funktor in einem komplexen Term in der Wissensbank vorkommt.

Prädikatenlogik

- Funktionssymbol $\rightsquigarrow$ Term $\rightsquigarrow$ Domänenelement
- Prädikat $\rightsquigarrow$ Atom $\rightsquigarrow$ Wahrheitswert

Prolog

- keine syntaktische Unterscheidung zwischen Funktionssymbolen und Prädikaten
- Faustregel:
 - Äußerster Funktor: Prädikat

Definition 5.7 (Prädikat)

Ein **Prädikat** ist ein Atom, das als äußerster Funktor in einem komplexen Term in der Wissensbank vorkommt.

Prädikatenlogik

- Funktionssymbol $\rightsquigarrow$ Term $\rightsquigarrow$ Domänenelement

- Prädikat $\rightsquigarrow$ Atom $\rightsquigarrow$ Wahrheitswert

Prolog

- keine syntaktische Unterscheidung zwischen Funktionssymbolen und Prädikaten

- Faustregel:

 - Äußerster Funktor: Prädikat

 - Innere Funktoren: Funktionssymbole

Definition 5.7 (Prädikat)

Ein **Prädikat** ist ein Atom, das als äußerster Funktor in einem komplexen Term in der Wissensbank vorkommt.

Prädikatenlogik

- Funktionssymbol $\rightsquigarrow$ Term $\rightsquigarrow$ Domänenelement
- Prädikat $\rightsquigarrow$ Atom $\rightsquigarrow$ Wahrheitswert
- Prolog**
 - keine syntaktische Unterscheidung zwischen Funktionssymbolen und Prädikaten
 - Faustregel:
 - Äußerster Funktor: Prädikat
 - Innere Funktoren: Funktionssymbole

Beispiel 5.8

- `knows(koons, father(butch)).`
 - `knows` ist zweistelliges Prädikat
 - `father` ist einstelliges Funktionssymbol ("Der Vater von Butch")
- `father(butch).`
 - `father` ist einstelliges Prädikat ("Butch ist ein Vater.")

Übung 5.9

Sind die folgenden Ausdrücke Atome, Variablen, komplexe Terme oder nichts von allem?

- 1 `loves(Vincent,mia)`
- 2 `'loves(Vincent,mia)'`
- 3 `Butch(boxer)`
- 4 `boxer(Butch)`
- 5 `and(big(burger),kahuna(burger))`
- 6 `and(big(X),kahuna(X))`
- 7 `_and(big(X),kahuna(X))`
- 8 `(Butch kills Vincent)`
- 9 `kills(Butch Vincent)`
- 10 `kills(Butch, Vincent`
- 11 `kills(Butch, Vincent)`
- 12 `kills(Butch(X), Vincent(Y))`

Einfacher Term

Atom beginnt mit Kleinbuchstabe
entspricht Funktionssymbol oder Prädikat in Prädikatenlogik

Einfacher Term

Atom beginnt mit Kleinbuchstabe

entspricht Funktionssymbol oder Prädikat in Prädikatenlogik

Variable beginnt mit Großbuchstabe oder Unterstrich

Einfacher Term

- Atom** beginnt mit Kleinbuchstabe
entspricht Funktionssymbol oder Prädikat in Prädikatenlogik
- Variable** beginnt mit Großbuchstabe oder Unterstrich
- Zahl** Ganzzahl oder Gleitkommazahl

- Einfacher Term
 - Atom beginnt mit Kleinbuchstabe
entspricht Funktionssymbol oder Prädikat in Prädikatenlogik
 - Variable beginnt mit Großbuchstabe oder Unterstrich
 - Zahl Ganzzahl oder Gleitkommazahl
- Komplexer Term besteht aus
 - Funktor Atom

- Einfacher Term
 - Atom beginnt mit Kleinbuchstabe
entspricht Funktionssymbol oder Prädikat in Prädikatenlogik
 - Variable beginnt mit Großbuchstabe oder Unterstrich
 - Zahl Ganzzahl oder Gleitkommazahl
- Komplexer Term besteht aus
 - Funktor Atom
 - Argumente beliebige Terme

- Einfacher Term
 - Atom beginnt mit Kleinbuchstabe
entspricht Funktionssymbol oder Prädikat in Prädikatenlogik
 - Variable beginnt mit Großbuchstabe oder Unterstrich
 - Zahl Ganzzahl oder Gleitkommazahl
- Komplexer Term besteht aus
 - Funktor Atom
 - Argumente beliebige Terme
- Prädikat Äußerster Funktor eines komplexen Terms

- 1. Einführung
- 2. Mengen
- 3. Aussagenlogik
- 4. Prädikatenlogik
- 5. Prolog**
 - 5.1 Terme
 - 5.2 Fakten, Regeln und Anfragen**
 - 5.3 Unifikation, Termgleichheit und Arithmetik
 - 5.4 Horn-Resolution und Suchbäume
 - 5.5 Rekursion
 - 5.6 Listen
 - 5.7 Der Cut

Definition 5.10 (Goal)

Jedes Atom und jeder komplexe Term ist ein **Goal**.

Definition 5.10 (Goal)

Jedes Atom und jeder komplexe Term ist ein **Goal**.

- entspricht prädikatenlogischem Atombegriff

Definition 5.10 (Goal)

Jedes Atom und jeder komplexe Term ist ein **Goal**.

- entspricht prädikatenlogischem Atombegriff
- in Wissensbank: Aussage über Domäne

Definition 5.10 (Goal)

Jedes Atom und jeder komplexe Term ist ein **Goal**.

- entspricht prädikatenlogischem Atombegriff
- in Wissensbank: Aussage über Domäne
- als Anfrage: Frage zur Domäne

Definition 5.10 (Goal)

Jedes Atom und jeder komplexe Term ist ein **Goal**.

- entspricht prädikatenlogischem Atombegriff
- in Wissensbank: Aussage über Domäne
- als Anfrage: Frage zur Domäne

Definition 5.10 (Goal)

Jedes Atom und jeder komplexe Term ist ein **Goal**.

- entspricht prädikatenlogischem Atombegriff
- in Wissensbank: Aussage über Domäne
- als Anfrage: Frage zur Domäne

Beispiel 5.11 (Goals)

Term	Aussage	Anfrage
rain	Es regnet.	Regnet es?

Definition 5.10 (Goal)

Jedes Atom und jeder komplexe Term ist ein **Goal**.

- entspricht prädikatenlogischem Atombegriff
- in Wissensbank: Aussage über Domäne
- als Anfrage: Frage zur Domäne

Beispiel 5.11 (Goals)

Term	Aussage	Anfrage
<code>rain</code>	Es regnet.	Regnet es?
<code>woman(mia)</code>	Mia ist eine Frau.	Ist Mia eine Frau?

Definition 5.10 (Goal)

Jedes Atom und jeder komplexe Term ist ein **Goal**.

- entspricht prädikatenlogischem Atombegriff
- in Wissensbank: Aussage über Domäne
- als Anfrage: Frage zur Domäne

Beispiel 5.11 (Goals)

Term	Aussage	Anfrage
<code>rain</code>	Es regnet.	Regnet es?
<code>woman(mia)</code>	Mia ist eine Frau.	Ist Mia eine Frau?
<code>mother(john,anna)</code>	Johns Mutter ist Anna.	Ist Anna Johns Mutter?

Definition 5.10 (Goal)

Jedes Atom und jeder komplexe Term ist ein **Goal**.

- entspricht prädikatenlogischem Atombegriff
- in Wissensbank: Aussage über Domäne
- als Anfrage: Frage zur Domäne

Beispiel 5.11 (Goals)

Term	Aussage	Anfrage
<code>rain</code>	Es regnet.	Regnet es?
<code>woman(mia)</code>	Mia ist eine Frau.	Ist Mia eine Frau?
<code>mother(john,anna)</code>	Johns Mutter ist Anna.	Ist Anna Johns Mutter?
<code>age(john,12)</code>	John hat das Alter 12.	Hat John das Alter 12?

Definition 5.10 (Goal)

Jedes Atom und jeder komplexe Term ist ein **Goal**.

- entspricht prädikatenlogischem Atombegriff
- in Wissensbank: Aussage über Domäne
- als Anfrage: Frage zur Domäne

Beispiel 5.11 (Goals)

Term	Aussage	Anfrage
<code>rain</code>	Es regnet.	Regnet es?
<code>woman(mia)</code>	Mia ist eine Frau.	Ist Mia eine Frau?
<code>mother(john,anna)</code>	Johns Mutter ist Anna.	Ist Anna Johns Mutter?
<code>age(john,12)</code>	John hat das Alter 12.	Hat John das Alter 12?

Keine Goals: Zahlen, Variablen

Fakten

Syntax $\langle \text{Goal} \rangle .$

Fakten

Syntax $\langle \text{Goal} \rangle$.

Semantik Der Fakt gilt in der Domäne.

Fakten

Syntax <Goal>.

Semantik Der Fakt gilt in der Domäne.

Beispiel 5.12

Wissensbank

```
woman(mia).  
woman(jody).  
woman(yolanda).  
dances(jody).  
party.
```

Fakten

Syntax `<Goal> .`

Semantik Der Fakt gilt in der Domäne.

Anfragen

Syntax `<Goal> .`

Beispiel 5.12

Wissensbank

```
woman(mia).  
woman(jody).  
woman(yolanda).  
dances(jody).  
party.
```

Fakten

Syntax <Goal> .

Semantik Der Fakt gilt in der Domäne.

Anfragen

Syntax <Goal> .

Semantik Gilt die Aussage in der Domäne?

Beispiel 5.12

Wissensbank

```
woman(mia).  
woman(jody).  
woman(yolanda).  
dances(jody).  
party.
```


Fakten

Syntax <Goal>.

Semantik Der Fakt gilt in der Domäne.

Beispiel 5.12

Wissensbank

```
woman(mia).  
woman(jody).  
woman(yolanda).  
dances(jody).  
party.
```

Anfragen

Syntax <Goal>.

Semantik Gilt die Aussage in der Domäne?

Beispiel 5.13

Interpreter

```
?- woman(mia).
```

Fakten

Syntax <Goal>.

Semantik Der Fakt gilt in der Domäne.

Beispiel 5.12

Wissensbank

```
woman(mia).  
woman(jody).  
woman(yolanda).  
dances(jody).  
party.
```

Anfragen

Syntax <Goal>.

Semantik Gilt die Aussage in der Domäne?

Beispiel 5.13

Interpreter

```
?- woman(mia).  
true.
```

Fakten

Syntax <Goal>.

Semantik Der Fakt gilt in der Domäne.

Beispiel 5.12

Wissensbank

```
woman(mia).  
woman(jody).  
woman(yolanda).  
dances(jody).  
party.
```

Anfragen

Syntax <Goal>.

Semantik Gilt die Aussage in der Domäne?

Beispiel 5.13

Interpreter

```
?- woman(mia).  
true.  
?- dances(jody).
```

Fakten

Syntax <Goal>.

Semantik Der Fakt gilt in der Domäne.

Beispiel 5.12

Wissensbank

```
woman(mia).  
woman(jody).  
woman(yolanda).  
dances(jody).  
party.
```

Anfragen

Syntax <Goal>.

Semantik Gilt die Aussage in der Domäne?

Beispiel 5.13

Interpreter

```
?- woman(mia).  
true.  
?- dances(jody).  
true.
```

Fakten

Syntax <Goal>.

Semantik Der Fakt gilt in der Domäne.

Beispiel 5.12

Wissensbank

```
woman(mia).  
woman(jody).  
woman(yolanda).  
dances(jody).  
party.
```

Anfragen

Syntax <Goal>.

Semantik Gilt die Aussage in der Domäne?

Beispiel 5.13

Interpreter

```
?- woman(mia).  
true.  
?- dances(jody).  
true.  
?- dances(mia).
```

Fakten

Syntax <Goal>.

Semantik Der Fakt gilt in der Domäne.

Beispiel 5.12

Wissensbank

```
woman(mia).  
woman(jody).  
woman(yolanda).  
dances(jody).  
party.
```

Anfragen

Syntax <Goal>.

Semantik Gilt die Aussage in der Domäne?

Beispiel 5.13

Interpreter

```
?- woman(mia).  
true.  
?- dances(jody).  
true.  
?- dances(mia).  
false.
```

Fakten

Syntax <Goal>.

Semantik Der Fakt gilt in der Domäne.

Beispiel 5.12

Wissensbank

```
woman(mia).  
woman(jody).  
woman(yolanda).  
dances(jody).  
party.
```

Anfragen

Syntax <Goal>.

Semantik Gilt die Aussage in der Domäne?

Beispiel 5.13

Interpreter

```
?- woman(mia).  
true.  
?- dances(jody).  
true.  
?- dances(mia).  
false.  
?- party.
```

Fakten

Syntax <Goal>.

Semantik Der Fakt gilt in der Domäne.

Beispiel 5.12

Wissensbank

```
woman(mia).  
woman(jody).  
woman(yolanda).  
dances(jody).  
party.
```

Anfragen

Syntax <Goal>.

Semantik Gilt die Aussage in der Domäne?

Beispiel 5.13

Interpreter

```
?- woman(mia).  
true.  
?- dances(jody).  
true.  
?- dances(mia).  
false.  
?- party.  
true.
```


Beispiel 5.14

Wissensbank

dances(jody).

Interpreter

?- dances(jody).

true.

?- dances(mia).

false.

- Antwort **true** bedeutet: Anfrage ist gültig

Beispiel 5.14

Wissensbank

dances(jody).

Interpreter

?- dances(jody).

true.

?- dances(mia).

false.

- Antwort **true** bedeutet: Anfrage ist **gültig**
 - In jedem Modell der KB gilt auch die Anfrage

Beispiel 5.14

Wissensbank

dances(jody).

Interpreter

?- dances(jody).

true.

?- dances(mia).

false.

- Antwort **true** bedeutet: Anfrage ist **gültig**
 - In jedem Modell der KB gilt auch die Anfrage
- Antwort **false** bedeutet: Anfrage ist **falsifizierbar**

Beispiel 5.14

Wissensbank

dances(jody).

Interpreter

?- dances(jody).

true.

?- dances(mia).

false.

- Antwort **true** bedeutet: Anfrage ist **gültig**
 - In jedem Modell der KB gilt auch die Anfrage
- Antwort **false** bedeutet: Anfrage ist **falsifizierbar**
 - Es gibt ein Modell der KB, in dem die Anfrage nicht gilt.

Beispiel 5.14

Wissensbank

dances(jody).

Interpreter

?- dances(jody).

true.

?- dances(mia).

false.

- Antwort **true** bedeutet: Anfrage ist **gültig**
 - In jedem Modell der KB gilt auch die Anfrage
- Antwort **false** bedeutet: Anfrage ist **falsifizierbar**
 - Es gibt ein Modell der KB, in dem die Anfrage nicht gilt.
 - Antwort **false** bedeutet **nicht**, dass aus der KB folgt, dass **Mia nicht tanzt**.

Beispiel 5.14

Wissensbank

dances(jody).

Interpreter

?- dances(jody).

true.

?- dances(mia).

false.

- Antwort **true** bedeutet: Anfrage ist **gültig**
 - In jedem Modell der KB gilt auch die Anfrage
- Antwort **false** bedeutet: Anfrage ist **falsifizierbar**
 - Es gibt ein Modell der KB, in dem die Anfrage nicht gilt.
 - Antwort **false** bedeutet **nicht**, dass aus der KB folgt, dass **Mia nicht tanzt**.
 - Sondern: Aus der KB **folgt nicht**, dass Mia tanzt.

Beispiel 5.14

Wissensbank

dances(jody).

Interpreter

?- dances(jody).

true.

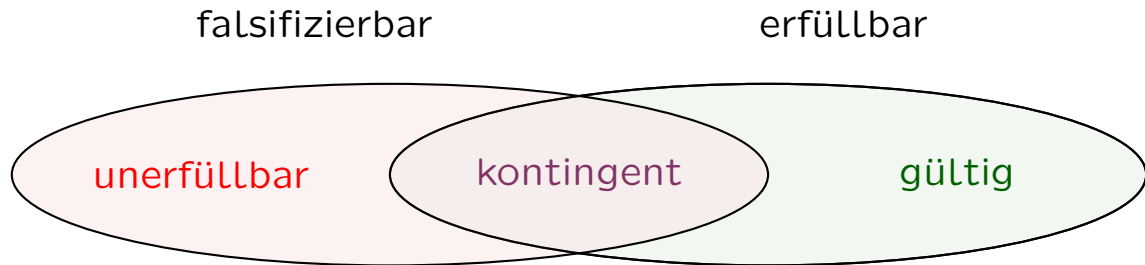
?- dances(mia).

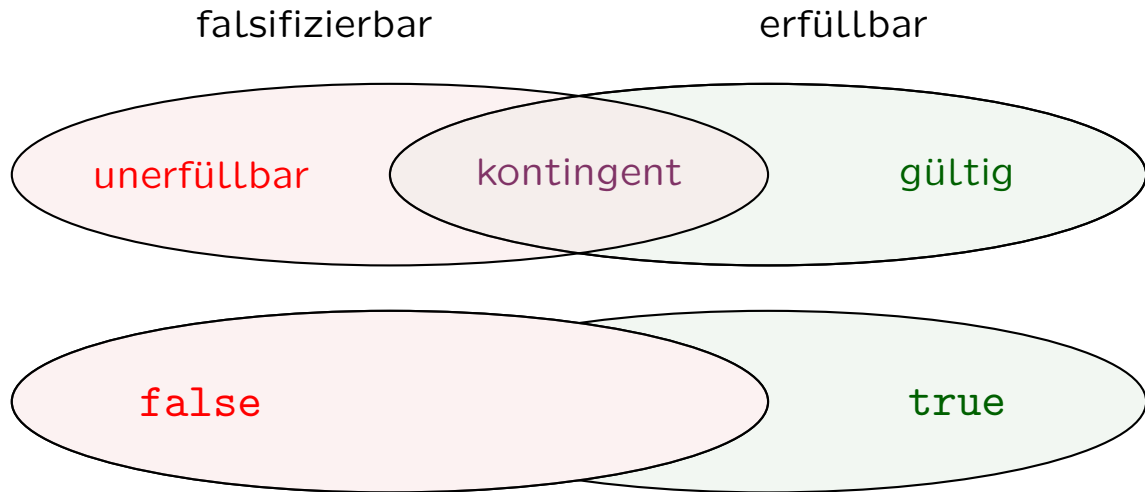
false.

- Antwort **true** bedeutet: Anfrage ist **gültig**
 - In jedem Modell der KB gilt auch die Anfrage
- Antwort **false** bedeutet: Anfrage ist **falsifizierbar**
 - Es gibt ein Modell der KB, in dem die Anfrage nicht gilt.
 - Antwort **false** bedeutet **nicht**, dass aus der KB folgt, dass **Mia nicht tanzt**.
 - Sondern: Aus der KB **folgt nicht**, dass Mia tanzt.

Definition 5.15 (Closed World Assumption)

Alle Aussagen, die nicht mit Sicherheit aus der KB folgen, werden als **false** betrachtet.





Syntax $\langle \text{Kopf} \rangle \text{ :- } \langle \text{Rumpf} \rangle \text{ .}$
Kopf und Rumpf: Goals

Syntax $\langle \text{Kopf} \rangle \text{ :- } \langle \text{Rumpf} \rangle .$

Kopf und Rumpf: Goals

Semantik Kopf *wenn* Rumpf

Der Rumpf impliziert den Kopf.

Syntax <Kopf> :- <Rumpf> .

Kopf und Rumpf: Goals

Semantik Kopf **wenn** Rumpf

Der Rumpf impliziert den Kopf.

Beispiel 5.16

Wissensbank

```
works(jules).  
hungry(vincent).  
hungry(jules) :- works(jules).  
eats(vincent) :- hungry(vincent).  
eats(jules) :- hungry(jules).
```

Syntax <Kopf> :- <Rumpf> .

Kopf und Rumpf: Goals

Semantik Kopf **wenn** Rumpf

Der Rumpf impliziert den Kopf.

Beispiel 5.16

Wissensbank

```
works(jules).  
hungry(vincent).  
hungry(jules) :- works(jules).  
eats(vincent) :- hungry(vincent).  
eats(jules) :- hungry(jules).
```

- 2 Fakten
- 3 Regeln

Syntax <Kopf> :- <Rumpf> .

Kopf und Rumpf: Goals

Semantik Kopf **wenn** Rumpf

Der Rumpf impliziert den Kopf.

Beispiel 5.16

Wissensbank

```
works(jules).  
hungry(vincent).  
hungry(jules) :- works(jules).  
eats(vincent) :- hungry(vincent).  
eats(jules) :- hungry(jules).
```

- 2 Fakten
- 3 Regeln

Interpreter

```
?- eats(vincent).
```

Syntax <Kopf> :- <Rumpf> .

Kopf und Rumpf: Goals

Semantik Kopf **wenn** Rumpf

Der Rumpf impliziert den Kopf.

Beispiel 5.16

Wissensbank

```
works(jules).  
hungry(vincent).  
hungry(jules) :- works(jules).  
eats(vincent) :- hungry(vincent).  
eats(jules) :- hungry(jules).
```

- 2 Fakten
- 3 Regeln

Interpreter

```
?- eats(vincent).  
true.
```

Syntax <Kopf> :- <Rumpf> .

Kopf und Rumpf: Goals

Semantik Kopf **wenn** Rumpf

Der Rumpf impliziert den Kopf.

Beispiel 5.16

Wissensbank

```
works(jules).  
hungry(vincent).  
hungry(jules) :- works(jules).  
eats(vincent) :- hungry(vincent).  
eats(jules) :- hungry(jules).
```

- 2 Fakten
- 3 Regeln

Interpreter

```
?- eats(vincent).  
true.  
?- eats(jules).
```


Syntax <Kopf> :- <Rumpf> .

Kopf und Rumpf: Goals

Semantik Kopf **wenn** Rumpf

Der Rumpf impliziert den Kopf.

Beispiel 5.16

Wissensbank

```
works(jules).  
hungry(vincent).  
hungry(jules) :- works(jules).  
eats(vincent) :- hungry(vincent).  
eats(jules) :- hungry(jules).
```

- 2 Fakten
- 3 Regeln

Interpreter

```
?- eats(vincent).  
true.  
?- eats(jules).  
true.
```

Syntax <Kopf> :- <Rumpf> .

Kopf und Rumpf: Goals

Semantik Kopf **wenn** Rumpf

Der Rumpf impliziert den Kopf.

Beispiel 5.16

Wissensbank

```
works(jules).  
hungry(vincent).  
hungry(jules) :- works(jules).  
eats(vincent) :- hungry(vincent).  
eats(jules) :- hungry(jules).
```

- 2 Fakten
- 3 Regeln

Interpreter

```
?- eats(vincent).  
true.  
?- eats(jules).  
true.
```

Fakt kann als Regel mit leerem Rumpf aufgefasst werden: `works(jules) :- true.`

Syntax $\langle \text{Kopf} \rangle :- \langle \text{Rumpf} \rangle$

Kopf: Goal

Rumpf: $\langle \text{Goal1} \rangle , \langle \text{Goal1} \rangle , \dots$

Syntax $\langle \text{Kopf} \rangle :- \langle \text{Rumpf} \rangle$

Kopf: Goal

Rumpf: $\langle \text{Goal1} \rangle , \langle \text{Goal1} \rangle , \dots$

Semantik Kopf wenn Goal1 **und** Goal2 **und** ...

Konjunktion der Goals im Rumpf impliziert den Kopf.

Syntax $\langle \text{Kopf} \rangle :- \langle \text{Rumpf} \rangle$

Kopf: Goal

Rumpf: $\langle \text{Goal1} \rangle , \langle \text{Goal1} \rangle , \dots$

Semantik Kopf wenn Goal1 und Goal2 und ...

Konjunktion der Goals im Rumpf impliziert den Kopf.

Beispiel 5.17

Wissensbank

```
happy(butch).  
hungry(vincent).  
eats(butch) :- happy(butch), hungry(butch).  
eats(vincent) :- happy(vincent).  
eats(vincent) :- hungry(vincent).
```

Syntax $\langle \text{Kopf} \rangle :- \langle \text{Rumpf} \rangle$

Kopf: Goal

Rumpf: $\langle \text{Goal1} \rangle , \langle \text{Goal1} \rangle , \dots$

Semantik Kopf wenn Goal1 **und** Goal2 **und** ...

Konjunktion der Goals im Rumpf impliziert den Kopf.

Beispiel 5.17

Wissensbank

```
happy(butch).  
hungry(vincent).  
eats(butch) :- happy(butch), hungry(butch).  
eats(vincent) :- happy(vincent).  
eats(vincent) :- hungry(vincent).
```

Interpreter

```
?- eats(butch).
```

Syntax $\langle \text{Kopf} \rangle :- \langle \text{Rumpf} \rangle$

Kopf: Goal

Rumpf: $\langle \text{Goal1} \rangle , \langle \text{Goal1} \rangle , \dots$

Semantik Kopf wenn Goal1 **und** Goal2 **und** ...

Konjunktion der Goals im Rumpf impliziert den Kopf.

Beispiel 5.17

Wissensbank

```
happy(butch).  
hungry(vincent).  
eats(butch) :- happy(butch), hungry(butch).  
eats(vincent) :- happy(vincent).  
eats(vincent) :- hungry(vincent).
```

Interpreter

```
?- eats(butch).  
false.
```

Syntax $\langle \text{Kopf} \rangle :- \langle \text{Rumpf} \rangle$

Kopf: Goal

Rumpf: $\langle \text{Goal1} \rangle , \langle \text{Goal1} \rangle , \dots$

Semantik Kopf wenn Goal1 **und** Goal2 **und** ...

Konjunktion der Goals im Rumpf impliziert den Kopf.

Beispiel 5.17

Wissensbank

```
happy(butch).  
hungry(vincent).  
eats(butch) :- happy(butch), hungry(butch).  
eats(vincent) :- happy(vincent).  
eats(vincent) :- hungry(vincent).
```

Interpreter

```
?- eats(butch).  
false.  
?- eats(vincent).
```


Syntax $\langle \text{Kopf} \rangle :- \langle \text{Rumpf} \rangle$

Kopf: Goal

Rumpf: $\langle \text{Goal1} \rangle , \langle \text{Goal1} \rangle , \dots$

Semantik Kopf wenn Goal1 **und** Goal2 **und** ...

Konjunktion der Goals im Rumpf impliziert den Kopf.

Beispiel 5.17

Wissensbank

```
happy(butch).  
hungry(vincent).  
eats(butch) :- happy(butch), hungry(butch).  
eats(vincent) :- happy(vincent).  
eats(vincent) :- hungry(vincent).
```

Interpreter

```
?- eats(butch).  
false.  
?- eats(vincent).  
true.
```

Syntax $\langle \text{Kopf} \rangle :- \langle \text{Rumpf} \rangle$
Kopf: $\langle \text{Goal} \rangle$
Rumpf: $\langle \text{Goal1} \rangle ; \langle \text{Goal1} \rangle ; \dots$

Syntax $\langle \text{Kopf} \rangle :- \langle \text{Rumpf} \rangle$

Kopf: $\langle \text{Goal} \rangle$

Rumpf: $\langle \text{Goal1} \rangle ; \langle \text{Goal1} \rangle ; \dots$

Semantik Kopf wenn Goal1 oder Goal2 oder ...

Disjunktion der Goals im Rumpf impliziert den Kopf.

Syntax <Kopf> :- <Rumpf>

Kopf: <Goal>

Rumpf: <Goal1> ; <Goal1> ; ...

Semantik Kopf wenn Goal1 oder Goal2 oder ...

Disjunktion der Goals im Rumpf impliziert den Kopf.

Beispiel 5.18

Wissensbank

```
happy(butch).  
hungry(vincent).  
eats(butch) :- happy(butch), hungry(butch).  
eats(vincent) :- happy(vincent); hungry(vincent).
```

Syntax <Kopf> :- <Rumpf>

Kopf: <Goal>

Rumpf: <Goal1> ; <Goal1> ; ...

Semantik Kopf wenn Goal1 oder Goal2 oder ...

Disjunktion der Goals im Rumpf impliziert den Kopf.

Beispiel 5.18

Wissensbank

```
happy(butch).  
hungry(vincent).  
eats(butch) :- happy(butch), hungry(butch).  
eats(vincent) :- happy(vincent); hungry(vincent).
```

Interpreter

```
?- eats(butch).
```

Syntax <Kopf> :- <Rumpf>

Kopf: <Goal>

Rumpf: <Goal1> ; <Goal1> ; ...

Semantik Kopf wenn Goal1 **oder** Goal2 **oder** ...

Disjunktion der Goals im Rumpf impliziert den Kopf.

Beispiel 5.18

Wissensbank

```
happy(butch).  
hungry(vincent).  
eats(butch) :- happy(butch), hungry(butch).  
eats(vincent) :- happy(vincent); hungry(vincent).
```

Interpreter

```
?- eats(butch).  
false.
```

Syntax <Kopf> :- <Rumpf>

Kopf: <Goal>

Rumpf: <Goal1> ; <Goal1> ; ...

Semantik Kopf wenn Goal1 **oder** Goal2 **oder** ...

Disjunktion der Goals im Rumpf impliziert den Kopf.

Beispiel 5.18

Wissensbank

```
happy(butch).  
hungry(vincent).  
eats(butch) :- happy(butch), hungry(butch).  
eats(vincent) :- happy(vincent); hungry(vincent).
```

Interpreter

```
?- eats(butch).  
false.  
?- eats(vincent).
```

Syntax <Kopf> :- <Rumpf>

Kopf: <Goal>

Rumpf: <Goal1> ; <Goal1> ; ...

Semantik Kopf wenn Goal1 **oder** Goal2 **oder** ...

Disjunktion der Goals im Rumpf impliziert den Kopf.

Beispiel 5.18

Wissensbank

```
happy(butch).  
hungry(vincent).  
eats(butch) :- happy(butch), hungry(butch).  
eats(vincent) :- happy(vincent); hungry(vincent).
```

Interpreter

```
?- eats(butch).  
false.  
?- eats(vincent).  
true.
```


Syntax <Kopf> :- <Rumpf>

Kopf: <Goal>

Rumpf: <Goal1> ; <Goal1> ; ...

Semantik Kopf wenn Goal1 **oder** Goal2 **oder** ...

Disjunktion der Goals im Rumpf impliziert den Kopf.

Beispiel 5.18

Wissensbank

```
happy(butch).  
hungry(vincent).  
eats(butch) :- happy(butch), hungry(butch).  
eats(vincent) :- happy(vincent); hungry(vincent).
```

Interpreter

```
?- eats(butch).  
false.  
?- eats(vincent).  
true.
```

Konvention: Statt Semikolon **mehrere Regeln** verwenden $\rightsquigarrow$ bessere Lesbarkeit

- können vom Nutzer als Argumente eines Funktors verwendet werden

- können vom Nutzer als Argumente eines Funktors verwendet werden
- können vom Interpreter an Terme **gebunden** werden

- können vom Nutzer als Argumente eines Funktors verwendet werden
- können vom Interpreter an Terme **gebunden** werden
- **gebundene Variable kann Wert nie mehr ändern**

- können vom Nutzer als Argumente eines Funktors verwendet werden
- können vom Interpreter an Terme **gebunden** werden
- **gebundene Variable kann Wert nie mehr ändern**
- Variablenbindung wird zurückgegeben

- können vom Nutzer als Argumente eines Funktors verwendet werden
- können vom Interpreter an Terme **gebunden** werden
- **gebundene Variable kann Wert nie mehr ändern**
- Variablenbindung wird zurückgegeben
- Suche nach anderen Bindungen: ; eingeben

- können vom Nutzer als Argumente eines Funktors verwendet werden
- können vom Interpreter an Terme **gebunden** werden
- **gebundene Variable kann Wert nie mehr ändern**
- Variablenbindung wird zurückgegeben
- Suche nach anderen Bindungen: ; eingeben

Beispiel 5.19

Wissensbank

```
woman(mia).  
woman(jody).  
woman(yolanda).
```

- können vom Nutzer als Argumente eines Funktors verwendet werden
- können vom Interpreter an Terme **gebunden** werden
- **gebundene Variable kann Wert nie mehr ändern**
- Variablenbindung wird zurückgegeben
- Suche nach anderen Bindungen: **;** eingeben

Beispiel 5.19

Wissensbank

```
woman(mia).  
woman(jody).  
woman(yolanda).
```

Interpreter

```
?- woman(X).
```


- können vom Nutzer als Argumente eines Funktors verwendet werden
- können vom Interpreter an Terme **gebunden** werden
- **gebundene Variable kann Wert nie mehr ändern**
- Variablenbindung wird zurückgegeben
- Suche nach anderen Bindungen: **;** eingeben

Beispiel 5.19

Wissensbank

```
woman(mia).  
woman(jody).  
woman(yolanda).
```

Interpreter

```
?- woman(X).  
X = mia
```

- können vom Nutzer als Argumente eines Funktors verwendet werden
- können vom Interpreter an Terme **gebunden** werden
- **gebundene Variable kann Wert nie mehr ändern**
- Variablenbindung wird zurückgegeben
- Suche nach anderen Bindungen: **;** eingeben

Beispiel 5.19

Wissensbank

```
woman(mia).  
woman(jody).  
woman(yolanda).
```

Interpreter

```
?- woman(X).  
X = mia ;
```

- können vom Nutzer als Argumente eines Funktors verwendet werden
- können vom Interpreter an Terme **gebunden** werden
- **gebundene Variable kann Wert nie mehr ändern**
- Variablenbindung wird zurückgegeben
- Suche nach anderen Bindungen: **;** eingeben

Beispiel 5.19

Wissensbank

```
woman(mia).  
woman(jody).  
woman(yolanda).
```

Interpreter

```
?- woman(X).  
X = mia ;  
X = jody
```

- können vom Nutzer als Argumente eines Funktors verwendet werden
- können vom Interpreter an Terme **gebunden** werden
- **gebundene Variable kann Wert nie mehr ändern**
- Variablenbindung wird zurückgegeben
- Suche nach anderen Bindungen: **;** eingeben

Beispiel 5.19

Wissensbank

```
woman(mia).  
woman(jody).  
woman(yolanda).
```

Interpreter

```
?- woman(X).  
X = mia ;  
X = jody ;
```

- können vom Nutzer als Argumente eines Funktors verwendet werden
- können vom Interpreter an Terme **gebunden** werden
- **gebundene Variable kann Wert nie mehr ändern**
- Variablenbindung wird zurückgegeben
- Suche nach anderen Bindungen: **;** eingeben

Beispiel 5.19

Wissensbank

```
woman(mia).  
woman(jody).  
woman(yolanda).
```

Interpreter

```
?- woman(X).  
X = mia ;  
X = jody ;  
X = yolanda.
```

- Anfragen mit mehreren Goals, getrennt durch Komma

- Anfragen mit mehreren Goals, getrennt durch Komma
- Variablenbindung muss **alle** Goals gleichzeitig wahr machen

- Anfragen mit mehreren Goals, getrennt durch Komma
- Variablenbindung muss **alle** Goals gleichzeitig wahr machen

Beispiel 5.20

Wissensbank

```
woman(mia).  
woman(jody).  
woman(yolanda).
```

```
loves(vincent, mia).  
loves(marsellus, mia).  
loves(pumpkin, honey_bunny).  
loves(honey_bunny, pumpkin).
```


- Anfragen mit mehreren Goals, getrennt durch Komma
- Variablenbindung muss **alle** Goals gleichzeitig wahr machen

Beispiel 5.20

Wissensbank

```
woman(mia).  
woman(jody).  
woman(yolanda).
```

```
loves(vincent, mia).  
loves(marsellus, mia).  
loves(pumpkin, honey_bunny).  
loves(honey_bunny, pumpkin).
```

Interpreter

```
?- loves(marsellus,X), woman(X).
```

- Anfragen mit mehreren Goals, getrennt durch Komma
- Variablenbindung muss **alle** Goals gleichzeitig wahr machen

Beispiel 5.20

Wissensbank

```
woman(mia).  
woman(jody).  
woman(yolanda).
```

```
loves(vincent, mia).  
loves(marsellus, mia).  
loves(pumpkin, honey_bunny).  
loves(honey_bunny, pumpkin).
```

Interpreter

```
?- loves(marsellus,X), woman(X).  
X = mia.
```

- Anfragen mit mehreren Goals, getrennt durch Komma
- Variablenbindung muss **alle** Goals gleichzeitig wahr machen

Beispiel 5.20

Wissensbank

```
woman(mia).  
woman(jody).  
woman(yolanda).
```

```
loves(vincent, mia).  
loves(marsellus, mia).  
loves(pumpkin, honey_bunny).  
loves(honey_bunny, pumpkin).
```

Interpreter

```
?- loves(marsellus,X), woman(X).  
X = mia.  
?- loves(pumpkin,X), woman(X).
```

- Anfragen mit mehreren Goals, getrennt durch Komma
- Variablenbindung muss **alle** Goals gleichzeitig wahr machen

Beispiel 5.20

Wissensbank

```
woman(mia).  
woman(jody).  
woman(yolanda).
```

```
loves(vincent, mia).  
loves(marsellus, mia).  
loves(pumpkin, honey_bunny).  
loves(honey_bunny, pumpkin).
```

Interpreter

```
?- loves(marsellus,X), woman(X).  
X = mia.  
?- loves(pumpkin,X), woman(X).  
false.
```

- Klauseln in der Wissensbank können Variablen enthalten

- Klauseln in der Wissensbank können Variablen enthalten
- Diese sind implizit **universell** quantifiziert

- Klauseln in der Wissensbank können Variablen enthalten
- Diese sind implizit **universell** quantifiziert

Beispiel 5.21

Wissensbank

```
loves(vincent,mia).  
loves(marsellus,mia).  
loves(pumpkin, honey_bunny).  
loves(honey_bunny, pumpkin).  
  
jealous(X,Y):- loves(X,Z), loves(Y,Z).  
  
human(X).
```

- Klauseln in der Wissensbank können Variablen enthalten
- Diese sind implizit **universell** quantifiziert

Beispiel 5.21

Wissensbank

```
loves(vincent,mia).  
loves(marsellus,mia).  
loves(pumpkin, honey_bunny).  
loves(honey_bunny, pumpkin).  
  
jealous(X,Y):- loves(X,Z), loves(Y,Z).  
  
human(X).
```

Interpreter

```
?- jealous(marsellus, W).
```


- Klauseln in der Wissensbank können Variablen enthalten
- Diese sind implizit **universell** quantifiziert

Beispiel 5.21

Wissensbank

```
loves(vincent,mia).  
loves(marsellus,mia).  
loves(pumpkin, honey_bunny).  
loves(honey_bunny, pumpkin).  
  
jealous(X,Y):- loves(X,Z), loves(Y,Z).  
  
human(X).
```

Interpreter

```
?- jealous(marsellus, W).  
W = vincent
```

- Klauseln in der Wissensbank können Variablen enthalten
- Diese sind implizit **universell** quantifiziert

Beispiel 5.21

Wissensbank

```
loves(vincent,mia).  
loves(marsellus,mia).  
loves(pumpkin, honey_bunny).  
loves(honey_bunny, pumpkin).  
  
jealous(X,Y):- loves(X,Z), loves(Y,Z).  
  
human(X).
```

Interpreter

```
?- jealous(marsellus, W).  
W = vincent ;
```

- Klauseln in der Wissensbank können Variablen enthalten
- Diese sind implizit **universell** quantifiziert

Beispiel 5.21

Wissensbank

```
loves(vincent,mia).  
loves(marsellus,mia).  
loves(pumpkin, honey_bunny).  
loves(honey_bunny, pumpkin).  
  
jealous(X,Y):- loves(X,Z), loves(Y,Z).  
  
human(X).
```

Interpreter

```
?- jealous(marsellus, W).  
W = vincent ;  
W = marsellus.
```

- Klauseln in der Wissensbank können Variablen enthalten
- Diese sind implizit **universell** quantifiziert

Beispiel 5.21

Wissensbank

```
loves(vincent,mia).  
loves(marsellus,mia).  
loves(pumpkin, honey_bunny).  
loves(honey_bunny, pumpkin).  
  
jealous(X,Y):- loves(X,Z), loves(Y,Z).  
  
human(X).
```

Interpreter

```
?- jealous(marsellus, W).  
W = vincent ;  
W = marsellus.  
?- human(vincent).
```

- Klauseln in der Wissensbank können Variablen enthalten
- Diese sind implizit **universell** quantifiziert

Beispiel 5.21

Wissensbank

```
loves(vincent,mia).  
loves(marsellus,mia).  
loves(pumpkin, honey_bunny).  
loves(honey_bunny, pumpkin).  
  
jealous(X,Y):- loves(X,Z), loves(Y,Z).  
  
human(X).
```

Interpreter

```
?- jealous(marsellus, W).  
W = vincent ;  
W = marsellus.  
?- human(vincent).  
true.
```

- Klauseln in der Wissensbank können Variablen enthalten
- Diese sind implizit **universell** quantifiziert

Beispiel 5.21

Wissensbank

```
loves(vincent,mia).  
loves(marsellus,mia).  
loves(pumpkin, honey_bunny).  
loves(honey_bunny, pumpkin).  
  
jealous(X,Y):- loves(X,Z), loves(Y,Z).  
  
human(X).
```

Interpreter

```
?- jealous(marsellus, W).  
W = vincent ;  
W = marsellus.  
?- human(vincent).  
true.
```

Mit welcher Anfrage erhält man alle Eifersüchtigen?

Übung 5.22

Formalisieren Sie die folgenden Aussagen als Prolog-Wissensbank:

- 1 Butch ist ein Killer.
- 2 Marsellus und Mia sind miteinander verheiratet.
- 3 Zed ist tot.
- 4 Marsellus tötet jeden, der Mia die Füße massiert.
- 5 Mia liebt jeden guten Tänzer.
- 6 Jules isst alles, was nahrhaft oder lecker ist.

Übung 5.23

Gegeben sei die folgende Wissensbank:

```
hasWand(harry).  
quidditchPlayer(harry).  
wizard(ron).  
wizard(X) :- hasBroom(X), hasWand(X).  
hasBroom(X) :- quidditchPlayer(X).
```

Welche Antworten gibt der Interpreter auf die folgenden Anfragen?

- 1 wizard(ron).
- 2 wizard(hermione).
- 3 wizard(harry).
- 4 wizard(X).

Goal Atom oder komplexer Term

Goal Atom oder komplexer Term

Fakt $\langle \text{Goal} \rangle$.

Goal Atom oder komplexer Term

Fakt $\langle \text{Goal} \rangle .$

Regel $\langle \text{Goal} \rangle \text{ :- } \langle \text{Goal} \rangle [, \langle \text{Goal} \rangle \dots] .$

Goal Atom oder komplexer Term

Fakt $\langle \text{Goal} \rangle .$

Regel $\langle \text{Goal} \rangle :- \langle \text{Goal} \rangle [, \langle \text{Goal} \rangle \dots] .$

Anfrage $\langle \text{Goal} \rangle [, \langle \text{Goal} \rangle \dots] .$

Goal Atom oder komplexer Term

Fakt $\langle \text{Goal} \rangle .$

Regel $\langle \text{Goal} \rangle :- \langle \text{Goal} \rangle [, \langle \text{Goal} \rangle \dots] .$

Anfrage $\langle \text{Goal} \rangle [, \langle \text{Goal} \rangle \dots] .$

- erfolgreich: **true** oder Variablenbindung
weitere Antworten mit ;

Goal Atom oder komplexer Term

Fakt `<Goal> .`

Regel `<Goal> :- <Goal> [, <Goal> ...] .`

Anfrage `<Goal> [, <Goal> ...] .`

- erfolgreich: `true` oder Variablenbindung
weitere Antworten mit `;`
- nicht erfolgreich: `false`

Goal Atom oder komplexer Term

Fakt `<Goal> .`

Regel `<Goal> :- <Goal> [, <Goal> ...] .`

Anfrage `<Goal> [, <Goal> ...] .`

- erfolgreich: `true` oder Variablenbindung
weitere Antworten mit `;`
- nicht erfolgreich: `false`

Goal Atom oder komplexer Term

Fakt <Goal> .

Regel <Goal> :- <Goal> [, <Goal> ...] .

Anfrage <Goal> [, <Goal> ...] .

- erfolgreich: **true** oder Variablenbindung
weitere Antworten mit ;
- nicht erfolgreich: **false**

Variablen

- in Wissensbank: **universell** quantifiziert („Jede Frau ist ein Mensch.“)
- in Anfrage: **existentiell** quantifiziert („Gibt es einen Menschen?“)

1. Einführung

2. Mengen

3. Aussagenlogik

4. Prädikatenlogik

5. Prolog

5.1 Terme

5.2 Fakten, Regeln und Anfragen

5.3 Unifikation, Termgleichheit und Arithmetik

5.4 Horn-Resolution und Suchbäume

5.5 Rekursion

5.6 Listen

5.7 Der Cut

Definition 5.24

Prolog-Terme t und s sind **unifizierbar**, wenn es eine Substitution σ gibt, die die in s und t enthaltenen Variablen so an Terme bindet, dass $\sigma(t) = \sigma(s)$ gilt.

Definition 5.24

Prolog-Terme t und s sind **unifizierbar**, wenn es eine Substitution σ gibt, die die in s und t enthaltenen Variablen so an Terme bindet, dass $\sigma(t) = \sigma(s)$ gilt.

Unterschiede zur Unifikation in der Prädikatenlogik

- keine strikte Trennung zwischen Funktionssymbolen und Prädikaten
X ist unifizierbar mit `woman(mia)`.

Definition 5.24

Prolog-Terme t und s sind **unifizierbar**, wenn es eine Substitution σ gibt, die die in s und t enthaltenen Variablen so an Terme bindet, dass $\sigma(t) = \sigma(s)$ gilt.

Unterschiede zur Unifikation in der Prädikatenlogik

- keine strikte Trennung zwischen Funktionssymbolen und Prädikaten
X ist unifizierbar mit `woman(mia)`.
- kein Occurs-Check
X ist unifizierbar mit `father(X)`.
Abhilfe: `unify_with_occurs_check(X, father(X))`.

Definition 5.24

Prolog-Terme t und s sind **unifizierbar**, wenn es eine Substitution σ gibt, die die in s und t enthaltenen Variablen so an Terme bindet, dass $\sigma(t) = \sigma(s)$ gilt.

Unterschiede zur Unifikation in der Prädikatenlogik

- keine strikte Trennung zwischen Funktionssymbolen und Prädikaten
X ist unifizierbar mit `woman(mia)`.
- kein Occurs-Check
X ist unifizierbar mit `father(X)`.
Abhilfe: `unify_with_occurs_check(X, father(X))`.

Definition 5.24

Prolog-Terme t und s sind **unifizierbar**, wenn es eine Substitution σ gibt, die die in s und t enthaltenen Variablen so an Terme bindet, dass $\sigma(t) = \sigma(s)$ gilt.

Unterschiede zur Unifikation in der Prädikatenlogik

- keine strikte Trennung zwischen Funktionssymbolen und Prädikaten
X ist unifizierbar mit `woman(mia)`.
- kein Occurs-Check
X ist unifizierbar mit `father(X)`.
Abhilfe: `unify_with_occurs_check(X, father(X))`.

Unifikation und Stelligkeit

Wissensbank
<code>married(marsellus, mia).</code>

Definition 5.24

Prolog-Terme t und s sind **unifizierbar**, wenn es eine Substitution σ gibt, die die in s und t enthaltenen Variablen so an Terme bindet, dass $\sigma(t) = \sigma(s)$ gilt.

Unterschiede zur Unifikation in der Prädikatenlogik

- keine strikte Trennung zwischen Funktionssymbolen und Prädikaten
X ist unifizierbar mit `woman(mia)`.
- kein Occurs-Check
X ist unifizierbar mit `father(X)`.
Abhilfe: `unify_with_occurs_check(X, father(X))`.

Unifikation und Stelligkeit

Wissensbank
<code>married(marsellus, mia).</code>

Interpreter
<code>?- married(X).</code> ERROR: Unknown procedure: married/1

Das Prädikat `=/2`

- ist erfolgreich, wenn die beiden Argumente unifizierbar sind
- gibt die für die Unifikation notwendigen Variablenbindungen zurück

Das Prädikat `=/2`

- ist erfolgreich, wenn die beiden Argumente unifizierbar sind
- gibt die für die Unifikation notwendigen Variablenbindungen zurück

Beispiel 5.25

Interpreter

```
?- X = mia.  
X = mia.
```

Unifikation mit =

Das Prädikat =/2

- ist erfolgreich, wenn die beiden Argumente unifizierbar sind
- gibt die für die Unifikation notwendigen Variablenbindungen zurück

Beispiel 5.25

Interpreter

```
?- X = mia.
```

```
X = mia.
```

```
?- woman(X) = woman(mia).
```

```
X = mia.
```

Das Prädikat `=/2`

- ist erfolgreich, wenn die beiden Argumente unifizierbar sind
- gibt die für die Unifikation notwendigen Variablenbindungen zurück

Beispiel 5.25

Interpreter

```
?- X = mia.  
X = mia.  
?- woman(X) = woman(mia).  
X = mia.  
?- loves(X,vincent) = loves(mia,X).  
false.
```

Das Prädikat `=/2`

- ist erfolgreich, wenn die beiden Argumente unifizierbar sind
- gibt die für die Unifikation notwendigen Variablenbindungen zurück

Beispiel 5.25

Interpreter

```
?- X = mia.  
X = mia.  
?- woman(X) = woman(mia).  
X = mia.  
?- loves(X,vincent) = loves(mia,X).  
false.  
?- loves(mia,vincent) = loves(X,X).  
false.
```

Unifikation mit =

Das Prädikat `=/2`

- ist erfolgreich, wenn die beiden Argumente unifizierbar sind
- gibt die für die Unifikation notwendigen Variablenbindungen zurück

Beispiel 5.25

Interpreter

```
?- X = mia.  
X = mia.  
?- woman(X) = woman(mia).  
X = mia.  
?- loves(X,vincent) = loves(mia,X).  
false.  
?- loves(mia,vincent) = loves(X,X).  
false.  
?- knows(father(vincent),X) = knows(Y,mother(mia)).  
X = mother(mia), Y = father(vincent).
```

Übung 5.26

Wie beantwortet der Prolog-Interpreter die folgenden Anfragen?

1 bread = bread.	8 food(X) = food(bread).
2 'Bread' = bread.	9 food(bread,X) = food(Y,sausage).
3 'bread' = bread.	10 food(bread,X,beer) = food(Y,sausage,X).
4 Bread = bread.	11 food(bread,X,beer) = food(Y,kahuna_burger).
5 bread = sausage.	12 food(X) = X.
6 food(bread) = bread.	13 meal(food(bread),drink(beer)) = meal(X,Y).
7 food(bread) = X.	14 meal(food(bread),X) = meal(X,drink(beer)).

- zweistellig ($\backslash=/2$)

Das Prädikat $\backslash=$

- zweistellig ($\backslash=/2$)
- Negation von $=$

Das Prädikat `\=`

- zweistellig (`\=/2`)
- Negation von `=`
- gibt `false` zurück, wenn die Argumente unifizierbar sind

Das Prädikat `\=`

- zweistellig (`\=/2`)
- Negation von `=`
- gibt `false` zurück, wenn die Argumente unifizierbar sind
- gibt `true` zurück, wenn die Argumente `nicht` unifizierbar sind

Das Prädikat `\=`

- zweistellig (`\=/2`)
- Negation von `=`
- gibt `false` zurück, wenn die Argumente unifizierbar sind
- gibt `true` zurück, wenn die Argumente `nicht` unifizierbar sind
- gibt nie Variablenbindungen zurück

Das Prädikat \=

- zweistellig (\=/2)
- Negation von =
- gibt **false** zurück, wenn die Argumente unifizierbar sind
- gibt **true** zurück, wenn die Argumente **nicht** unifizierbar sind
- gibt nie Variablenbindungen zurück

Beispiel 5.27

```
?- mia \= vincent.
```

Das Prädikat \=

- zweistellig (\=/2)
- Negation von =
- gibt **false** zurück, wenn die Argumente unifizierbar sind
- gibt **true** zurück, wenn die Argumente **nicht** unifizierbar sind
- gibt nie Variablenbindungen zurück

Beispiel 5.27

```
?- mia \= vincent.  
true.
```

Das Prädikat \=

- zweistellig (\=/2)
- Negation von =
- gibt **false** zurück, wenn die Argumente unifizierbar sind
- gibt **true** zurück, wenn die Argumente **nicht** unifizierbar sind
- gibt nie Variablenbindungen zurück

Beispiel 5.27

```
?- mia \= vincent.  
true.  
?- a \= b.
```

Das Prädikat \=

- zweistellig (\=/2)
- Negation von =
- gibt **false** zurück, wenn die Argumente unifizierbar sind
- gibt **true** zurück, wenn die Argumente **nicht** unifizierbar sind
- gibt nie Variablenbindungen zurück

Beispiel 5.27

```
?- mia \= vincent.  
true.  
?- a \= b.  
true.
```

Das Prädikat \=

- zweistellig (\=/2)
- Negation von =
- gibt **false** zurück, wenn die Argumente unifizierbar sind
- gibt **true** zurück, wenn die Argumente **nicht** unifizierbar sind
- gibt nie Variablenbindungen zurück

Beispiel 5.27

```
?- mia \= vincent.  
true.  
?- a \= b.  
true.  
?- a \= a.
```


Das Prädikat \=

- zweistellig (\=/2)
- Negation von =
- gibt **false** zurück, wenn die Argumente unifizierbar sind
- gibt **true** zurück, wenn die Argumente **nicht** unifizierbar sind
- gibt nie Variablenbindungen zurück

Beispiel 5.27

```
?- mia \= vincent.  
true.  
?- a \= b.  
true.  
?- a \= a.  
false.
```

Das Prädikat \=

- zweistellig (\=/2)
- Negation von =
- gibt **false** zurück, wenn die Argumente unifizierbar sind
- gibt **true** zurück, wenn die Argumente **nicht** unifizierbar sind
- gibt nie Variablenbindungen zurück

Beispiel 5.27

```
?- mia \= vincent.  
true.  
?- a \= b.  
true.  
?- a \= a.  
false.  
?- a \= B.
```

Das Prädikat \=

- zweistellig (\=/2)
- Negation von =
- gibt **false** zurück, wenn die Argumente unifizierbar sind
- gibt **true** zurück, wenn die Argumente **nicht** unifizierbar sind
- gibt nie Variablenbindungen zurück

Beispiel 5.27

```
?- mia \= vincent.  
true.  
?- a \= b.  
true.  
?- a \= a.  
false.  
?- a \= B.  
false.
```

- zweistellig (\=/2)
- Negation von =
- gibt **false** zurück, wenn die Argumente unifizierbar sind
- gibt **true** zurück, wenn die Argumente **nicht** unifizierbar sind
- gibt nie Variablenbindungen zurück

Beispiel 5.27

```
?- mia \= vincent.  
true.  
?- a \= b.  
true.  
?- a \= a.  
false.  
?- a \= B.  
false.  
? B = b, a \= B.
```

Das Prädikat \=

- zweistellig (\=/2)
- Negation von =
- gibt **false** zurück, wenn die Argumente unifizierbar sind
- gibt **true** zurück, wenn die Argumente **nicht** unifizierbar sind
- gibt nie Variablenbindungen zurück

Beispiel 5.27

```
?- mia \= vincent.  
true.  
?- a \= b.  
true.  
?- a \= a.  
false.  
?- a \= B.  
false.  
? B = b, a \= B.  
B = b.
```

Ein Goal einer Anfrage wird unifiziert

- mit den **Fakten** der Wissensbank

Ein Goal einer Anfrage wird unifiziert

- mit den **Fakten** der Wissensbank
- mit den **Regel-Köpfen** der Wissensbank

Ein Goal einer Anfrage wird unifiziert

- mit den **Fakten** der Wissensbank
- mit den **Regel-Köpfen** der Wissensbank

Beispiel 5.28

Wissensbank

```
vertical(line(point(X,Y), point(X,Z))).  
horizontal(line(point(X,Y), point(Z,Y))).
```


Ein Goal einer Anfrage wird unifiziert

- mit den **Fakten** der Wissensbank
- mit den **Regel-Köpfen** der Wissensbank

Beispiel 5.28

Wissensbank

```
vertical(line(point(X,Y), point(X,Z))).  
horizontal(line(point(X,Y), point(Z,Y))).
```

Interpreter

```
?- horizontal(line(point(1,2), point(3,W))).
```

Ein Goal einer Anfrage wird unifiziert

- mit den **Fakten** der Wissensbank
- mit den **Regel-Köpfen** der Wissensbank

Beispiel 5.28

Wissensbank

```
vertical(line(point(X,Y), point(X,Z))).  
horizontal(line(point(X,Y), point(Z,Y))).
```

Interpreter

```
?- horizontal(line(point(1,2), point(3,W))).  
W = 2.
```

Ein Goal einer Anfrage wird unifiziert

- mit den **Fakten** der Wissensbank
- mit den **Regel-Köpfen** der Wissensbank

Beispiel 5.28

Wissensbank

```
vertical(line(point(X,Y), point(X,Z))).  
horizontal(line(point(X,Y), point(Z,Y))).
```

Interpreter

```
?- horizontal(line(point(1,2), point(3,W))).  
W = 2.  
?- horizontal(line(point(2,3), Point)).
```

Ein Goal einer Anfrage wird unifiziert

- mit den **Fakten** der Wissensbank
- mit den **Regel-Köpfen** der Wissensbank

Beispiel 5.28

Wissensbank

```
vertical(line(point(X,Y), point(X,Z))).  
horizontal(line(point(X,Y), point(Z,Y))).
```

Interpreter

```
?- horizontal(line(point(1,2), point(3,W))).  
W = 2.  
?- horizontal(line(point(2,3), Point)).  
Point = point(_G2239,3).
```

Unifikation für Anfragen

Ein Goal einer Anfrage wird unifiziert

- mit den **Fakten** der Wissensbank
- mit den **Regel-Köpfen** der Wissensbank

Beispiel 5.28

Wissensbank

```
vertical(line(point(X,Y), point(X,Z))).  
horizontal(line(point(X,Y), point(Z,Y))).
```

Interpreter

```
?- horizontal(line(point(1,2), point(3,W))).  
W = 2.  
?- horizontal(line(point(2,3), Point)).  
Point = point(_G2239,3).
```

- **_G2239** ist **ungebundene interne Variable**

Unifikation für Anfragen

Ein Goal einer Anfrage wird unifiziert

- mit den **Fakten** der Wissensbank
- mit den **Regel-Köpfen** der Wissensbank

Beispiel 5.28

Wissensbank

```
vertical(line(point(X,Y), point(X,Z))).  
horizontal(line(point(X,Y), point(Z,Y))).
```

Interpreter

```
?- horizontal(line(point(1,2), point(3,W))).  
W = 2.  
?- horizontal(line(point(2,3), Point)).  
Point = point(_G2239,3).
```

- `_G2239` ist **ungebundende interne Variable**
- Bedeutung: Jede Bindung führt zum Erfolg

Übung 5.29

Gegeben sei die folgende Wissensbank:

```
house_elf(dobby).  
witch(hermione).  
witch('McGonagall').  
wizard(ron).  
magic(X):- house_elf(X).  
magic(X):- wizard(X).  
magic(X):- witch(X).
```

Welche Antworten erzeugen die folgenden Anfragen?

- 1 magic(house_elf).
- 2 wizard(harry).
- 3 magic(wizard).
- 4 magic('McGonagall').
- 5 magic(Hermione).

- ist mit jedem Term unifizierbar

Die anonyme Variable _

- ist mit jedem Term unifizierbar
- kann bei jedem Vorkommen **unterschiedlich** gebunden werden

Die anonyme Variable `_`

- ist mit jedem Term unifizierbar
- kann bei jedem Vorkommen **unterschiedlich** gebunden werden
- Bindung wird nicht zurückgegeben

Die anonyme Variable `_`

- ist mit jedem Term unifizierbar
- kann bei jedem Vorkommen **unterschiedlich** gebunden werden
- Bindung wird nicht zurückgegeben
- Zweck: Platzhalter für Terme, die uninteressant sind

Die anonyme Variable _

- ist mit jedem Term unifizierbar
- kann bei jedem Vorkommen **unterschiedlich** gebunden werden
- Bindung wird nicht zurückgegeben
- Zweck: Platzhalter für Terme, die uninteressant sind

Beispiel 5.30

Wissensbank
<pre>married(mia,marsellus). married(X) :- married(X,_). married(X) :- married(_,X).</pre>

Die anonyme Variable _

- ist mit jedem Term unifizierbar
- kann bei jedem Vorkommen **unterschiedlich** gebunden werden
- Bindung wird nicht zurückgegeben
- Zweck: Platzhalter für Terme, die uninteressant sind

Beispiel 5.30

Wissensbank

```
married(mia,marsellus).
```

```
married(X) :- married(X,_).
```

```
married(X) :- married(_,X).
```

Interpreter

```
?- married(X,Y).
```

Die anonyme Variable _

- ist mit jedem Term unifizierbar
- kann bei jedem Vorkommen **unterschiedlich** gebunden werden
- Bindung wird nicht zurückgegeben
- Zweck: Platzhalter für Terme, die uninteressant sind

Beispiel 5.30

Wissensbank

```
married(mia,marsellus).  
  
married(X) :- married(X,_).  
married(X) :- married(_,X).
```

Interpreter

```
?- married(X,Y).  
X = mia, Y = marsellus.
```

Die anonyme Variable _

- ist mit jedem Term unifizierbar
- kann bei jedem Vorkommen **unterschiedlich** gebunden werden
- Bindung wird nicht zurückgegeben
- Zweck: Platzhalter für Terme, die uninteressant sind

Beispiel 5.30

Wissensbank

```
married(mia,marsellus).  
  
married(X) :- married(X,_).  
married(X) :- married(_,X).
```

Interpreter

```
?- married(X,Y).  
X = mia, Y = marsellus.  
?- married(X,X).
```

Die anonyme Variable _

- ist mit jedem Term unifizierbar
- kann bei jedem Vorkommen **unterschiedlich** gebunden werden
- Bindung wird nicht zurückgegeben
- Zweck: Platzhalter für Terme, die uninteressant sind

Beispiel 5.30

Wissensbank

```
married(mia,marsellus).  
  
married(X) :- married(X,_).  
married(X) :- married(_,X).
```

Interpreter

```
?- married(X,Y).  
X = mia, Y = marsellus.  
?- married(X,X).  
false.
```


Die anonyme Variable _

- ist mit jedem Term unifizierbar
- kann bei jedem Vorkommen **unterschiedlich** gebunden werden
- Bindung wird nicht zurückgegeben
- Zweck: Platzhalter für Terme, die uninteressant sind

Beispiel 5.30

Wissensbank

```
married(mia,marsellus).  
  
married(X) :- married(X,_).  
married(X) :- married(_,X).
```

Interpreter

```
?- married(X,Y).  
X = mia, Y = marsellus.  
?- married(X,X).  
false.  
?- married(_,_).
```

Die anonyme Variable _

- ist mit jedem Term unifizierbar
- kann bei jedem Vorkommen **unterschiedlich** gebunden werden
- Bindung wird nicht zurückgegeben
- Zweck: Platzhalter für Terme, die uninteressant sind

Beispiel 5.30

Wissensbank

```
married(mia,marsellus).  
  
married(X) :- married(X,_).  
married(X) :- married(_,X).
```

Interpreter

```
?- married(X,Y).  
X = mia, Y = marsellus.  
?- married(X,X).  
false.  
?- married(_,_).  
true.
```

Die anonyme Variable _

- ist mit jedem Term unifizierbar
- kann bei jedem Vorkommen **unterschiedlich** gebunden werden
- Bindung wird nicht zurückgegeben
- Zweck: Platzhalter für Terme, die uninteressant sind

Beispiel 5.30

Wissensbank

```
married(mia,marsellus).  
  
married(X) :- married(X,_).  
married(X) :- married(_,X).
```

Interpreter

```
?- married(X,Y).  
X = mia, Y = marsellus.  
?- married(X,X).  
false.  
?- married(_,_).  
true.  
?- married(X).
```

Die anonyme Variable _

- ist mit jedem Term unifizierbar
- kann bei jedem Vorkommen **unterschiedlich** gebunden werden
- Bindung wird nicht zurückgegeben
- Zweck: Platzhalter für Terme, die uninteressant sind

Beispiel 5.30

Wissensbank

```
married(mia,marsellus).  
  
married(X) :- married(X,_).  
married(X) :- married(_,X).
```

Interpreter

```
?- married(X,Y).  
X = mia, Y = marsellus.  
?- married(X,X).  
false.  
?- married(_,_).  
true.  
?- married(X).  
X = mia
```

Die anonyme Variable _

- ist mit jedem Term unifizierbar
- kann bei jedem Vorkommen **unterschiedlich** gebunden werden
- Bindung wird nicht zurückgegeben
- Zweck: Platzhalter für Terme, die uninteressant sind

Beispiel 5.30

Wissensbank

```
married(mia,marsellus).  
  
married(X) :- married(X,_).  
married(X) :- married(_,X).
```

Interpreter

```
?- married(X,Y).  
X = mia, Y = marsellus.  
?- married(X,X).  
false.  
?- married(_,_).  
true.  
?- married(X).  
X = mia ;
```

Die anonyme Variable _

- ist mit jedem Term unifizierbar
- kann bei jedem Vorkommen **unterschiedlich** gebunden werden
- Bindung wird nicht zurückgegeben
- Zweck: Platzhalter für Terme, die uninteressant sind

Beispiel 5.30

Wissensbank

```
married(mia,marsellus).  
  
married(X) :- married(X,_).  
married(X) :- married(_,X).
```

Interpreter

```
?- married(X,Y).  
X = mia, Y = marsellus.  
?- married(X,X).  
false.  
?- married(_,_).  
true.  
?- married(X).  
X = mia ;  
X = marsellus.
```

Übung 5.31

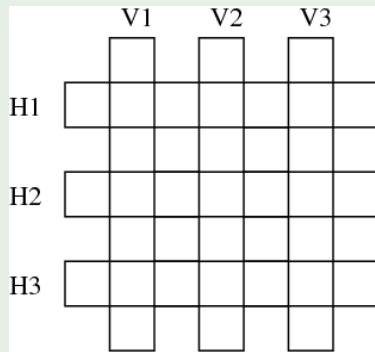
Gegeben sind die folgenden 6 italienischen Wörter:

astante, astoria, baratto, cobalto, pistola, statale.

Schreiben Sie ein Prädikat `crossword/6`, das diese Wörter wie in einem Kreuzworträtsel in das Schema rechts einträgt, so dass die Anfrage `crossword(V1,V2,V3,H1,H2,H3)` eine korrekte Belegung der Zeilen und Spalten ausgibt.

Verwenden Sie hierzu die folgende Wissensbank:

```
word(astante, a,s,t,a,n,t,e).  
word(astoria, a,s,t,o,r,i,a).  
word(baratto, b,a,r,a,t,t,o).  
word(cobalto, c,o,b,a,l,t,o).  
word(pistola, p,i,s,t,o,l,a).  
word(statale, s,t,a,t,a,l,e).
```



Prädikat = testet, ob Argumente **unifizierbar** sind

Prädikat = testet, ob Argumente **unifizierbar** sind

Prädikat == testet, ob Argumente bereits **gleich** sind

Prädikat = testet, ob Argumente **unifizierbar** sind

Prädikat == testet, ob Argumente bereits **gleich** sind

- zweistellig

Prädikat = testet, ob Argumente **unifizierbar** sind

Prädikat == testet, ob Argumente bereits **gleich** sind

- zweistellig
- gibt `true` zurück, wenn beide Argumente derselbe Term sind

Prädikat = testet, ob Argumente **unifizierbar** sind

Prädikat == testet, ob Argumente bereits **gleich** sind

- zweistellig
- gibt **true** zurück, wenn beide Argumente derselbe Term sind
- bindet **nie** Variablen

Prädikat = testet, ob Argumente **unifizierbar** sind

Prädikat == testet, ob Argumente bereits **gleich** sind

- zweistellig
- gibt `true` zurück, wenn beide Argumente derselbe Term sind
- bindet **nie** Variablen
- berücksichtigt bereits bestehende Variablenbindungen

Prädikat = testet, ob Argumente **unifizierbar** sind

Prädikat == testet, ob Argumente bereits **gleich** sind

- zweistellig
- gibt **true** zurück, wenn beide Argumente derselbe Term sind
- bindet **nie** Variablen
- berücksichtigt bereits bestehende Variablenbindungen
- Negation: **\==**

Prädikat = testet, ob Argumente **unifizierbar** sind

Prädikat == testet, ob Argumente bereits **gleich** sind

- zweistellig
- gibt **true** zurück, wenn beide Argumente derselbe Term sind
- bindet **nie** Variablen
- berücksichtigt bereits bestehende Variablenbindungen
- Negation: **\==**

Termgleichheit: Das Prädikat ==

Prädikat = testet, ob Argumente **unifizierbar** sind

Prädikat == testet, ob Argumente bereits **gleich** sind

- zweistellig
- gibt **true** zurück, wenn beide Argumente derselbe Term sind
- bindet **nie** Variablen
- berücksichtigt bereits bestehende Variablenbindungen
- Negation: **\==**

Beispiel 5.32

a = a.	true.	

Termgleichheit: Das Prädikat ==

Prädikat = testet, ob Argumente **unifizierbar** sind

Prädikat == testet, ob Argumente bereits **gleich** sind

- zweistellig
- gibt **true** zurück, wenn beide Argumente derselbe Term sind
- bindet **nie** Variablen
- berücksichtigt bereits bestehende Variablenbindungen
- Negation: **\==**

Beispiel 5.32

a = a.	true.	a == a.	true.

Termgleichheit: Das Prädikat ==

Prädikat = testet, ob Argumente **unifizierbar** sind

Prädikat == testet, ob Argumente bereits **gleich** sind

- zweistellig
- gibt **true** zurück, wenn beide Argumente derselbe Term sind
- bindet **nie** Variablen
- berücksichtigt bereits bestehende Variablenbindungen
- Negation: **\==**

Beispiel 5.32

a = a.	true.	a == a.	true.
a = b.	false.		

Termgleichheit: Das Prädikat ==

Prädikat = testet, ob Argumente **unifizierbar** sind

Prädikat == testet, ob Argumente bereits **gleich** sind

- zweistellig
- gibt **true** zurück, wenn beide Argumente derselbe Term sind
- bindet **nie** Variablen
- berücksichtigt bereits bestehende Variablenbindungen
- Negation: \==

Beispiel 5.32

a = a.	true.	a == a.	true.
a = b.	false.	a == b.	false.

Termgleichheit: Das Prädikat ==

Prädikat = testet, ob Argumente **unifizierbar** sind

Prädikat == testet, ob Argumente bereits **gleich** sind

- zweistellig
- gibt **true** zurück, wenn beide Argumente derselbe Term sind
- bindet **nie** Variablen
- berücksichtigt bereits bestehende Variablenbindungen
- Negation: **\==**

Beispiel 5.32

a = a.	true.	a == a.	true.
a = b.	false.	a == b.	false.
a = B.	B = a.		

Termgleichheit: Das Prädikat ==

Prädikat = testet, ob Argumente **unifizierbar** sind

Prädikat == testet, ob Argumente bereits **gleich** sind

- zweistellig
- gibt **true** zurück, wenn beide Argumente derselbe Term sind
- bindet **nie** Variablen
- berücksichtigt bereits bestehende Variablenbindungen
- Negation: **\==**

Beispiel 5.32

a = a.	true.	a == a.	true.
a = b.	false.	a == b.	false.
a = B.	B = a.	a == B.	false.

Termgleichheit: Das Prädikat ==

Prädikat = testet, ob Argumente **unifizierbar** sind

Prädikat == testet, ob Argumente bereits **gleich** sind

- zweistellig
- gibt **true** zurück, wenn beide Argumente derselbe Term sind
- bindet **nie** Variablen
- berücksichtigt bereits bestehende Variablenbindungen
- Negation: **\==**

Beispiel 5.32

a = a.	true.	a == a.	true.
a = b.	false.	a == b.	false.
a = B.	B = a.	a == B.	false.
a = B, a = B.	B = a.		

Termgleichheit: Das Prädikat ==

Prädikat = testet, ob Argumente **unifizierbar** sind

Prädikat == testet, ob Argumente bereits **gleich** sind

- zweistellig
- gibt **true** zurück, wenn beide Argumente derselbe Term sind
- bindet **nie** Variablen
- berücksichtigt bereits bestehende Variablenbindungen
- Negation: **\==**

Beispiel 5.32

<code>a = a.</code>	true.	<code>a == a.</code>	true.
<code>a = b.</code>	false.	<code>a == b.</code>	false.
<code>a = B.</code>	B = a.	<code>a == B.</code>	false.
<code>a = B, a = B.</code>	B = a.	<code>a = B, a == B.</code>	B = a.

Termgleichheit: Das Prädikat ==

Prädikat = testet, ob Argumente **unifizierbar** sind

Prädikat == testet, ob Argumente bereits **gleich** sind

- zweistellig
- gibt **true** zurück, wenn beide Argumente derselbe Term sind
- bindet **nie** Variablen
- berücksichtigt bereits bestehende Variablenbindungen
- Negation: **\==**

Beispiel 5.32

<code>a = a.</code>	true.	<code>a == a.</code>	true.
<code>a = b.</code>	false.	<code>a == b.</code>	false.
<code>a = B.</code>	B = a.	<code>a == B.</code>	false.
<code>a = B, a = B.</code>	B = a.	<code>a = B, a == B.</code>	B = a.
<code>vincent = 'vincent'.</code>	true.		

Termgleichheit: Das Prädikat ==

Prädikat = testet, ob Argumente **unifizierbar** sind

Prädikat == testet, ob Argumente bereits **gleich** sind

- zweistellig
- gibt **true** zurück, wenn beide Argumente derselbe Term sind
- bindet **nie** Variablen
- berücksichtigt bereits bestehende Variablenbindungen
- Negation: **\==**

Beispiel 5.32

<code>a = a.</code>	true.	<code>a == a.</code>	true.
<code>a = b.</code>	false.	<code>a == b.</code>	false.
<code>a = B.</code>	B = a.	<code>a == B.</code>	false.
<code>a = B, a = B.</code>	B = a.	<code>a = B, a == B.</code>	B = a.
<code>vincent = 'vincent'.</code>	true.	<code>vincent == 'vincent'.</code>	true.

Termgleichheit: Das Prädikat ==

Prädikat = testet, ob Argumente **unifizierbar** sind

Prädikat == testet, ob Argumente bereits **gleich** sind

- zweistellig
- gibt **true** zurück, wenn beide Argumente derselbe Term sind
- bindet **nie** Variablen
- berücksichtigt bereits bestehende Variablenbindungen
- Negation: **\==**

Beispiel 5.32

<code>a = a.</code>	true.	<code>a == a.</code>	true.
<code>a = b.</code>	false.	<code>a == b.</code>	false.
<code>a = B.</code>	B = a.	<code>a == B.</code>	false.
<code>a = B, a = B.</code>	B = a.	<code>a = B, a == B.</code>	B = a.
<code>vincent = 'vincent'.</code>	true.	<code>vincent == 'vincent'.</code>	true.
<code>Vincent = 'Vincent'.</code>	Vincent = 'Vincent'.		

Termgleichheit: Das Prädikat ==

Prädikat = testet, ob Argumente **unifizierbar** sind

Prädikat == testet, ob Argumente bereits **gleich** sind

- zweistellig
- gibt **true** zurück, wenn beide Argumente derselbe Term sind
- bindet **nie** Variablen
- berücksichtigt bereits bestehende Variablenbindungen
- Negation: **\==**

Beispiel 5.32

<code>a = a.</code>	true.	<code>a == a.</code>	true.
<code>a = b.</code>	false.	<code>a == b.</code>	false.
<code>a = B.</code>	B = a.	<code>a == B.</code>	false.
<code>a = B, a = B.</code>	B = a.	<code>a = B, a == B.</code>	B = a.
<code>vincent = 'vincent'.</code>	true.	<code>vincent == 'vincent'.</code>	true.
<code>Vincent = 'Vincent'.</code>	Vincent = 'Vincent'.	<code>Vincent == 'Vincent'.</code>	false.

Übung 5.33

Wie beantwortet der Prolog-Interpreter die folgenden Anfragen?

- 1 `married(X,Y) = married(Y,X).`
- 2 `married(X,Y) == married(Y,X).`
- 3 `married(X,Y) = married(Y,X), married(X,Y) == married(Y,X).`
- 4 `married(X,Y) == married(Y,X), married(X,Y) = married(Y,X).`
- 5 `married(X,Y) \== married(Y,X), married(X,Y) = married(Y,X).`
- 6 `married(X,Y) == married(Y,X), married(X,Y) \= married(Y,X).`
- 7 `married(X,Y) \== married(Y,X), married(X,Y) \= married(Y,X).`
- 8 `loves(X,Y) = loves(mia,mia).`
- 9 `loves(X,X) = loves(Y,lover(Y)).`
- 10 `loves(X,X) = loves(mia,lover(mia)).`

- Das Prädikat $=/2$ führt Unifikation durch

- Das Prädikat $=/2$ führt Unifikation durch
 - Anfrage $X = 2 + 3$ liefert Antwort $X = 2 + 3$

- Das Prädikat $=/2$ führt Unifikation durch
 - Anfrage $X = 2 + 3$ liefert Antwort $X = 2 + 3$
 - kann nicht für Rechnungen genutzt werden

- Das Prädikat $=/2$ führt Unifikation durch
 - Anfrage $X = 2 + 3$ liefert Antwort $X = 2 + 3$
 - kann nicht für Rechnungen genutzt werden
- Das Prädikat $is/2$ führt arithmetische Operationen durch

- Das Prädikat `=/2` führt Unifikation durch
 - Anfrage `X = 2 + 3` liefert Antwort `X = 2 + 3`
 - kann nicht für Rechnungen genutzt werden
- Das Prädikat `is/2` führt arithmetische Operationen durch
 - `X is 2 + 3` liefert Antwort `X = 5`

- Das Prädikat `=/2` führt Unifikation durch
 - Anfrage `X = 2 + 3` liefert Antwort `X = 2 + 3`
 - kann nicht für Rechnungen genutzt werden
- Das Prädikat `is/2` führt arithmetische Operationen durch
 - `X is 2 + 3` liefert Antwort `X = 5`
- Funktionsweise:

- Das Prädikat $=/2$ führt Unifikation durch
 - Anfrage $X = 2 + 3$ liefert Antwort $X = 2 + 3$
 - kann nicht für Rechnungen genutzt werden
- Das Prädikat $is/2$ führt arithmetische Operationen durch
 - $X \text{ is } 2 + 3$ liefert Antwort $X = 5$
- Funktionsweise:
 - rechte Seite wird **arithmetisch ausgewertet**

- Das Prädikat $=/2$ führt Unifikation durch
 - Anfrage $X = 2 + 3$ liefert Antwort $X = 2 + 3$
 - kann nicht für Rechnungen genutzt werden
- Das Prädikat $is/2$ führt arithmetische Operationen durch
 - $X is 2 + 3$ liefert Antwort $X = 5$
- Funktionsweise:
 - rechte Seite wird **arithmetisch ausgewertet**
 - Ergebnis wird mit linker Seite **unifiziert**

- Das Prädikat $=/2$ führt Unifikation durch
 - Anfrage $X = 2 + 3$ liefert Antwort $X = 2 + 3$
 - kann nicht für Rechnungen genutzt werden
- Das Prädikat $is/2$ führt arithmetische Operationen durch
 - $X \text{ is } 2 + 3$ liefert Antwort $X = 5$
- Funktionsweise:
 - rechte Seite wird **arithmetisch ausgewertet**
 - Ergebnis wird mit linker Seite **unifiziert**
- Integer-Operatoren: $+$, $-$, $*$, div , mod

- Das Prädikat $=/2$ führt Unifikation durch
 - Anfrage $X = 2 + 3$ liefert Antwort $X = 2 + 3$
 - kann nicht für Rechnungen genutzt werden
- Das Prädikat $is/2$ führt arithmetische Operationen durch
 - $X \text{ is } 2 + 3$ liefert Antwort $X = 5$
- Funktionsweise:
 - rechte Seite wird **arithmetisch ausgewertet**
 - Ergebnis wird mit linker Seite **unifiziert**
- Integer-Operatoren: $+$, $-$, $*$, **div**, **mod**
 - Operator $/$ liefert nicht immer Ganzzahlen

- Das Prädikat $=/2$ führt Unifikation durch
 - Anfrage $X = 2 + 3$ liefert Antwort $X = 2 + 3$
 - kann nicht für Rechnungen genutzt werden
- Das Prädikat $is/2$ führt arithmetische Operationen durch
 - $X \text{ is } 2 + 3$ liefert Antwort $X = 5$
- Funktionsweise:
 - rechte Seite wird **arithmetisch ausgewertet**
 - Ergebnis wird mit linker Seite **unifiziert**
- Integer-Operatoren: $+$, $-$, $*$, **div**, **mod**
 - Operator $/$ liefert nicht immer Ganzzahlen
 - „Punkt vor Strich“ wird respektiert

- Das Prädikat $=/2$ führt Unifikation durch
 - Anfrage $X = 2 + 3$ liefert Antwort $X = 2 + 3$
 - kann nicht für Rechnungen genutzt werden
- Das Prädikat $is/2$ führt arithmetische Operationen durch
 - $X \text{ is } 2 + 3$ liefert Antwort $X = 5$
- Funktionsweise:
 - rechte Seite wird **arithmetisch ausgewertet**
 - Ergebnis wird mit linker Seite **unifiziert**
- Integer-Operatoren: $+$, $-$, $*$, div , mod
 - Operator $/$ liefert nicht immer Ganzzahlen
 - „Punkt vor Strich“ wird respektiert
 - Klammern sind erlaubt

- Das Prädikat $=/2$ führt Unifikation durch
 - Anfrage $X = 2 + 3$ liefert Antwort $X = 2 + 3$
 - kann nicht für Rechnungen genutzt werden
- Das Prädikat $is/2$ führt arithmetische Operationen durch
 - $X \text{ is } 2 + 3$ liefert Antwort $X = 5$
- Funktionsweise:
 - rechte Seite wird **arithmetisch ausgewertet**
 - Ergebnis wird mit linker Seite **unifiziert**
- Integer-Operatoren: $+$, $-$, $*$, **div**, **mod**
 - Operator $/$ liefert nicht immer Ganzzahlen
 - „Punkt vor Strich“ wird respektiert
 - Klammern sind erlaubt
- Infix- und Präfix-Notation sind gleichwertig
 $X \text{ is } 1 + 2 * 3$ ist derselbe Term wie
 $is(X, +(1, *(2, 3)))$

- Das Prädikat $=/2$ führt Unifikation durch
 - Anfrage $X = 2 + 3$ liefert Antwort $X = 2 + 3$
 - kann nicht für Rechnungen genutzt werden
- Das Prädikat $is/2$ führt arithmetische Operationen durch
 - $X \text{ is } 2 + 3$ liefert Antwort $X = 5$
- Funktionsweise:
 - rechte Seite wird **arithmetisch ausgewertet**
 - Ergebnis wird mit linker Seite **unifiziert**
- Integer-Operatoren: $+$, $-$, $*$, **div**, **mod**
 - Operator $/$ liefert nicht immer Ganzzahlen
 - „Punkt vor Strich“ wird respektiert
 - Klammern sind erlaubt
- Infix- und Präfix-Notation sind gleichwertig
 $X \text{ is } 1 + 2 * 3$ ist derselbe Term wie
 $is(X, +(1, *(2, 3)))$
- Operatoren führen keine Berechnungen durch

- Das Prädikat $=/2$ führt Unifikation durch
 - Anfrage $X = 2 + 3$ liefert Antwort $X = 2 + 3$
 - kann nicht für Rechnungen genutzt werden
- Das Prädikat $is/2$ führt arithmetische Operationen durch
 - $X \text{ is } 2 + 3$ liefert Antwort $X = 5$
- Funktionsweise:
 - rechte Seite wird **arithmetisch ausgewertet**
 - Ergebnis wird mit linker Seite **unifiziert**
- Integer-Operatoren: $+$, $-$, $*$, div , mod
 - Operator $/$ liefert nicht immer Ganzzahlen
 - „Punkt vor Strich“ wird respektiert
 - Klammern sind erlaubt
- Infix- und Präfix-Notation sind gleichwertig
 $X \text{ is } 1 + 2 * 3$ ist derselbe Term wie
 $is(X, +(1, *(2, 3)))$
- Operatoren führen keine Berechnungen durch
 - $+(2, 3)$ ist ein komplexer Term wie $mother(john, mary)$

- Das Prädikat $=/2$ führt Unifikation durch
 - Anfrage $X = 2 + 3$ liefert Antwort $X = 2 + 3$
 - kann nicht für Rechnungen genutzt werden
- Das Prädikat $is/2$ führt arithmetische Operationen durch
 - $X \text{ is } 2 + 3$ liefert Antwort $X = 5$
- Funktionsweise:
 - rechte Seite wird **arithmetisch ausgewertet**
 - Ergebnis wird mit linker Seite **unifiziert**
- Integer-Operatoren: $+$, $-$, $*$, div , mod
 - Operator $/$ liefert nicht immer Ganzzahlen
 - „Punkt vor Strich“ wird respektiert
 - Klammern sind erlaubt
- Infix- und Präfix-Notation sind gleichwertig
 $X \text{ is } 1 + 2 * 3$ ist derselbe Term wie
 $is(X, +(1, *(2, 3)))$
- Operatoren führen keine Berechnungen durch
 - $+(2, 3)$ ist ein komplexer Term wie $mother(john, mary)$
 - Berechnung erfolgt erst durch is

- Das Prädikat `=/2` führt Unifikation durch
 - Anfrage `X = 2 + 3` liefert Antwort `X = 2 + 3`
 - kann nicht für Rechnungen genutzt werden
- Das Prädikat `is/2` führt arithmetische Operationen durch
 - `X is 2 + 3` liefert Antwort `X = 5`
- Funktionsweise:
 - rechte Seite wird **arithmetisch ausgewertet**
 - Ergebnis wird mit linker Seite **unifiziert**
- Integer-Operatoren: `+`, `-`, `*`, `div`, `mod`
 - Operator `/` liefert nicht immer Ganzzahlen
 - „Punkt vor Strich“ wird respektiert
 - Klammern sind erlaubt
- Infix- und Präfix-Notation sind gleichwertig
`X is 1 + 2 * 3` ist derselbe Term wie `is(X, +(1, *(2, 3)))`
- Operatoren führen keine Berechnungen durch
 - `+(2, 3)` ist ein komplexer Term wie `mother(john, mary)`
 - Berechnung erfolgt erst durch `is`

Beispiel 5.34

Interpreter

```
?- X is 1 + 2 * 3.  
X = 7.
```

- Das Prädikat `=/2` führt Unifikation durch
 - Anfrage `X = 2 + 3` liefert Antwort `X = 2 + 3`
 - kann nicht für Rechnungen genutzt werden
- Das Prädikat `is/2` führt arithmetische Operationen durch
 - `X is 2 + 3` liefert Antwort `X = 5`
- Funktionsweise:
 - rechte Seite wird **arithmetisch ausgewertet**
 - Ergebnis wird mit linker Seite **unifiziert**
- Integer-Operatoren: `+`, `-`, `*`, `div`, `mod`
 - Operator `/` liefert nicht immer Ganzzahlen
 - „Punkt vor Strich“ wird respektiert
 - Klammern sind erlaubt
- Infix- und Präfix-Notation sind gleichwertig
`X is 1 + 2 * 3` ist derselbe Term wie `is(X, +(1, *(2, 3)))`
- Operatoren führen keine Berechnungen durch
 - `+(2, 3)` ist ein komplexer Term wie `mother(john, mary)`
 - Berechnung erfolgt erst durch `is`

Beispiel 5.34

Interpreter

```
?- X is 1 + 2 * 3.  
X = 7.  
?- X is (1 + 2) * 3.  
X = 9.
```

- Das Prädikat `=/2` führt Unifikation durch
 - Anfrage `X = 2 + 3` liefert Antwort `X = 2 + 3`
 - kann nicht für Rechnungen genutzt werden
- Das Prädikat `is/2` führt arithmetische Operationen durch
 - `X is 2 + 3` liefert Antwort `X = 5`
- Funktionsweise:
 - rechte Seite wird **arithmetisch ausgewertet**
 - Ergebnis wird mit linker Seite **unifiziert**
- Integer-Operatoren: `+`, `-`, `*`, `div`, `mod`
 - Operator `/` liefert nicht immer Ganzzahlen
 - „Punkt vor Strich“ wird respektiert
 - Klammern sind erlaubt
- Infix- und Präfix-Notation sind gleichwertig
`X is 1 + 2 * 3` ist derselbe Term wie `is(X, +(1, *(2, 3)))`
- Operatoren führen keine Berechnungen durch
 - `+(2, 3)` ist ein komplexer Term wie `mother(john, mary)`
 - Berechnung erfolgt erst durch `is`

Beispiel 5.34

Interpreter

```
?- X is 1 + 2 * 3.  
X = 7.  
?- X is (1 + 2) * 3.  
X = 9.  
?- X is 7 div 3.  
X = 2.
```

- Das Prädikat `=/2` führt Unifikation durch
 - Anfrage `X = 2 + 3` liefert Antwort `X = 2 + 3`
 - kann nicht für Rechnungen genutzt werden
- Das Prädikat `is/2` führt arithmetische Operationen durch
 - `X is 2 + 3` liefert Antwort `X = 5`
- Funktionsweise:
 - rechte Seite wird **arithmetisch ausgewertet**
 - Ergebnis wird mit linker Seite **unifiziert**
- Integer-Operatoren: `+`, `-`, `*`, `div`, `mod`
 - Operator `/` liefert nicht immer Ganzzahlen
 - „Punkt vor Strich“ wird respektiert
 - Klammern sind erlaubt
- Infix- und Präfix-Notation sind gleichwertig
`X is 1 + 2 * 3` ist derselbe Term wie `is(X, +(1, *(2, 3)))`
- Operatoren führen keine Berechnungen durch
 - `+(2, 3)` ist ein komplexer Term wie `mother(john, mary)`
 - Berechnung erfolgt erst durch `is`

Beispiel 5.34

Interpreter

```
?- X is 1 + 2 * 3.  
X = 7.  
?- X is (1 + 2) * 3.  
X = 9.  
?- X is 7 div 3.  
X = 2.  
?- X is 7 mod 3.  
X = 1.
```


- Das Prädikat `=/2` führt Unifikation durch
 - Anfrage `X = 2 + 3` liefert Antwort `X = 2 + 3`
 - kann nicht für Rechnungen genutzt werden
- Das Prädikat `is/2` führt arithmetische Operationen durch
 - `X is 2 + 3` liefert Antwort `X = 5`
- Funktionsweise:
 - rechte Seite wird **arithmetisch ausgewertet**
 - Ergebnis wird mit linker Seite **unifiziert**
- Integer-Operatoren: `+`, `-`, `*`, `div`, `mod`
 - Operator `/` liefert nicht immer Ganzzahlen
 - „Punkt vor Strich“ wird respektiert
 - Klammern sind erlaubt
- Infix- und Präfix-Notation sind gleichwertig
`X is 1 + 2 * 3` ist derselbe Term wie `is(X, +(1, *(2, 3)))`
- Operatoren führen keine Berechnungen durch
 - `+(2, 3)` ist ein komplexer Term wie `mother(john, mary)`
 - Berechnung erfolgt erst durch `is`

Beispiel 5.34

Interpreter

```
?- X is 1 + 2 * 3.  
X = 7.  
?- X is (1 + 2) * 3.  
X = 9.  
?- X is 7 div 3.  
X = 2.  
?- X is 7 mod 3.  
X = 1.  
?- 7 is 1 + 2 * 3.  
true.
```

Beispiel 5.35

Wissensbank

```
square(X,Y) :- Y is X * X.
```

```
inc(X,Y) :- Y is X + 1.
```

```
addThreeAndDouble(X, Y) :- Y is (X+3) * 2.
```

Beispiel 5.35

Wissensbank

```
square(X,Y) :- Y is X * X.  
inc(X,Y) :- Y is X + 1.  
addThreeAndDouble(X, Y) :- Y is (X+3) * 2.
```

Interpreter

```
?- square(-5,Y).  
Y = 25.
```

Beispiel 5.35

Wissensbank

```
square(X,Y) :- Y is X * X.  
inc(X,Y) :- Y is X + 1.  
addThreeAndDouble(X, Y) :- Y is (X+3) * 2.
```

Interpreter

```
?- square(-5,Y).  
Y = 25.  
?- inc(6,Y).  
Y = 7.
```

Beispiel 5.35

Wissensbank

```
square(X,Y) :- Y is X * X.  
inc(X,Y) :- Y is X + 1.  
addThreeAndDouble(X, Y) :- Y is (X+3) * 2.
```

Interpreter

```
?- square(-5,Y).  
Y = 25.  
?- inc(6,Y).  
Y = 7.  
?- addThreeAndDouble(3,Y).  
Y = 12.
```

- alle Variablen auf **rechter** Seite müssen **gebunden** sein

```
?- X is Y + 2.
```

```
ERROR: Arguments are not sufficiently instantiated
```

- alle Variablen auf **rechter** Seite müssen **gebunden** sein

```
?- X is Y + 2.
```

```
ERROR: Arguments are not sufficiently instantiated
```

- rechte Seite muss korrekter arithmetischer Ausdruck sein

```
?- X is 5 + z.
```

```
ERROR: Arithmetic: 'z/0' is not a function
```

- alle Variablen auf **rechter** Seite müssen **gebunden** sein

```
?- X is Y + 2.
```

```
ERROR: Arguments are not sufficiently instantiated
```

- rechte Seite muss korrekter arithmetischer Ausdruck sein

```
?- X is 5 + z.
```

```
ERROR: Arithmetic: 'z/0' is not a function
```

- linke Seite wird **nicht** ausgewertet

```
?- 1 + 2 is 1 + 2.
```

```
false.
```

Der Term 3 ist nicht unifizierbar mit $1 + 2$

Vergleichsprädikate

Arithmetik	Prolog
$x < y$	$X < Y$
$x \leq y$	$X \leq Y$
$x = y$	$X =:= Y$
$x \neq y$	$X \neq Y$
$x \geq y$	$X \geq Y$
$x > y$	$X > Y$

Arithmetik	Prolog
$x < y$	$X < Y$
$x \leq y$	$X \leq Y$
$x = y$	$X =:= Y$
$x \neq y$	$X \neq Y$
$x \geq y$	$X \geq Y$
$x > y$	$X > Y$

■ Funktionsweise:

Arithmetik	Prolog
$x < y$	$X < Y$
$x \leq y$	$X \leq Y$
$x = y$	$X =:= Y$
$x \neq y$	$X \neq Y$
$x \geq y$	$X \geq Y$
$x > y$	$X > Y$

■ Funktionsweise:

- 1 Beide Seiten werden arithmetisch ausgewertet

Arithmetik	Prolog
$x < y$	$X < Y$
$x \leq y$	$X \leq Y$
$x = y$	$X =:= Y$
$x \neq y$	$X \neq Y$
$x \geq y$	$X \geq Y$
$x > y$	$X > Y$

■ Funktionsweise:

- 1 Beide Seiten werden arithmetisch ausgewertet
- 2 Ergebnisse werden verglichen

Arithmetik	Prolog
$x < y$	$X < Y$
$x \leq y$	$X \leq Y$
$x = y$	$X =:= Y$
$x \neq y$	$X \neq Y$
$x \geq y$	$X \geq Y$
$x > y$	$X > Y$

■ Funktionsweise:

- 1 Beide Seiten werden arithmetisch ausgewertet
- 2 Ergebnisse werden verglichen
- 3 Abhängig vom Ergebnis ist das Goal erfolgreich oder scheitert

Arithmetik	Prolog
$x < y$	$X < Y$
$x \leq y$	$X \leq Y$
$x = y$	$X =:= Y$
$x \neq y$	$X \neq Y$
$x \geq y$	$X \geq Y$
$x > y$	$X > Y$

■ Funktionsweise:

- 1 Beide Seiten werden arithmetisch ausgewertet
- 2 Ergebnisse werden verglichen
- 3 Abhängig vom Ergebnis ist das Goal erfolgreich oder scheitert

■ Beide Seiten müssen korrekte arithmetische Terme sein

Arithmetik	Prolog
$x < y$	$X < Y$
$x \leq y$	$X \leq Y$
$x = y$	$X =:= Y$
$x \neq y$	$X \neq Y$
$x \geq y$	$X \geq Y$
$x > y$	$X > Y$

■ Funktionsweise:

- 1 Beide Seiten werden arithmetisch ausgewertet
- 2 Ergebnisse werden verglichen
- 3 Abhängig vom Ergebnis ist das Goal erfolgreich oder scheitert

■ Beide Seiten müssen korrekte arithmetische Terme sein

■ Keine Seite darf ungebundene Variablen enthalten

Arithmetik	Prolog
$x < y$	$X < Y$
$x \leq y$	$X \text{ =< } Y$
$x = y$	$X \text{ =:= } Y$
$x \neq y$	$X \text{ =\= } Y$
$x \geq y$	$X \geq Y$
$x > y$	$X > Y$

■ Funktionsweise:

- 1 Beide Seiten werden arithmetisch ausgewertet
- 2 Ergebnisse werden verglichen
- 3 Abhängig vom Ergebnis ist das Goal erfolgreich oder scheitert

■ Beide Seiten müssen korrekte arithmetische Terme sein

■ Keine Seite darf ungebundene Variablen enthalten

■ Vorsicht: Es heißt =< und nicht <=!

Arithmetik	Prolog
$x < y$	$X < Y$
$x \leq y$	$X \leq Y$
$x = y$	$X =:= Y$
$x \neq y$	$X \neq Y$
$x \geq y$	$X \geq Y$
$x > y$	$X > Y$

■ Funktionsweise:

- 1 Beide Seiten werden arithmetisch ausgewertet
- 2 Ergebnisse werden verglichen
- 3 Abhängig vom Ergebnis ist das Goal erfolgreich oder scheitert

- Beide Seiten müssen korrekte arithmetische Terme sein
- Keine Seite darf ungebundene Variablen enthalten
- Vorsicht: Es heißt $=<$ und nicht $<=$!

Beispiel 5.36

```
?- 1 + 2 =:= 2 + 1.  
true.
```

Arithmetik	Prolog
$x < y$	$X < Y$
$x \leq y$	$X \leq Y$
$x = y$	$X =:= Y$
$x \neq y$	$X \neq Y$
$x \geq y$	$X \geq Y$
$x > y$	$X > Y$

■ Funktionsweise:

- 1 Beide Seiten werden arithmetisch ausgewertet
- 2 Ergebnisse werden verglichen
- 3 Abhängig vom Ergebnis ist das Goal erfolgreich oder scheitert

- Beide Seiten müssen korrekte arithmetische Terme sein
- Keine Seite darf ungebundene Variablen enthalten
- Vorsicht: Es heißt `=<` und nicht `<=`!

Beispiel 5.36

```
?- 1 + 2 =:= 2 + 1.  
true.  
?- 3 > 5.  
false.
```

Arithmetik	Prolog
$x < y$	$X < Y$
$x \leq y$	$X \leq Y$
$x = y$	$X =:= Y$
$x \neq y$	$X \neq Y$
$x \geq y$	$X \geq Y$
$x > y$	$X > Y$

■ Funktionsweise:

- 1 Beide Seiten werden arithmetisch ausgewertet
- 2 Ergebnisse werden verglichen
- 3 Abhängig vom Ergebnis ist das Goal erfolgreich oder scheitert

- Beide Seiten müssen korrekte arithmetische Terme sein
- Keine Seite darf ungebundene Variablen enthalten
- Vorsicht: Es heißt $=<$ und nicht $<=$!

Beispiel 5.36

```
?- 1 + 2 =:= 2 + 1.  
true.  
?- 3 > 5.  
false.  
?- 2 + 3 \= 5.  
false.
```

Arithmetik	Prolog
$x < y$	$X < Y$
$x \leq y$	$X \leq Y$
$x = y$	$X =:= Y$
$x \neq y$	$X \neq Y$
$x \geq y$	$X \geq Y$
$x > y$	$X > Y$

■ Funktionsweise:

- 1 Beide Seiten werden arithmetisch ausgewertet
- 2 Ergebnisse werden verglichen
- 3 Abhängig vom Ergebnis ist das Goal erfolgreich oder scheitert

- Beide Seiten müssen korrekte arithmetische Terme sein
- Keine Seite darf ungebundene Variablen enthalten
- Vorsicht: Es heißt $\leq$ und nicht $<=$!

Beispiel 5.36

```
?- 1 + 2 =:= 2 + 1.  
true.  
?- 3 > 5.  
false.  
?- 2 + 3 \= 5.  
false.  
?- 4 >= 4.  
true.
```

Arithmetik	Prolog
$x < y$	$X < Y$
$x \leq y$	$X \leq Y$
$x = y$	$X =:= Y$
$x \neq y$	$X \neq Y$
$x \geq y$	$X \geq Y$
$x > y$	$X > Y$

■ Funktionsweise:

- 1 Beide Seiten werden arithmetisch ausgewertet
- 2 Ergebnisse werden verglichen
- 3 Abhängig vom Ergebnis ist das Goal erfolgreich oder scheitert

■ Beide Seiten müssen korrekte arithmetische Terme sein

■ Keine Seite darf ungebundene Variablen enthalten

■ Vorsicht: Es heißt $\leq$ und nicht $<=$!

Beispiel 5.36

```
?- 1 + 2 =:= 2 + 1.  
true.  
?- 3 > 5.  
false.  
?- 2 + 3 \= 5.  
false.  
?- 4 >= 4.  
true.
```

```
?- X + 2 \= X + 3.  
ERROR: Arguments are not sufficiently instantiated
```

Arithmetik	Prolog
$x < y$	$X < Y$
$x \leq y$	$X =< Y$
$x = y$	$X =:= Y$
$x \neq y$	$X =\backslash= Y$
$x \geq y$	$X >= Y$
$x > y$	$X > Y$

■ Funktionsweise:

- 1 Beide Seiten werden arithmetisch ausgewertet
- 2 Ergebnisse werden verglichen
- 3 Abhängig vom Ergebnis ist das Goal erfolgreich oder scheitert

■ Beide Seiten müssen korrekte arithmetische Terme sein

■ Keine Seite darf ungebundene Variablen enthalten

■ Vorsicht: Es heißt `=<` und nicht `<=`!

Beispiel 5.36

```
?- 1 + 2 =:= 2 + 1.  
true.  
?- 3 > 5.  
false.  
?- 2 + 3 =\= 5.  
false.  
?- 4 >= 4.  
true.
```

```
?- X + 2 =< X + 3.  
ERROR: Arguments are not sufficiently instantiated  
?- X = 17, X + 2 =< X + 3.  
X = 17.
```

Arithmetik	Prolog
$x < y$	$X < Y$
$x \leq y$	$X \leq Y$
$x = y$	$X =:= Y$
$x \neq y$	$X \neq Y$
$x \geq y$	$X \geq Y$
$x > y$	$X > Y$

■ Funktionsweise:

- 1 Beide Seiten werden arithmetisch ausgewertet
- 2 Ergebnisse werden verglichen
- 3 Abhängig vom Ergebnis ist das Goal erfolgreich oder scheitert

■ Beide Seiten müssen korrekte arithmetische Terme sein

■ Keine Seite darf ungebundene Variablen enthalten

■ Vorsicht: Es heißt $\leq$ und nicht $<=$!

Beispiel 5.36

```
?- 1 + 2 =:= 2 + 1.  
true.  
?- 3 > 5.  
false.  
?- 2 + 3 \= 5.  
false.  
?- 4 >= 4.  
true.
```

```
?- X + 2 <= X + 3.  
ERROR: Arguments are not sufficiently instantiated  
?- X = 17, X + 2 <= X + 3.  
X = 17.  
?- X = 3 + Y, Y = 2 * 4, X + 2 =:= 3 + 2 * 5.  
X = 3 + 2 * 4, Y = 2 * 4.
```

Arithmetik	Prolog
$x < y$	$X < Y$
$x \leq y$	$X \leq Y$
$x = y$	$X =:= Y$
$x \neq y$	$X \neq Y$
$x \geq y$	$X \geq Y$
$x > y$	$X > Y$

■ Funktionsweise:

- 1 Beide Seiten werden arithmetisch ausgewertet
- 2 Ergebnisse werden verglichen
- 3 Abhängig vom Ergebnis ist das Goal erfolgreich oder scheitert

■ Beide Seiten müssen korrekte arithmetische Terme sein

■ Keine Seite darf ungebundene Variablen enthalten

■ Vorsicht: Es heißt $\leq$ und nicht $<=$!

Beispiel 5.36

```
?- 1 + 2 =:= 2 + 1.
```

```
true.
```

```
?- 3 > 5.
```

```
false.
```

```
?- 2 + 3 \= 5.
```

```
false.
```

```
?- 4 >= 4.
```

```
true.
```

```
?- X + 2 <= X + 3.
```

```
ERROR: Arguments are not sufficiently instantiated
```

```
?- X = 17, X + 2 <= X + 3.
```

```
X = 17.
```

```
?- X = 3 + Y, Y = 2 * 4, X + 2 =:= 3 + 2 * 5.
```

```
X = 3 + 2 * 4, Y = 2 * 4.
```

```
?- X =:= 2 + 3.
```

```
ERROR: Arguments are not sufficiently instantiated
```


Übung 5.37

Welche Antworten liefern die folgenden Anfragen?

1 `14 is 2 * 7.`

2 `14 =\= 2 * 6.`

3 `14 = 2 * 7.`

4 `14 == 2 * 7.`

5 `14 \== 2 * 7.`

6 `14 =:= 2 * 7.`

7 `14 \= X * Y.`

8 `2 + 3 == 3 + 2.`

9 `2 + 3 =:= 3 + 2.`

10 `7 - 2 =\= 9 - 2.`

11 `p == 'p'.`

12 `p =\= 'p'.`

13 `vincent == VAR.`

14 `vincent = VAR, VAR == vincent.`

- Unifikation bindet Variablen an andere Terme

- Unifikation bindet Variablen an andere Terme
 - implizit bei Suche nach passendem Goal in Wissensbank

- Unifikation bindet Variablen an andere Terme
 - implizit bei Suche nach passendem Goal in Wissensbank
 - explizit durch Prädikat =

- Unifikation bindet Variablen an andere Terme
 - implizit bei Suche nach passendem Goal in Wissensbank
 - explizit durch Prädikat =
- Anonyme Variable _ kann bei jedem Auftreten neu gebunden werden

- Unifikation bindet Variablen an andere Terme
 - implizit bei Suche nach passendem Goal in Wissensbank
 - explizit durch Prädikat =
- Anonyme Variable _ kann bei jedem Auftreten neu gebunden werden
- Prädikat == testet Termgleichheit

- Unifikation bindet Variablen an andere Terme
 - implizit bei Suche nach passendem Goal in Wissensbank
 - explizit durch Prädikat =
- Anonyme Variable _ kann bei jedem Auftreten neu gebunden werden
- Prädikat == testet Termgleichheit
 - bindet keine Variablen

- Unifikation bindet Variablen an andere Terme
 - implizit bei Suche nach passendem Goal in Wissensbank
 - explizit durch Prädikat =
- Anonyme Variable _ kann bei jedem Auftreten neu gebunden werden
- Prädikat == testet Termgleichheit
 - bindet keine Variablen
- Prädikat is wertet rechte Seite aus und unifiziert Ergebnis mit linker Seite

- Unifikation bindet Variablen an andere Terme
 - implizit bei Suche nach passendem Goal in Wissensbank
 - explizit durch Prädikat =
- Anonyme Variable _ kann bei jedem Auftreten neu gebunden werden
- Prädikat == testet Termgleichheit
 - bindet keine Variablen
- Prädikat is wertet rechte Seite aus und unifiziert Ergebnis mit linker Seite
- Prädikate ==, \=, >, <, >=, <= werten beide Seiten aus und vergleichen Ergebnisse

1. Einführung

2. Mengen

3. Aussagenlogik

4. Prädikatenlogik

5. Prolog

5.1 Terme

5.2 Fakten, Regeln und Anfragen

5.3 Unifikation, Termgleichheit und Arithmetik

5.4 Horn-Resolution und Suchbäume

5.5 Rekursion

5.6 Listen

5.7 Der Cut

$$R_1 \wedge R_2 \wedge \dots \wedge R_n \rightarrow K$$

$$R_1 \wedge R_2 \wedge \dots \wedge R_n \rightarrow K$$

- erstmals beschrieben 1951 von Alfred Horn



Alfred Horn
(1918–2001)

$$R_1 \wedge R_2 \wedge \dots \wedge R_n \rightarrow K$$

- erstmals beschrieben 1951 von Alfred Horn
- erlauben effiziente Schlussfolgerungsverfahren



Alfred Horn
(1918–2001)

$$R_1 \wedge R_2 \wedge \dots \wedge R_n \rightarrow K$$

- erstmals beschrieben 1951 von Alfred Horn
- erlauben effiziente Schlussfolgerungsverfahren
 - Aussagenlogik: P-vollständig statt NP-vollständig



Alfred Horn
(1918–2001)

$$R_1 \wedge R_2 \wedge \dots \wedge R_n \rightarrow K$$

- erstmals beschrieben 1951 von Alfred Horn
- erlauben effiziente Schlussfolgerungsverfahren
 - Aussagenlogik: P-vollständig statt NP-vollständig
 - Prädikatenlogik: weniger Verzweigungen



Alfred Horn
(1918–2001)

$$R_1 \wedge R_2 \wedge \dots \wedge R_n \rightarrow K$$

- erstmals beschrieben 1951 von Alfred Horn
- erlauben effiziente Schlussfolgerungsverfahren
 - Aussagenlogik: P-vollständig statt NP-vollständig
 - Prädikatenlogik: weniger Verzweigungen

Definition 5.38

Eine prädikatenlogische **Horn-Formel** hat die Form
Rumpf $\rightarrow$ Kopf, wobei



Alfred Horn
(1918–2001)

$$R_1 \wedge R_2 \wedge \dots \wedge R_n \rightarrow K$$

- erstmals beschrieben 1951 von Alfred Horn
- erlauben effiziente Schlussfolgerungsverfahren
 - Aussagenlogik: P-vollständig statt NP-vollständig
 - Prädikatenlogik: weniger Verzweigungen

Definition 5.38

Eine prädikatenlogische **Horn-Formel** hat die Form
Rumpf $\rightarrow$ Kopf, wobei

Kopf ein Atom oder **F** ist;



Alfred Horn
(1918–2001)

$$R_1 \wedge R_2 \wedge \dots \wedge R_n \rightarrow K$$

- erstmals beschrieben 1951 von Alfred Horn
- erlauben effiziente Schlussfolgerungsverfahren
 - Aussagenlogik: P-vollständig statt NP-vollständig
 - Prädikatenlogik: weniger Verzweigungen

Definition 5.38

Eine prädikatenlogische **Horn-Formel** hat die Form
Rumpf $\rightarrow$ Kopf, wobei

Kopf ein Atom oder **F** ist;

Rumpf eine Konjunktion von Atomen oder **W** ist.



Alfred Horn
(1918–2001)

$$R_1 \wedge R_2 \wedge \dots \wedge R_n \rightarrow K$$

- erstmals beschrieben 1951 von Alfred Horn
- erlauben effiziente Schlussfolgerungsverfahren
 - Aussagenlogik: P-vollständig statt NP-vollständig
 - Prädikatenlogik: weniger Verzweigungen

Definition 5.38

Eine prädikatenlogische **Horn-Formel** hat die Form
Rumpf $\rightarrow$ Kopf, wobei

Kopf ein Atom oder **F** ist;

Rumpf eine Konjunktion von Atomen oder **W** ist.

Eine **Horn-Klausel** ist eine in KNF transformierte Horn-Formel.



Alfred Horn
(1918–2001)

	Kopf ist Atom <i>definite</i> Klausel	Kopf ist F <i>Ziel</i> -Klausel
Rumpf ist Konjunktion		
Rumpf ist W		

	Kopf ist Atom <i>definite</i> Klausel	Kopf ist F <i>Ziel</i> -Klausel
Rumpf ist Konjunktion	$B(x, y) \wedge K(y, z) \rightarrow O(z, x)$	
Rumpf ist W		

	Kopf ist Atom <i>definite</i> Klausel	Kopf ist F <i>Ziel</i> -Klausel
Rumpf ist Konjunktion	$B(x, y) \wedge K(y, z) \rightarrow O(z, x)$ $\{\neg B(x, y), \neg K(y, z), O(z, x)\}$	
Rumpf ist W		

	Kopf ist Atom definite Klausel	Kopf ist F Ziel-Klausel
Rumpf ist Konjunktion	$B(x, y) \wedge K(y, z) \rightarrow O(z, x)$ $\{\neg B(x, y), \neg K(y, z), O(z, x)\}$ Regel-Klausel	
Rumpf ist W		

	Kopf ist Atom <i>definite</i> Klausel	Kopf ist F <i>Ziel</i> -Klausel
Rumpf ist Konjunktion	$B(x, y) \wedge K(y, z) \rightarrow O(z, x)$ $\{\neg B(x, y), \neg K(y, z), O(z, x)\}$ <i>Regel</i> -Klausel	
Rumpf ist W	$\mathbf{W} \rightarrow B(a, b)$	

	Kopf ist Atom <i>definite</i> Klausel	Kopf ist F <i>Ziel</i> -Klausel
Rumpf ist Konjunktion	$B(x, y) \wedge K(y, z) \rightarrow O(z, x)$ $\{\neg B(x, y), \neg K(y, z), O(z, x)\}$ <i>Regel</i> -Klausel	
Rumpf ist W	$\mathbf{W} \rightarrow B(a, b)$ $\{B(a, b)\}$	

	Kopf ist Atom definite Klausel	Kopf ist F Ziel-Klausel
Rumpf ist Konjunktion	$B(x, y) \wedge K(y, z) \rightarrow O(z, x)$ $\{\neg B(x, y), \neg K(y, z), O(z, x)\}$ Regel-Klausel	
Rumpf ist W	$\mathbf{W} \rightarrow B(a, b)$ $\{B(a, b)\}$ Fakten-Klausel	

	Kopf ist Atom <i>definite</i> Klausel	Kopf ist F <i>Ziel</i> -Klausel
Rumpf ist Konjunktion	$B(x, y) \wedge K(y, z) \rightarrow O(z, x)$ $\{\neg B(x, y), \neg K(y, z), O(z, x)\}$ <i>Regel</i> -Klausel	$K(x, y) \wedge W(x) \rightarrow \mathbf{F}$
Rumpf ist W	$\mathbf{W} \rightarrow B(a, b)$ $\{B(a, b)\}$ <i>Fakten</i> -Klausel	

	Kopf ist Atom definite Klausel	Kopf ist F Ziel -Klausel
Rumpf ist Konjunktion	$B(x, y) \wedge K(y, z) \rightarrow O(z, x)$ $\{\neg B(x, y), \neg K(y, z), O(z, x)\}$ Regel -Klausel	$K(x, y) \wedge W(x) \rightarrow \mathbf{F}$ $\{\neg K(x, y), \neg W(x)\}$
Rumpf ist W	$\mathbf{W} \rightarrow B(a, b)$ $\{B(a, b)\}$ Fakten -Klausel	

	Kopf ist Atom <i>definite</i> Klausel	Kopf ist F <i>Ziel</i> -Klausel
Rumpf ist Konjunktion	$B(x, y) \wedge K(y, z) \rightarrow O(z, x)$ $\{\neg B(x, y), \neg K(y, z), O(z, x)\}$ <i>Regel</i> -Klausel	$K(x, y) \wedge W(x) \rightarrow \mathbf{F}$ $\{\neg K(x, y), \neg W(x)\}$ <i>Anfrage</i> -Klausel
Rumpf ist W	$\mathbf{W} \rightarrow B(a, b)$ $\{B(a, b)\}$ <i>Fakten</i> -Klausel	

	Kopf ist Atom definite Klausel	Kopf ist F Ziel -Klausel
Rumpf ist Konjunktion	$B(x, y) \wedge K(y, z) \rightarrow O(z, x)$ $\{\neg B(x, y), \neg K(y, z), O(z, x)\}$ Regel -Klausel	$K(x, y) \wedge W(x) \rightarrow \mathbf{F}$ $\{\neg K(x, y), \neg W(x)\}$ Anfrage -Klausel
Rumpf ist W	$\mathbf{W} \rightarrow B(a, b)$ $\{B(a, b)\}$ Fakten -Klausel	$\mathbf{W} \rightarrow \mathbf{F}$ $\{\}$ leere Klausel

	Kopf ist Atom <i>definite</i> Klausel	Kopf ist F <i>Ziel</i> -Klausel
Rumpf ist Konjunktion	$B(x, y) \wedge K(y, z) \rightarrow O(z, x)$ $\{\neg B(x, y), \neg K(y, z), O(z, x)\}$ <i>Regel</i> -Klausel	$K(x, y) \wedge W(x) \rightarrow \mathbf{F}$ $\{\neg K(x, y), \neg W(x)\}$ <i>Anfrage</i> -Klausel
Rumpf ist W	$\mathbf{W} \rightarrow B(a, b)$ $\{B(a, b)\}$ <i>Fakten</i> -Klausel	$\mathbf{W} \rightarrow \mathbf{F}$ $\{\}$ leere Klausel

- Horn-Klauseln haben höchstens ein positives Literal

	Kopf ist Atom <i>definite</i> Klausel	Kopf ist F <i>Ziel</i> -Klausel
Rumpf ist Konjunktion	$B(x, y) \wedge K(y, z) \rightarrow O(z, x)$ $\{\neg B(x, y), \neg K(y, z), O(z, x)\}$ <i>Regel</i> -Klausel	$K(x, y) \wedge W(x) \rightarrow \mathbf{F}$ $\{\neg K(x, y), \neg W(x)\}$ <i>Anfrage</i> -Klausel
Rumpf ist W	$\mathbf{W} \rightarrow B(a, b)$ $\{B(a, b)\}$ <i>Fakten</i> -Klausel	$\mathbf{W} \rightarrow \mathbf{F}$ $\{\}$ leere Klausel

- Horn-Klauseln haben höchstens ein positives Literal
- Resolution von zwei definiten Klauseln erzeugt eine definite Klausel

	Kopf ist Atom definite Klausel	Kopf ist F Ziel -Klausel
Rumpf ist Konjunktion	$B(x, y) \wedge K(y, z) \rightarrow O(z, x)$ $\{\neg B(x, y), \neg K(y, z), O(z, x)\}$ Regel -Klausel	$K(x, y) \wedge W(x) \rightarrow \mathbf{F}$ $\{\neg K(x, y), \neg W(x)\}$ Anfrage -Klausel
Rumpf ist W	$\mathbf{W} \rightarrow B(a, b)$ $\{B(a, b)\}$ Fakten -Klausel	$\mathbf{W} \rightarrow \mathbf{F}$ $\{\}$ leere Klausel

- Horn-Klauseln haben höchstens ein positives Literal
- Resolution von zwei definiten Klauseln erzeugt eine definite Klausel
- Resolution einer Anfrage- mit einer definiten Klausel erzeugt eine Zielklausel

	Kopf ist Atom <i>definite</i> Klausel	Kopf ist F <i>Ziel</i> -Klausel
Rumpf ist Konjunktion	$B(x, y) \wedge K(y, z) \rightarrow O(z, x)$ $\{\neg B(x, y), \neg K(y, z), O(z, x)\}$ <i>Regel</i> -Klausel	$K(x, y) \wedge W(x) \rightarrow \mathbf{F}$ $\{\neg K(x, y), \neg W(x)\}$ <i>Anfrage</i> -Klausel
Rumpf ist W	$\mathbf{W} \rightarrow B(a, b)$ $\{B(a, b)\}$ <i>Fakten</i> -Klausel	$\mathbf{W} \rightarrow \mathbf{F}$ $\{\}$ leere Klausel

- Horn-Klauseln haben höchstens ein positives Literal
- Resolution von zwei definiten Klauseln erzeugt eine definite Klausel
- Resolution einer Anfrage- mit einer definiten Klausel erzeugt eine Zielklausel
- Resolution von zwei Zielklauseln ist nicht möglich

Beispiel 5.39

Klausel	Prolog-Syntax	Logik-Syntax	natürliche Sprache
Fakt	<code>human(vincent)</code>	$H(v)$	Vincent ist ein Mensch.
	<code>male(vincent)</code>	$M(v)$	Vincent ist männlich.
	<code>human(mia)</code>	$H(m)$	Mia ist ein Mensch.
	<code>female(mia)</code>	$F(m)$	Mia ist weiblich.

Beispiel 5.39

Klausel	Prolog-Syntax	Logik-Syntax	natürliche Sprache
Fakt	<code>human(vincent)</code>	$H(v)$	Vincent ist ein Mensch.
	<code>male(vincent)</code>	$M(v)$	Vincent ist männlich.
	<code>human(mia)</code>	$H(m)$	Mia ist ein Mensch.
	<code>female(mia)</code>	$F(m)$	Mia ist weiblich.
Regel	<code>woman(X) :- human(X), female(X)</code>	$\forall x (H(x) \wedge F(x) \rightarrow W(x))$	Wer menschlich und weiblich ist, ist eine Frau.

Beispiel 5.39

Klausel	Prolog-Syntax	Logik-Syntax	natürliche Sprache
Fakt	human(vincent)	$H(v)$	Vincent ist ein Mensch.
	male(vincent)	$M(v)$	Vincent ist männlich.
	human(mia)	$H(m)$	Mia ist ein Mensch.
	female(mia)	$F(m)$	Mia ist weiblich.
Regel	woman(X) :- human(X), female(X)	$\forall x (H(x) \wedge F(x) \rightarrow W(x))$	Wer menschlich und weiblich ist, ist eine Frau.
Anfrage	woman(X)	$\exists x W(x)$	Gibt es eine Frau?

Beispiel 5.39

Klausel	Prolog-Syntax	Logik-Syntax	natürliche Sprache
Fakt	human(vincent)	$H(v)$	Vincent ist ein Mensch.
	male(vincent)	$M(v)$	Vincent ist männlich.
	human(mia)	$H(m)$	Mia ist ein Mensch.
	female(mia)	$F(m)$	Mia ist weiblich.
Regel	woman(X) :- human(X), female(X)	$\forall x (H(x) \wedge F(x) \rightarrow W(x))$	Wer menschlich und weiblich ist, ist eine Frau.
Anfrage	woman(X)	$\exists x W(x)$	Gibt es eine Frau?

- Gesuchte Antwort: Folgt die Anfrage Q aus der Wissensbank K ?

Beispiel 5.39

Klausel	Prolog-Syntax	Logik-Syntax	natürliche Sprache
Fakt	human(vincent)	$H(v)$	Vincent ist ein Mensch.
	male(vincent)	$M(v)$	Vincent ist männlich.
	human(mia)	$H(m)$	Mia ist ein Mensch.
	female(mia)	$F(m)$	Mia ist weiblich.
Regel	woman(X) :- human(X), female(X)	$\forall x (H(x) \wedge F(x) \rightarrow W(x))$	Wer menschlich und weiblich ist, ist eine Frau.
Anfrage	woman(X)	$\exists x W(x)$	Gibt es eine Frau?

■ Gesuchte Antwort: Folgt die Anfrage Q aus der Wissensbank K ?

■ teste mittels Resolution die Erfüllbarkeit $K \wedge \neg Q$

Beispiel 5.39

Klausel	Prolog-Syntax	Logik-Syntax	natürliche Sprache
Fakt	human(vincent)	$H(v)$	Vincent ist ein Mensch.
	male(vincent)	$M(v)$	Vincent ist männlich.
	human(mia)	$H(m)$	Mia ist ein Mensch.
	female(mia)	$F(m)$	Mia ist weiblich.
Regel	woman(X) :- human(X), female(X)	$\forall x (H(x) \wedge F(x) \rightarrow W(x))$	Wer menschlich und weiblich ist, ist eine Frau.
Anfrage	woman(X)	$\exists x W(x)$	Gibt es eine Frau?

■ Gesuchte Antwort: Folgt die Anfrage Q aus der Wissensbank K ?

- teste mittels Resolution die Erfüllbarkeit $K \wedge \neg Q$
- im Beispiel: $\forall x \neg W(x)$

Beispiel 5.39

Klausel	Prolog-Syntax	Logik-Syntax	natürliche Sprache
Fakt	human(vincent)	$H(v)$	Vincent ist ein Mensch.
	male(vincent)	$M(v)$	Vincent ist männlich.
	human(mia)	$H(m)$	Mia ist ein Mensch.
	female(mia)	$F(m)$	Mia ist weiblich.
Regel	woman(X) :- human(X), female(X)	$\forall x (H(x) \wedge F(x) \rightarrow W(x))$	Wer menschlich und weiblich ist, ist eine Frau.
Anfrage	woman(X)	$\exists x W(x)$	Gibt es eine Frau?

- Gesuchte Antwort: Folgt die Anfrage Q aus der Wissensbank K ?
 - teste mittels Resolution die Erfüllbarkeit $K \wedge \neg Q$
 - im Beispiel: $\forall x \neg W(x)$
- Negation wird in Prolog nicht explizit angegeben

Beispiel 5.39

Klausel	Prolog-Syntax	Logik-Syntax	natürliche Sprache
Fakt	<code>human(vincent)</code>	$H(v)$	Vincent ist ein Mensch.
	<code>male(vincent)</code>	$M(v)$	Vincent ist männlich.
	<code>human(mia)</code>	$H(m)$	Mia ist ein Mensch.
	<code>female(mia)</code>	$F(m)$	Mia ist weiblich.
Regel	<code>woman(X) :- human(X), female(X)</code>	$\forall x (H(x) \wedge F(x) \rightarrow W(x))$	Wer menschlich und weiblich ist, ist eine Frau.
Anfrage	<code>woman(X)</code>	$\exists x W(x)$	Gibt es eine Frau?

- Gesuchte Antwort: Folgt die Anfrage Q aus der Wissensbank K ?
 - teste mittels Resolution die Erfüllbarkeit $K \wedge \neg Q$
 - im Beispiel: $\forall x \neg W(x)$
- Negation wird in Prolog nicht explizit angegeben
 - Regelköpfe (einschließlich Fakten) sind positiv

Beispiel 5.39

Klausel	Prolog-Syntax	Logik-Syntax	natürliche Sprache
Fakt	<code>human(vincent)</code>	$H(v)$	Vincent ist ein Mensch.
	<code>male(vincent)</code>	$M(v)$	Vincent ist männlich.
	<code>human(mia)</code>	$H(m)$	Mia ist ein Mensch.
	<code>female(mia)</code>	$F(m)$	Mia ist weiblich.
Regel	<code>woman(X) :- human(X), female(X)</code>	$\forall x (H(x) \wedge F(x) \rightarrow W(x))$	Wer menschlich und weiblich ist, ist eine Frau.
Anfrage	<code>woman(X)</code>	$\exists x W(x)$	Gibt es eine Frau?

- Gesuchte Antwort: Folgt die Anfrage Q aus der Wissensbank K ?
 - teste mittels Resolution die Erfüllbarkeit $K \wedge \neg Q$
 - im Beispiel: $\forall x \neg W(x)$
- Negation wird in Prolog nicht explizit angegeben
 - Regelköpfe (einschließlich Fakten) sind positiv
 - Regelrumpfe und Anfragen sind implizit negiert

SLD-Resolution

Selective Wähle nächstes Literal (In Prolog: zuletzt eingeführtes)

SLD-Resolution

Selective Wähle nächstes Literal (In Prolog: zuletzt eingeführtes)

Linear Erste Elternklausel ist zuletzt erzeugte Zielklausel

SLD-Resolution

Selective Wähle nächstes Literal (In Prolog: zuletzt eingeführtes)

Linear Erste Elternklausel ist zuletzt erzeugte Zielklausel

Definite clause Zweite Elternklausel ist definite Klausel

SLD-Resolution

Selective Wähle nächstes Literal (In Prolog: zuletzt eingeführtes)

Linear Erste Elternklausel ist zuletzt erzeugte Zielklausel

Definite clause Zweite Elternklausel ist definite Klausel

SLD-Resolution

Selective Wähle nächstes Literal (In Prolog: zuletzt eingeführtes)

Linear Erste Elternklausel ist zuletzt erzeugte Zielklausel

Definite clause Zweite Elternklausel ist definite Klausel

Methode

- lege Atome (Goals) der Anfrage auf **Stack**

SLD-Resolution

Selective Wähle nächstes Literal (In Prolog: zuletzt eingeführtes)

Linear Erste Elternklausel ist zuletzt erzeugte Zielklausel

Definite clause Zweite Elternklausel ist definite Klausel

Methode

- lege Atome (Goals) der Anfrage auf **Stack**
- unifiziere oberstes Goal mit allen Regelköpfen

SLD-Resolution

Selective Wähle nächstes Literal (In Prolog: zuletzt eingeführtes)

Linear Erste Elternklausel ist zuletzt erzeugte Zielklausel

Definite clause Zweite Elternklausel ist definite Klausel

Methode

- lege Atome (Goals) der Anfrage auf **Stack**
- unifiziere oberstes Goal mit allen Regelköpfen
- wenn erfolgreich: ersetze Goal durch Regelrumpf (evtl. mehrere Möglichkeiten)

SLD-Resolution

Selective Wähle nächstes Literal (In Prolog: zuletzt eingeführtes)

Linear Erste Elternklausel ist zuletzt erzeugte Zielklausel

Definite clause Zweite Elternklausel ist definite Klausel

Methode

- lege Atome (Goals) der Anfrage auf **Stack**
- unifiziere oberstes Goal mit allen Regelköpfen
- wenn erfolgreich: ersetze Goal durch Regelrumpf (evtl. mehrere Möglichkeiten)
- wenn Stack leer: Anfrageklausel ist unerfüllbar bzgl. Wissensbank

SLD-Resolution

Selective Wähle nächstes Literal (In Prolog: zuletzt eingeführtes)

Linear Erste Elternklausel ist zuletzt erzeugte Zielklausel

Definite clause Zweite Elternklausel ist definite Klausel

Methode

- lege Atome (Goals) der Anfrage auf **Stack**
- unifiziere oberstes Goal mit allen Regelköpfen
- wenn erfolgreich: ersetze Goal durch Regelrumpf (evtl. mehrere Möglichkeiten)
- wenn Stack leer: Anfrageklausel ist unerfüllbar bzgl. Wissensbank

SLD-Resolution

Selective Wähle nächstes Literal (In Prolog: zuletzt eingeführtes)

Linear Erste Elternklausel ist zuletzt erzeugte Zielklausel

Definite clause Zweite Elternklausel ist definite Klausel

Methode

- lege Atome (Goals) der Anfrage auf **Stack**
- unifiziere oberstes Goal mit allen Regelköpfen
- wenn erfolgreich: ersetze Goal durch Regelrumpf (evtl. mehrere Möglichkeiten)
- wenn Stack leer: Anfrageklausel ist unerfüllbar bzgl. Wissensbank

Vorteile

- weniger Unifikationen: Nur gewähltes Literal mit Köpfen der definiten Klauseln

SLD-Resolution

Selective Wähle nächstes Literal (In Prolog: zuletzt eingeführtes)

Linear Erste Elternklausel ist zuletzt erzeugte Zielklausel

Definite clause Zweite Elternklausel ist definite Klausel

Methode

- lege Atome (Goals) der Anfrage auf **Stack**
- unifiziere oberstes Goal mit allen Regelköpfen
- wenn erfolgreich: ersetze Goal durch Regelrumpf (evtl. mehrere Möglichkeiten)
- wenn Stack leer: Anfrageklausel ist unerfüllbar bzgl. Wissensbank

Vorteile

- weniger Unifikationen: Nur gewähltes Literal mit Köpfen der definiten Klauseln
- weniger erzeugte Klauseln: Nur eine Resolvente pro Anfrage-Klausel

Beispiel 5.40

Wissensbank

```
human(vincent).  
male(vincent).
```

```
human(mia).  
female(mia).
```

```
woman(Y) :- human(Y), female(Y).
```

Beispiel 5.40

Wissensbank

```
human(vincent).  
male(vincent).
```

```
human(mia).  
female(mia).
```

```
woman(Y) :- human(Y), female(Y).
```

woman(X)

Beispiel 5.40

Wissensbank

```
human(vincent).  
male(vincent).
```

```
human(mia).  
female(mia).
```

```
woman(Y) :- human(Y), female(Y).
```

woman(X)

| Y = X

human(X)
female(X)

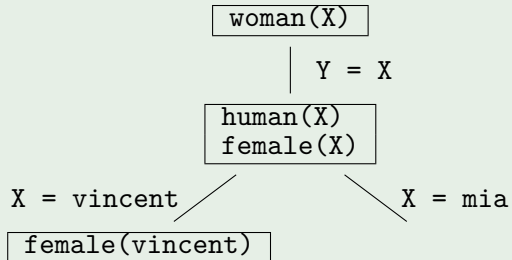
Beispiel 5.40

Wissensbank

```
human(vincent).  
male(vincent).
```

```
human(mia).  
female(mia).
```

```
woman(Y) :- human(Y), female(Y).
```



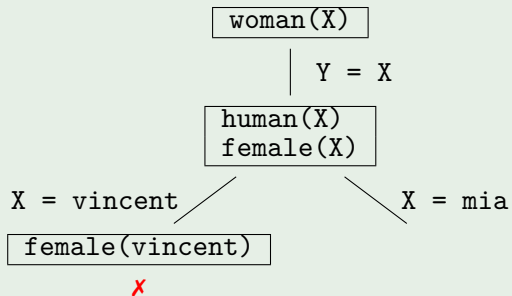
Beispiel 5.40

Wissensbank

```
human(vincent).  
male(vincent).
```

```
human(mia).  
female(mia).
```

```
woman(Y) :- human(Y), female(Y).
```



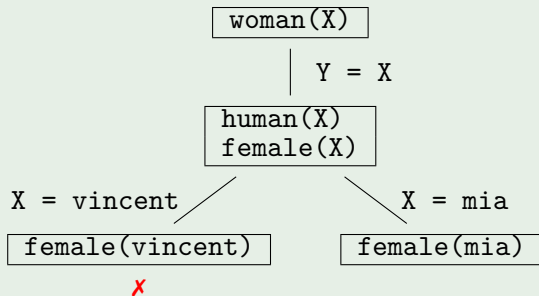
Beispiel 5.40

Wissensbank

```
human(vincent).  
male(vincent).
```

```
human(mia).  
female(mia).
```

```
woman(Y) :- human(Y), female(Y).
```



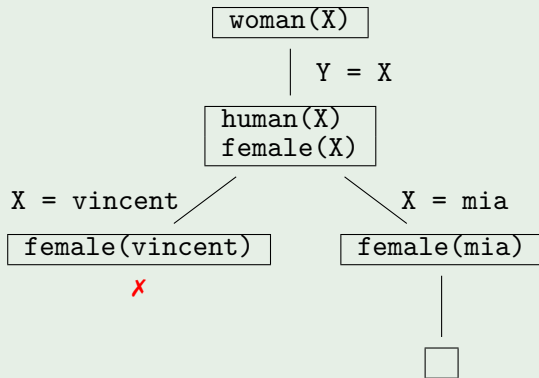
Beispiel 5.40

Wissensbank

```
human(vincent).  
male(vincent).
```

```
human(mia).  
female(mia).
```

```
woman(Y) :- human(Y), female(Y).
```



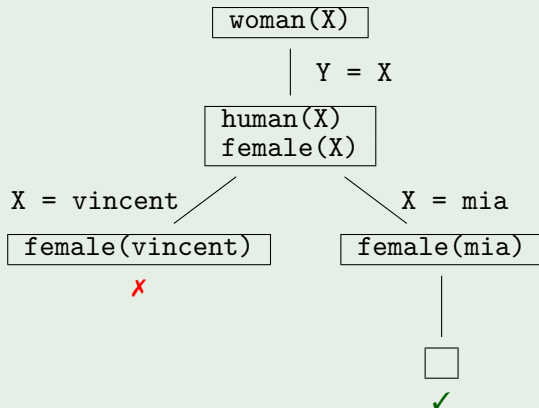
Beispiel 5.40

Wissensbank

```
human(vincent).  
male(vincent).
```

```
human(mia).  
female(mia).
```

```
woman(Y) :- human(Y), female(Y).
```



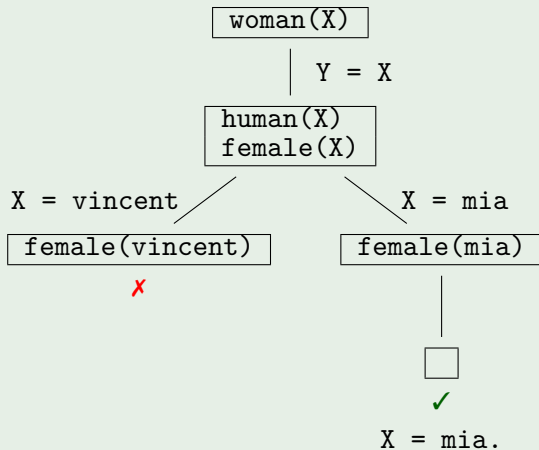
Beispiel 5.40

Wissensbank

```
human(vincent).  
male(vincent).
```

```
human(mia).  
female(mia).
```

```
woman(Y) :- human(Y), female(Y).
```



Eingabe: Anfrage-Klausel Q , Wissensbank K

Ausgabe: Variablenbindung, falls $K \models Q$ gilt, sonst `false`

Eingabe: Anfrage-Klausel Q , Wissensbank K

Ausgabe: Variablenbindung, falls $K \models Q$ gilt, sonst **false**

1: initialisiere Stack S mit Goals aus Q

Eingabe: Anfrage-Klausel Q , Wissensbank K

Ausgabe: Variablenbindung, falls $K \models Q$ gilt, sonst **false**

- 1: initialisiere Stack S mit Goals aus Q
- 2: **while** S ist nicht leer **do**
- 3: $G := \text{pop}(S)$; $i := 0$

Eingabe: Anfrage-Klausel Q , Wissensbank K

Ausgabe: Variablenbindung, falls $K \models Q$ gilt, sonst **false**

- 1: initialisiere Stack S mit Goals aus Q
- 2: **while** S ist nicht leer **do**
- 3: $G := \text{pop}(S)$; $i := 0$
- 4: **for all** Klausel C in K **do**
- 5: $\sigma_i := \text{unify}(G, \text{kopf}(C))$

Eingabe: Anfrage-Klausel Q , Wissensbank K

Ausgabe: Variablenbindung, falls $K \models Q$ gilt, sonst **false**

- 1: initialisiere Stack S mit Goals aus Q
- 2: **while** S ist nicht leer **do**
- 3: $G := \text{pop}(S)$; $i := 0$
- 4: **for all** Klausel C in K **do**
- 5: $\sigma_i := \text{unify}(G, \text{kopf}(C))$
- 6: **if** σ_i existiert **then**
- 7: $C_i := C$; $i := i + 1$

Eingabe: Anfrage-Klausel Q , Wissensbank K

Ausgabe: Variablenbindung, falls $K \models Q$ gilt, sonst **false**

- 1: initialisiere Stack S mit Goals aus Q
- 2: **while** S ist nicht leer **do**
- 3: $G := \text{pop}(S)$; $i := 0$
- 4: **for all** Klausel C in K **do**
- 5: $\sigma_i := \text{unify}(G, \text{kopf}(C))$
- 6: **if** σ_i existiert **then**
- 7: $C_i := C$; $i := i + 1$
- 8: **if** $i \geq 1$ **then**

Eingabe: Anfrage-Klausel Q , Wissensbank K

Ausgabe: Variablenbindung, falls $K \models Q$ gilt, sonst **false**

- 1: initialisiere Stack S mit Goals aus Q
- 2: **while** S ist nicht leer **do**
- 3: $G := \text{pop}(S)$; $i := 0$
- 4: **for all** Klausel C in K **do**
- 5: $\sigma_i := \text{unify}(G, \text{kopf}(C))$
- 6: **if** σ_i existiert **then**
- 7: $C_i := C$; $i := i + 1$
- 8: **if** $i \geq 1$ **then**
- 9: **if** $i \geq 2$ **then**

Eingabe: Anfrage-Klausel Q , Wissensbank K

Ausgabe: Variablenbindung, falls $K \models Q$ gilt, sonst **false**

- 1: initialisiere Stack S mit Goals aus Q
- 2: **while** S ist nicht leer **do**
- 3: $G := \text{pop}(S)$; $i := 0$
- 4: **for all** Klausel C in K **do**
- 5: $\sigma_i := \text{unify}(G, \text{kopf}(C))$
- 6: **if** σ_i existiert **then**
- 7: $C_i := C$; $i := i + 1$
- 8: **if** $i \geq 1$ **then**
- 9: **if** $i \geq 2$ **then**
- 10: lege Verzweigungspunkt an: Speichere C_i und σ_i für alle $i \geq 2$

Eingabe: Anfrage-Klausel Q , Wissensbank K

Ausgabe: Variablenbindung, falls $K \models Q$ gilt, sonst **false**

```
1: initialisiere Stack  $S$  mit Goals aus  $Q$ 
2: while  $S$  ist nicht leer do
3:    $G := \text{pop}(S)$ ;  $i := 0$ 
4:   for all Klausel  $C$  in  $K$  do
5:      $\sigma_i := \text{unify}(G, \text{kopf}(C))$ 
6:     if  $\sigma_i$  existiert then
7:        $C_i := C$ ;  $i := i + 1$ 
8:   if  $i \geq 1$  then
9:     if  $i \geq 2$  then
10:       lege Verzweigungspunkt an: Speichere  $C_i$  und  $\sigma_i$  für alle  $i \geq 2$ 
11:       push( $S$ , rumpf( $C_1$ )))
```


Eingabe: Anfrage-Klausel Q , Wissensbank K

Ausgabe: Variablenbindung, falls $K \models Q$ gilt, sonst **false**

```
1: initialisiere Stack  $S$  mit Goals aus  $Q$ 
2: while  $S$  ist nicht leer do
3:    $G := \text{pop}(S)$ ;  $i := 0$ 
4:   for all Klausel  $C$  in  $K$  do
5:      $\sigma_i := \text{unify}(G, \text{kopf}(C))$ 
6:     if  $\sigma_i$  existiert then
7:        $C_i := C$ ;  $i := i + 1$ 
8:   if  $i \geq 1$  then
9:     if  $i \geq 2$  then
10:      lege Verzweigungspunkt an: Speichere  $C_i$  und  $\sigma_i$  für alle  $i \geq 2$ 
11:       $\text{push}(S, \text{rumpf}(C_1))$ 
12:      Wende  $\sigma_1$  auf alle Goals in  $S$  an
```

Eingabe: Anfrage-Klausel Q , Wissensbank K

Ausgabe: Variablenbindung, falls $K \models Q$ gilt, sonst **false**

```
1: initialisiere Stack  $S$  mit Goals aus  $Q$ 
2: while  $S$  ist nicht leer do
3:    $G := \text{pop}(S)$ ;  $i := 0$ 
4:   for all Klausel  $C$  in  $K$  do
5:      $\sigma_i := \text{unify}(G, \text{kopf}(C))$ 
6:     if  $\sigma_i$  existiert then
7:        $C_i := C$ ;  $i := i + 1$ 
8:   if  $i \geq 1$  then
9:     if  $i \geq 2$  then
10:      lege Verzweigungspunkt an: Speichere  $C_i$  und  $\sigma_i$  für alle  $i \geq 2$ 
11:       $\text{push}(S, \text{rumpf}(C_1))$ 
12:      Wende  $\sigma_1$  auf alle Goals in  $S$  an
13:   else
14:     if Verzweigungspunkt  $B$  vorhanden then
```

Eingabe: Anfrage-Klausel Q , Wissensbank K

Ausgabe: Variablenbindung, falls $K \models Q$ gilt, sonst **false**

```
1: initialisiere Stack  $S$  mit Goals aus  $Q$ 
2: while  $S$  ist nicht leer do
3:    $G := \text{pop}(S)$ ;  $i := 0$ 
4:   for all Klausel  $C$  in  $K$  do
5:      $\sigma_i := \text{unify}(G, \text{kopf}(C))$ 
6:     if  $\sigma_i$  existiert then
7:        $C_i := C$ ;  $i := i + 1$ 
8:   if  $i \geq 1$  then
9:     if  $i \geq 2$  then
10:      lege Verzweigungspunkt an: Speichere  $C_i$  und  $\sigma_i$  für alle  $i \geq 2$ 
11:       $\text{push}(S, \text{rumpf}(C_1))$ 
12:      Wende  $\sigma_1$  auf alle Goals in  $S$  an
13:   else
14:     if Verzweigungspunkt  $B$  vorhanden then
15:       Backtrack zu  $B$ ; Weiter mit nächstem  $C_i$ 
```

Eingabe: Anfrage-Klausel Q , Wissensbank K

Ausgabe: Variablenbindung, falls $K \models Q$ gilt, sonst **false**

```
1: initialisiere Stack  $S$  mit Goals aus  $Q$ 
2: while  $S$  ist nicht leer do
3:    $G := \text{pop}(S)$ ;  $i := 0$ 
4:   for all Klausel  $C$  in  $K$  do
5:      $\sigma_i := \text{unify}(G, \text{kopf}(C))$ 
6:     if  $\sigma_i$  existiert then
7:        $C_i := C$ ;  $i := i + 1$ 
8:   if  $i \geq 1$  then
9:     if  $i \geq 2$  then
10:      lege Verzweigungspunkt an: Speichere  $C_i$  und  $\sigma_i$  für alle  $i \geq 2$ 
11:       $\text{push}(S, \text{rumpf}(C_1))$ 
12:      Wende  $\sigma_1$  auf alle Goals in  $S$  an
13:   else
14:     if Verzweigungspunkt  $B$  vorhanden then
15:       Backtrack zu  $B$ ; Weiter mit nächstem  $C_i$ 
16:   else
17:     return false
```

Eingabe: Anfrage-Klausel Q , Wissensbank K

Ausgabe: Variablenbindung, falls $K \models Q$ gilt, sonst **false**

```
1: initialisiere Stack  $S$  mit Goals aus  $Q$ 
2: while  $S$  ist nicht leer do
3:    $G := \text{pop}(S)$ ;  $i := 0$ 
4:   for all Klausel  $C$  in  $K$  do
5:      $\sigma_i := \text{unify}(G, \text{kopf}(C))$ 
6:     if  $\sigma_i$  existiert then
7:        $C_i := C$ ;  $i := i + 1$ 
8:   if  $i \geq 1$  then
9:     if  $i \geq 2$  then
10:      lege Verzweigungspunkt an: Speichere  $C_i$  und  $\sigma_i$  für alle  $i \geq 2$ 
11:       $\text{push}(S, \text{rumpf}(C_1))$ 
12:      Wende  $\sigma_1$  auf alle Goals in  $S$  an
13:   else
14:     if Verzweigungspunkt  $B$  vorhanden then
15:       Backtrack zu  $B$ ; Weiter mit nächstem  $C_i$ 
16:     else
17:       return false
18: return Variablenbindungen
```

Signatur: $(\{m^{(0)}, v^{(0)}\}, \{W^{(1)}, H^{(1)}, F^{(1)}\})$

Ziel-Klauseln (Stack) definite Klauseln (Wissensbank)

Signatur: $(\{m^{(0)}, v^{(0)}\}, \{W^{(1)}, H^{(1)}, F^{(1)}\})$

Ziel-Klauseln (Stack) definite Klauseln (Wissensbank)

$\{\neg W(x)\}$

Signatur: $(\{m^{(0)}, v^{(0)}\}, \{W^{(1)}, H^{(1)}, F^{(1)}\})$

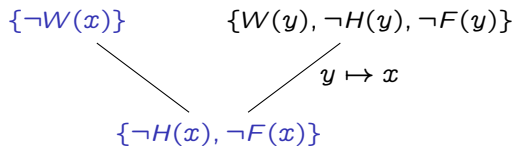
Ziel-Klauseln (Stack) definite Klauseln (Wissensbank)

$\{\neg W(x)\}$

$\{W(y), \neg H(y), \neg F(y)\}$

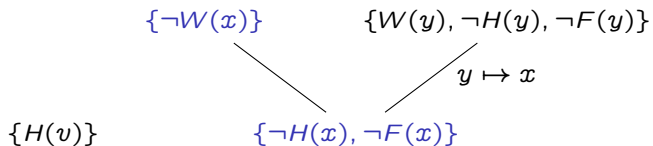
Signatur: $(\{m^{(0)}, v^{(0)}\}, \{W^{(1)}, H^{(1)}, F^{(1)}\})$

Ziel-Klauseln (Stack) definite Klauseln (Wissensbank)



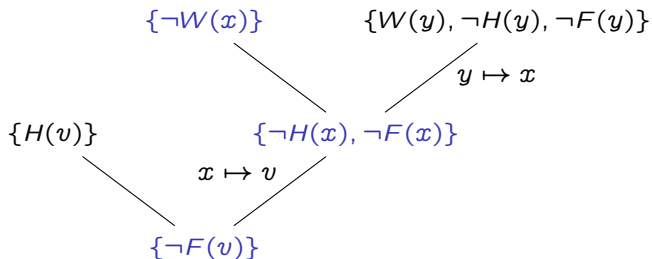
Signatur: $(\{m^{(0)}, v^{(0)}\}, \{W^{(1)}, H^{(1)}, F^{(1)}\})$

Ziel-Klauseln (Stack) definite Klauseln (Wissensbank)



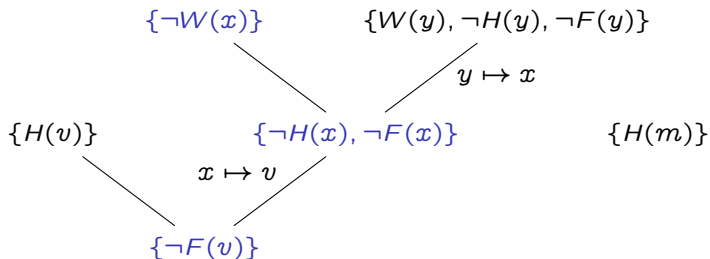
Signatur: $(\{m^{(0)}, v^{(0)}\}, \{W^{(1)}, H^{(1)}, F^{(1)}\})$

Ziel-Klauseln (Stack) definite Klauseln (Wissensbank)



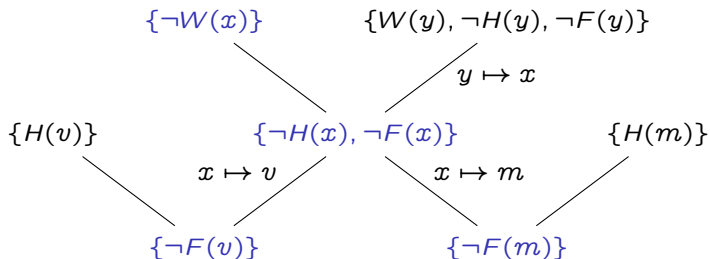
Signatur: $(\{m^{(0)}, v^{(0)}\}, \{W^{(1)}, H^{(1)}, F^{(1)}\})$

Ziel-Klauseln (Stack) definite Klauseln (Wissensbank)



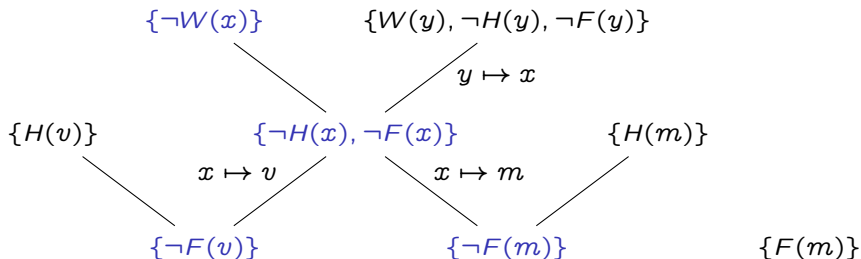
Signatur: $(\{m^{(0)}, v^{(0)}\}, \{W^{(1)}, H^{(1)}, F^{(1)}\})$

Ziel-Klauseln (Stack) definite Klauseln (Wissensbank)



Signatur: $(\{m^{(0)}, v^{(0)}\}, \{W^{(1)}, H^{(1)}, F^{(1)}\})$

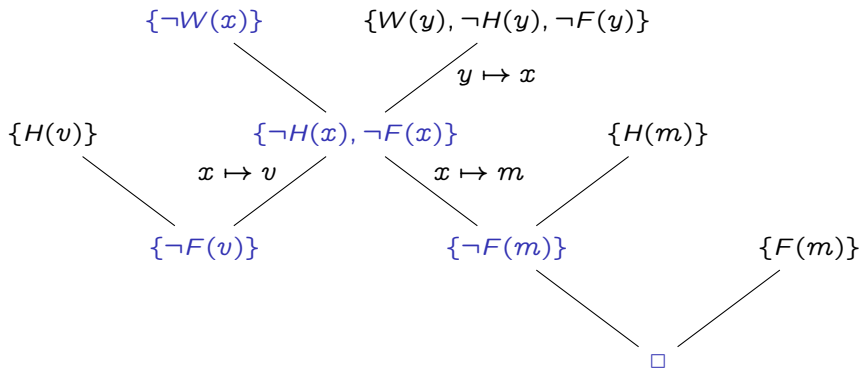
Ziel-Klauseln (Stack) definite Klauseln (Wissensbank)



Signatur: $(\{m^{(0)}, v^{(0)}\}, \{W^{(1)}, H^{(1)}, F^{(1)}\})$

Ziel-Klauseln (Stack)

definite Klauseln (Wissensbank)



Übung 5.41

Gegeben sei die Wissensbank aus Übung 5.29:

```
house_elf(dobby).  
witch(hermione).  
witch('McGonagall').  
wizard(ron).  
  
magic(X):- house_elf(X).  
magic(X):- wizard(X).  
magic(X):- witch(X).
```

Zeichnen Sie den Suchbaum für die Anfrage

```
?- magic(A).
```

Übung 5.42

Gegeben sei die folgende Wissensbank:

```
brother(albert,arthur).  
brother(arthur,albert).  
brother(beth,bob).  
  
child(albert,beth).  
child(beth,charlie).  
  
uncle(X,Y):- brother(Z,Y), child(Z,X).
```

Zeichnen Sie den Suchbaum für die Anfrage

```
?- uncle(A,B).
```


Horn-Formel $R_1 \wedge R_2 \wedge \dots \wedge R_n \rightarrow K$

Horn-Formel $R_1 \wedge R_2 \wedge \dots \wedge R_n \rightarrow K$

Horn-Klausel $\neg R_1 \vee \neg R_2 \vee \dots \vee \neg R_n \vee K$

Horn-Formel $R_1 \wedge R_2 \wedge \dots \wedge R_n \rightarrow K$

Horn-Klausel $\neg R_1 \vee \neg R_2 \vee \dots \vee \neg R_n \vee K$

definite Klausel genau ein positives Atom

Horn-Formel $R_1 \wedge R_2 \wedge \dots \wedge R_n \rightarrow K$

Horn-Klausel $\neg R_1 \vee \neg R_2 \vee \dots \vee \neg R_n \vee K$

definite Klausel genau ein positives Atom

Regel mindestens ein negatives Atom

Horn-Formel $R_1 \wedge R_2 \wedge \dots \wedge R_n \rightarrow K$

Horn-Klausel $\neg R_1 \vee \neg R_2 \vee \dots \vee \neg R_n \vee K$

definite Klausel genau ein positives Atom

Regel mindestens ein negatives Atom

Fakt kein negatives Atom

Horn-Formel $R_1 \wedge R_2 \wedge \dots \wedge R_n \rightarrow K$

Horn-Klausel $\neg R_1 \vee \neg R_2 \vee \dots \vee \neg R_n \vee K$

definite Klausel genau ein positives Atom

Regel mindestens ein negatives Atom

Fakt kein negatives Atom

Zielklausel kein positives Atom

Horn-Formel $R_1 \wedge R_2 \wedge \dots \wedge R_n \rightarrow K$

Horn-Klausel $\neg R_1 \vee \neg R_2 \vee \dots \vee \neg R_n \vee K$

definite Klausel genau ein positives Atom

Regel mindestens ein negatives Atom

Fakt kein negatives Atom

Zielklausel kein positives Atom

Anfrage mindestens ein negatives Atom

Horn-Formel $R_1 \wedge R_2 \wedge \dots \wedge R_n \rightarrow K$

Horn-Klausel $\neg R_1 \vee \neg R_2 \vee \dots \vee \neg R_n \vee K$

definite Klausel genau ein positives Atom

Regel mindestens ein negatives Atom

Fakt kein negatives Atom

Zielklausel kein positives Atom

Anfrage mindestens ein negatives Atom

leere Klausel kein negatives Atom

Horn-Formel $R_1 \wedge R_2 \wedge \dots \wedge R_n \rightarrow K$

Horn-Klausel $\neg R_1 \vee \neg R_2 \vee \dots \vee \neg R_n \vee K$

definite Klausel genau ein positives Atom

Regel mindestens ein negatives Atom

Fakt kein negatives Atom

Zielklausel kein positives Atom

Anfrage mindestens ein negatives Atom

leere Klausel kein negatives Atom

Zusammenfassung: Horn-Resolution

Horn-Formel $R_1 \wedge R_2 \wedge \dots \wedge R_n \rightarrow K$

Horn-Klausel $\neg R_1 \vee \neg R_2 \vee \dots \vee \neg R_n \vee K$

definite Klausel genau ein positives Atom

Regel mindestens ein negatives Atom

Fakt kein negatives Atom

Zielklausel kein positives Atom

Anfrage mindestens ein negatives Atom

leere Klausel kein negatives Atom

Horn-Resolution ■ beginnt mit Anfrage

Horn-Formel $R_1 \wedge R_2 \wedge \dots \wedge R_n \rightarrow K$

Horn-Klausel $\neg R_1 \vee \neg R_2 \vee \dots \vee \neg R_n \vee K$

definite Klausel genau ein positives Atom

Regel mindestens ein negatives Atom

Fakt kein negatives Atom

Zielklausel kein positives Atom

Anfrage mindestens ein negatives Atom

leere Klausel kein negatives Atom

Horn-Resolution

- beginnt mit Anfrage
- unifiziert erstes Goal mit Regel-**Köpfen**

Horn-Formel $R_1 \wedge R_2 \wedge \dots \wedge R_n \rightarrow K$

Horn-Klausel $\neg R_1 \vee \neg R_2 \vee \dots \vee \neg R_n \vee K$

definite Klausel genau ein positives Atom

Regel mindestens ein negatives Atom

Fakt kein negatives Atom

Zielklausel kein positives Atom

Anfrage mindestens ein negatives Atom

leere Klausel kein negatives Atom

Horn-Resolution

- beginnt mit Anfrage
- unifiziert erstes Goal mit Regel-**Köpfen**
- ersetzt erstes Goal durch Regel-**Rumpf**

Horn-Formel $R_1 \wedge R_2 \wedge \dots \wedge R_n \rightarrow K$

Horn-Klausel $\neg R_1 \vee \neg R_2 \vee \dots \vee \neg R_n \vee K$

definite Klausel genau ein positives Atom

Regel mindestens ein negatives Atom

Fakt kein negatives Atom

Zielklausel kein positives Atom

Anfrage mindestens ein negatives Atom

leere Klausel kein negatives Atom

Horn-Resolution

- beginnt mit Anfrage
- unifiziert erstes Goal mit Regel-**Köpfen**
- ersetzt erstes Goal durch Regel-**Rumpf**
- solange, bis alle Goals abgearbeitet sind (**true**)
oder keine Unifikation möglich ist (**false**)

Horn-Formel $R_1 \wedge R_2 \wedge \dots \wedge R_n \rightarrow K$

Horn-Klausel $\neg R_1 \vee \neg R_2 \vee \dots \vee \neg R_n \vee K$

definite Klausel genau ein positives Atom

Regel mindestens ein negatives Atom

Fakt kein negatives Atom

Zielklausel kein positives Atom

Anfrage mindestens ein negatives Atom

leere Klausel kein negatives Atom

Horn-Resolution

- beginnt mit Anfrage
- unifiziert erstes Goal mit Regel-**Köpfen**
- ersetzt erstes Goal durch Regel-**Rumpf**
- solange, bis alle Goals abgearbeitet sind (**true**)
oder keine Unifikation möglich ist (**false**)

Horn-Formel $R_1 \wedge R_2 \wedge \dots \wedge R_n \rightarrow K$

Horn-Klausel $\neg R_1 \vee \neg R_2 \vee \dots \vee \neg R_n \vee K$

definite Klausel genau ein positives Atom

Regel mindestens ein negatives Atom

Fakt kein negatives Atom

Zielklausel kein positives Atom

Anfrage mindestens ein negatives Atom

leere Klausel kein negatives Atom

Horn-Resolution

- beginnt mit Anfrage
- unifiziert erstes Goal mit Regel-**Köpfen**
- ersetzt erstes Goal durch Regel-**Rumpf**
- solange, bis alle Goals abgearbeitet sind (**true**)
oder keine Unifikation möglich ist (**false**)

Suchbaum ■ Stack von Goals

Horn-Formel $R_1 \wedge R_2 \wedge \dots \wedge R_n \rightarrow K$

Horn-Klausel $\neg R_1 \vee \neg R_2 \vee \dots \vee \neg R_n \vee K$

definite Klausel genau ein positives Atom

Regel mindestens ein negatives Atom

Fakt kein negatives Atom

Zielklausel kein positives Atom

Anfrage mindestens ein negatives Atom

leere Klausel kein negatives Atom

Horn-Resolution

- beginnt mit Anfrage
- unifiziert erstes Goal mit Regel-**Köpfen**
- ersetzt erstes Goal durch Regel-**Rumpf**
- solange, bis alle Goals abgearbeitet sind (**true**)
oder keine Unifikation möglich ist (**false**)

Suchbaum

- Stack von Goals
- Verzweigungspunkte bei mehreren Unifikationen

Horn-Formel $R_1 \wedge R_2 \wedge \dots \wedge R_n \rightarrow K$

Horn-Klausel $\neg R_1 \vee \neg R_2 \vee \dots \vee \neg R_n \vee K$

definite Klausel genau ein positives Atom

Regel mindestens ein negatives Atom

Fakt kein negatives Atom

Zielklausel kein positives Atom

Anfrage mindestens ein negatives Atom

leere Klausel kein negatives Atom

Horn-Resolution

- beginnt mit Anfrage
- unifiziert erstes Goal mit Regel-**Köpfen**
- ersetzt erstes Goal durch Regel-**Rumpf**
- solange, bis alle Goals abgearbeitet sind (**true**)
oder keine Unifikation möglich ist (**false**)

Suchbaum

- Stack von Goals
- Verzweigungspunkte bei mehreren Unifikationen
- Backtracking

- 1. Einführung
- 2. Mengen
- 3. Aussagenlogik
- 4. Prädikatenlogik
- 5. Prolog**
 - 5.1 Terme
 - 5.2 Fakten, Regeln und Anfragen
 - 5.3 Unifikation, Termgleichheit und Arithmetik
 - 5.4 Horn-Resolution und Suchbäume
 - 5.5 Rekursion**
 - 5.6 Listen
 - 5.7 Der Cut

Beispiel 5.43 (Fakultät iterativ)

```
int ifac (int n)
{
    int fac = 1;
    for (int i = 1; i <= n; i++)
        fac = fac * i;
    return fac;
}
```

Beispiel 5.43 (Fakultät iterativ)

```
int ifac (int n)
{
    int fac = 1;
    for (int i = 1; i <= n; i++)
        fac = fac * i;
    return fac;
}
```

Beispiel 5.44 (Fakultät rekursiv)

```
int rfac (int n)
{
    if (n == 0)
        return 1;
    else
        return n * rfac (n-1);
}
```

Beispiel 5.43 (Fakultät iterativ)

```
int ifac (int n)
{
    int fac = 1;
    for (int i = 1; i <= n; i++)
        fac = fac * i;
    return fac;
}
```

Beispiel 5.44 (Fakultät rekursiv)

```
int rfac (int n)
{
    if (n == 0)
        return 1;
    else
        return n * rfac (n-1);
}
```

To iterate is human, to recurse divine.

L Peter Deutsch

Definition 5.45

Ein Prolog-Prädikat p ist **rekursiv**, wenn in einer Regel, die p als Kopf hat, im Rumpf ebenfalls p vorkommt.

Definition 5.45

Ein Prolog-Prädikat p ist **rekursiv**, wenn in einer Regel, die p als Kopf hat, im Rumpf ebenfalls p vorkommt.

Beispiel 5.46

Wissensbank

```
child(anna,betty).  
child(betty,carol).  
child(carol,donna).
```

```
descend(X,Y):- child(X,Y).  
descend(X,Y):- child(X,Z), descend(Z,Y).
```

Definition 5.45

Ein Prolog-Prädikat p ist **rekursiv**, wenn in einer Regel, die p als Kopf hat, im Rumpf ebenfalls p vorkommt.

Beispiel 5.46

Wissensbank

```
child(anna,betty).  
child(betty,carol).  
child(carol,donna).  
  
descend(X,Y):- child(X,Y).  
descend(X,Y):- child(X,Z), descend(Z,Y).
```

Interpreter

```
?- descend(anna,betty).
```


Definition 5.45

Ein Prolog-Prädikat p ist **rekursiv**, wenn in einer Regel, die p als Kopf hat, im Rumpf ebenfalls p vorkommt.

Beispiel 5.46

Wissensbank

```
child(anna,betty).  
child(betty,carol).  
child(carol,donna).  
  
descend(X,Y):- child(X,Y).  
descend(X,Y):- child(X,Z), descend(Z,Y).
```

Interpreter

```
?- descend(anna,betty).  
true.
```

Definition 5.45

Ein Prolog-Prädikat p ist **rekursiv**, wenn in einer Regel, die p als Kopf hat, im Rumpf ebenfalls p vorkommt.

Beispiel 5.46

Wissensbank

```
child(anna,betty).  
child(betty,carol).  
child(carol,donna).  
  
descend(X,Y):- child(X,Y).  
descend(X,Y):- child(X,Z), descend(Z,Y).
```

Interpreter

```
?- descend(anna,betty).  
true.  
?- descend(anna,carol).
```

Definition 5.45

Ein Prolog-Prädikat p ist **rekursiv**, wenn in einer Regel, die p als Kopf hat, im Rumpf ebenfalls p vorkommt.

Beispiel 5.46

Wissensbank

```
child(anna,betty).  
child(betty,carol).  
child(carol,donna).  
  
descend(X,Y):- child(X,Y).  
descend(X,Y):- child(X,Z), descend(Z,Y).
```

Interpreter

```
?- descend(anna,betty).  
true.  
?- descend(anna,carol).  
true.
```

Definition 5.45

Ein Prolog-Prädikat p ist **rekursiv**, wenn in einer Regel, die p als Kopf hat, im Rumpf ebenfalls p vorkommt.

Beispiel 5.46

Wissensbank

```
child(anna,betty).  
child(betty,carol).  
child(carol,donna).  
  
descend(X,Y):- child(X,Y).  
descend(X,Y):- child(X,Z), descend(Z,Y).
```

Interpreter

```
?- descend(anna,betty).  
true.  
?- descend(anna,carol).  
true.  
?- descend(anna,donna).
```

Definition 5.45

Ein Prolog-Prädikat p ist **rekursiv**, wenn in einer Regel, die p als Kopf hat, im Rumpf ebenfalls p vorkommt.

Beispiel 5.46

Wissensbank

```
child(anna,betty).  
child(betty,carol).  
child(carol,donna).  
  
descend(X,Y):- child(X,Y).  
descend(X,Y):- child(X,Z), descend(Z,Y).
```

Interpreter

```
?- descend(anna,betty).  
true.  
?- descend(anna,carol).  
true.  
?- descend(anna,donna).  
true.
```

descend(anna,donna)

Wissensbank

```
child(anna,betty).  
child(betty,carol).  
child(carol,donna).  
  
% Basisklausel  
descend(X,Y):- child(X,Y).  
% Rekursive Klausel  
descend(X,Y):- child(X,Z),  
                descend(Z,Y).
```

Suchbaum für descend(anna,donna).

Wissensbank

```
child(anna,betty).  
child(betty,carol).  
child(carol,donna).
```

```
% Basisklausel
```

```
descend(X,Y):- child(X,Y).
```

```
% Rekursive Klausel
```

```
descend(X,Y):- child(X,Z),  
                descend(Z,Y).
```

child(anna,donna)

X1 = anna
Y1 = donna

descend(anna,donna)

X1 = anna
Y1 = donna

Suchbaum für descend(anna,donna).

Wissensbank

```
child(anna,betty).  
child(betty,carol).  
child(carol,donna).
```

```
% Basisklausel  
descend(X,Y):- child(X,Y).  
% Rekursive Klausel  
descend(X,Y):- child(X,Z),  
    descend(Z,Y).
```

child(anna,donna)

X1 = anna
Y1 = donna

descend(anna,donna)

X1 = anna
Y1 = donna

Variablenumbenennung

Bei jeder Unifikation werden die Variablen aus der Wissensbank umbenannt.

Suchbaum für descend(anna,donna).

Wissensbank

```
child(anna,betty).  
child(betty,carol).  
child(carol,donna).
```

```
% Basisklausel  
descend(X,Y):- child(X,Y).  
% Rekursive Klausel  
descend(X,Y):- child(X,Z),  
    descend(Z,Y).
```

child(anna,donna)

x

descend(anna,donna)

X1 = anna
Y1 = donna

X1 = anna
Y1 = donna

Variablenumbenennung

Bei jeder Unifikation werden die Variablen aus der Wissensbank umbenannt.

Suchbaum für descend(anna,donna).

Wissensbank

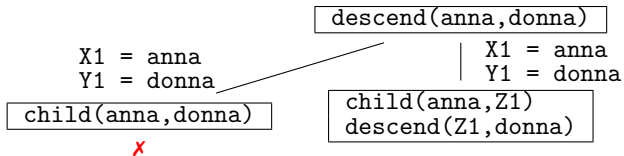
```
child(anna,betty).  
child(betty,carol).  
child(carol,donna).
```

```
% Basisklausel
```

```
descend(X,Y):- child(X,Y).
```

```
% Rekursive Klausel
```

```
descend(X,Y):- child(X,Z),  
                descend(Z,Y).
```



Variablenumbenennung

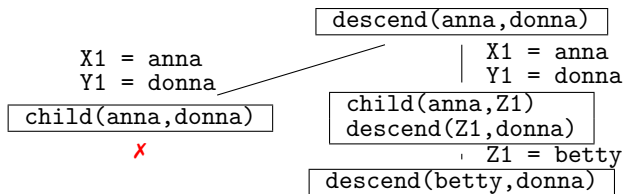
Bei jeder Unifikation werden die Variablen aus der Wissensbank umbenannt.

Suchbaum für descend(anna,donna).

Wissensbank

```
child(anna,betty).  
child(betty,carol).  
child(carol,donna).
```

```
% Basisklausel  
descend(X,Y):- child(X,Y).  
% Rekursive Klausel  
descend(X,Y):- child(X,Z),  
                descend(Z,Y).
```



Variablenumbenennung

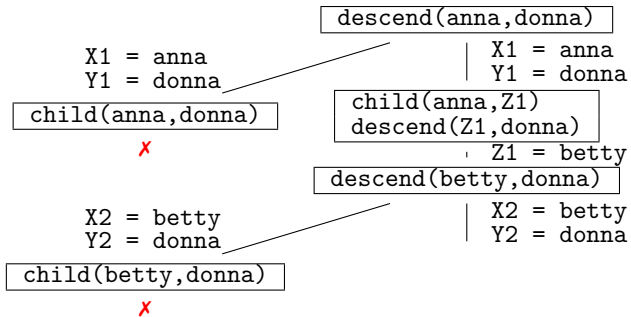
Bei jeder Unifikation werden die Variablen aus der Wissensbank umbenannt.

Suchbaum für descend(anna,donna).

Wissensbank

```
child(anna,betty).  
child(betty,carol).  
child(carol,donna).
```

```
% Basisklausel  
descend(X,Y):- child(X,Y).  
% Rekursive Klausel  
descend(X,Y):- child(X,Z),  
                descend(Z,Y).
```



Variablenumbenennung

Bei jeder Unifikation werden die Variablen aus der Wissensbank umbenannt.

Suchbaum für descend(anna,donna).

Wissensbank

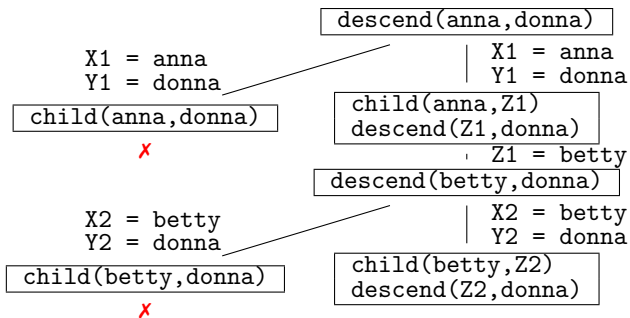
```
child(anna,betty).  
child(betty,carol).  
child(carol,donna).
```

```
% Basisklausel
```

```
descend(X,Y):- child(X,Y).
```

```
% Rekursive Klausel
```

```
descend(X,Y):- child(X,Z),  
                descend(Z,Y).
```



Variablenumbenennung

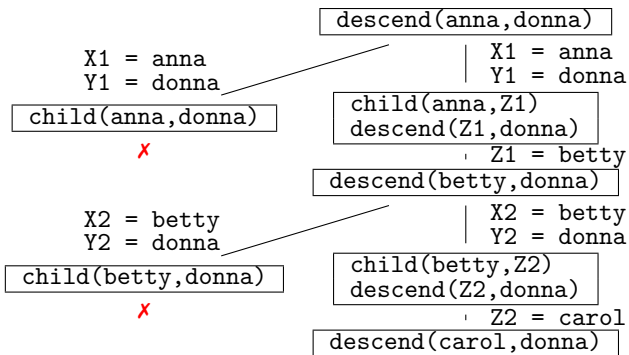
Bei jeder Unifikation werden die Variablen aus der Wissensbank umbenannt.

Suchbaum für descend(anna,donna).

Wissensbank

```
child(anna,betty).  
child(betty,carol).  
child(carol,donna).
```

```
% Basisklausel  
descend(X,Y):- child(X,Y).  
% Rekursive Klausel  
descend(X,Y):- child(X,Z),  
                descend(Z,Y).
```



Variablenumbenennung

Bei jeder Unifikation werden die Variablen aus der Wissensbank umbenannt.

Suchbaum für descend(anna,donna).

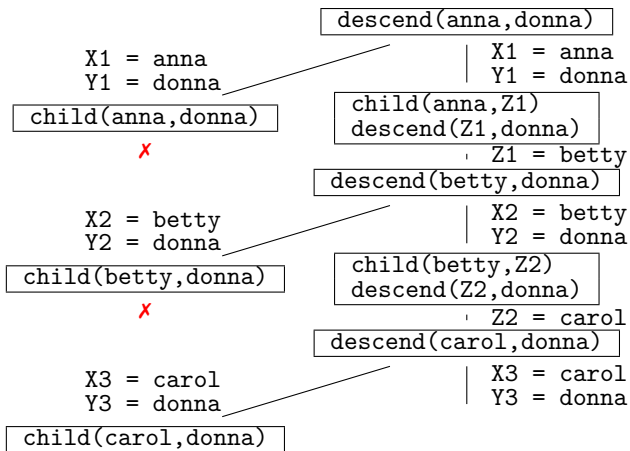
Wissensbank

```
child(anna,betty).  
child(betty,carol).  
child(carol,donna).
```

```
% Basisklausel  
descend(X,Y):- child(X,Y).  
% Rekursive Klausel  
descend(X,Y):- child(X,Z),  
                descend(Z,Y).
```

Variablenumbenennung

Bei jeder Unifikation werden die Variablen aus der Wissensbank umbenannt.



Suchbaum für descend(anna,donna).

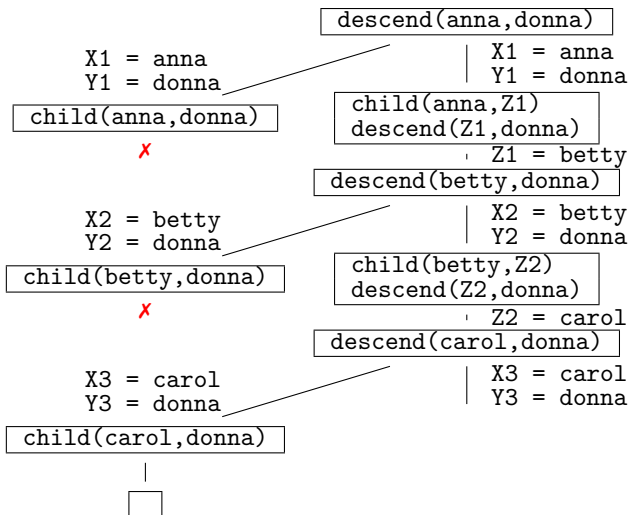
Wissensbank

```
child(anna,betty).  
child(betty,carol).  
child(carol,donna).
```

```
% Basisklausel  
descend(X,Y):- child(X,Y).  
% Rekursive Klausel  
descend(X,Y):- child(X,Z),  
                 descend(Z,Y).
```

Variablenumbenennung

Bei jeder Unifikation werden die Variablen aus der Wissensbank umbenannt.



Suchbaum für descend(anna,donna).

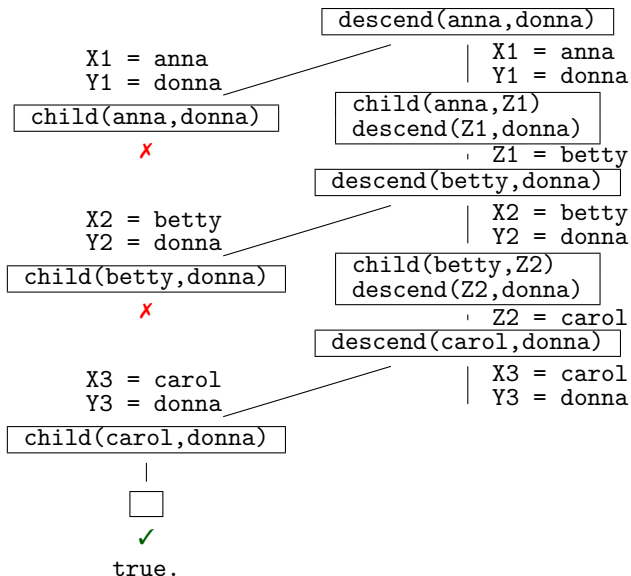
Wissensbank

```
child(anna,betty).  
child(betty,carol).  
child(carol,donna).
```

```
% Basisklausel  
descend(X,Y):- child(X,Y).  
% Rekursive Klausel  
descend(X,Y):- child(X,Z),  
                 descend(Z,Y).
```

Variablenumbenennung

Bei jeder Unifikation werden die Variablen aus der Wissensbank umbenannt.



Suchbaum für descend(anna,donna).

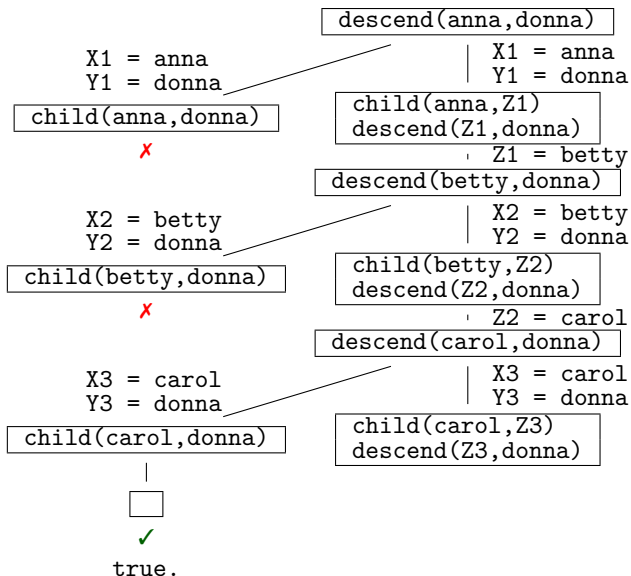
Wissensbank

```
child(anna,betty).  
child(betty,carol).  
child(carol,donna).
```

```
% Basisklausel  
descend(X,Y):- child(X,Y).  
% Rekursive Klausel  
descend(X,Y):- child(X,Z),  
                 descend(Z,Y).
```

Variablenumbenennung

Bei jeder Unifikation werden die Variablen aus der Wissensbank umbenannt.



Suchbaum für descend(anna,donna).

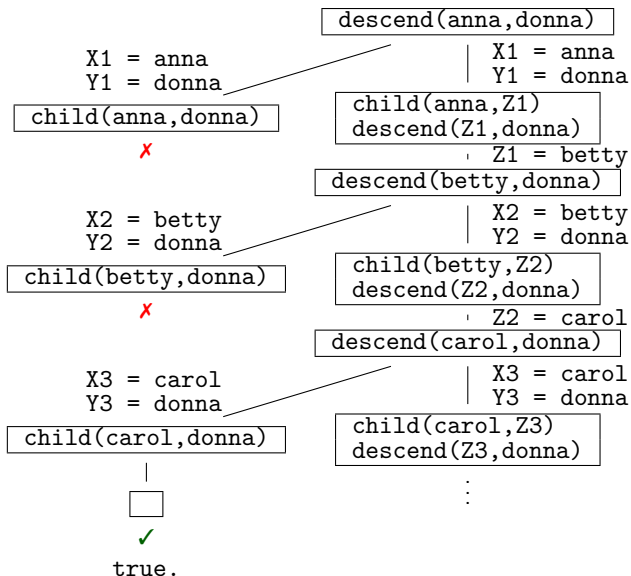
Wissensbank

```
child(anna,betty).  
child(betty,carol).  
child(carol,donna).
```

```
% Basisklausel  
descend(X,Y):- child(X,Y).  
% Rekursive Klausel  
descend(X,Y):- child(X,Z),  
                descend(Z,Y).
```

Variablenumbenennung

Bei jeder Unifikation werden die Variablen aus der Wissensbank umbenannt.



Übung 5.47

Ändert sich etwas an der Funktion des Prädikats `descend/2`, wenn man die rekursive Klausel

```
descend(X,Y) :- child(X,Z), descend(Z,Y).
```

durch

```
descend(X,Y) :- descend(X,Z), descend(Z,Y).
```

ersetzt?

Peano-Axiome:

- 1 Null ist eine natürliche Zahl.

Peano-Axiome:

- 1 Null ist eine natürliche Zahl.
- 2 Jede natürliche Zahl hat eine natürliche Zahl als Nachfolger.

Peano-Axiome:

- 1 Null ist eine natürliche Zahl.
- 2 Jede natürliche Zahl hat eine natürliche Zahl als Nachfolger.

Beispiel 5.48

Wissensbank

```
numeral(0).  
numeral(s(X)):- numeral(X).
```

Peano-Axiome:

- 1 Null ist eine natürliche Zahl.
- 2 Jede natürliche Zahl hat eine natürliche Zahl als Nachfolger.

Beispiel 5.48

Wissensbank

```
numeral(0).  
numeral(s(X)):- numeral(X).
```

Interpreter

```
?- numeral(s(s(s(0)))).
```


Peano-Axiome:

- 1 Null ist eine natürliche Zahl.
- 2 Jede natürliche Zahl hat eine natürliche Zahl als Nachfolger.

Beispiel 5.48

Wissensbank

```
numeral(0).  
numeral(s(X)):- numeral(X).
```

Interpreter

```
?- numeral(s(s(s(0)))).  
true.
```

Peano-Axiome:

- 1 Null ist eine natürliche Zahl.
- 2 Jede natürliche Zahl hat eine natürliche Zahl als Nachfolger.

Beispiel 5.48

Wissensbank

```
numeral(0).  
numeral(s(X)):- numeral(X).
```

Interpreter

```
?- numeral(s(s(s(0)))).  
true.  
?- numeral(X).
```

Peano-Axiome:

- 1 Null ist eine natürliche Zahl.
- 2 Jede natürliche Zahl hat eine natürliche Zahl als Nachfolger.

Beispiel 5.48

Wissensbank

```
numeral(0).  
numeral(s(X)):- numeral(X).
```

Interpreter

```
?- numeral(s(s(s(0)))).  
true.  
?- numeral(X).  
X = 0
```

Peano-Axiome:

- 1 Null ist eine natürliche Zahl.
- 2 Jede natürliche Zahl hat eine natürliche Zahl als Nachfolger.

Beispiel 5.48

Wissensbank

```
numeral(0).  
numeral(s(X)):- numeral(X).
```

Interpreter

```
?- numeral(s(s(s(0)))).  
true.  
?- numeral(X).  
X = 0 ;
```

Peano-Axiome:

- 1 Null ist eine natürliche Zahl.
- 2 Jede natürliche Zahl hat eine natürliche Zahl als Nachfolger.

Beispiel 5.48

Wissensbank

```
numeral(0).  
numeral(s(X)):- numeral(X).
```

Interpreter

```
?- numeral(s(s(s(0)))).  
true.  
?- numeral(X).  
X = 0 ;  
X = s(0)
```

Peano-Axiome:

- 1 Null ist eine natürliche Zahl.
- 2 Jede natürliche Zahl hat eine natürliche Zahl als Nachfolger.

Beispiel 5.48

Wissensbank

```
numeral(0).  
numeral(s(X)):- numeral(X).
```

Interpreter

```
?- numeral(s(s(s(0)))).  
true.  
?- numeral(X).  
X = 0 ;  
X = s(0) ;
```

Peano-Axiome:

- 1 Null ist eine natürliche Zahl.
- 2 Jede natürliche Zahl hat eine natürliche Zahl als Nachfolger.

Beispiel 5.48

Wissensbank

```
numeral(0).  
numeral(s(X)):- numeral(X).
```

Interpreter

```
?- numeral(s(s(s(0)))).  
true.  
?- numeral(X).  
X = 0 ;  
X = s(0) ;  
X = s(s(0))  
...
```

Beispiel 5.49

Wissensbank
<pre>% Basisklausel add(0,X,X).</pre>

Beispiel 5.49

Wissensbank

```
% Basisklausel  
add(0,X,X).
```

Intuitiv:

- „Die Summe von 0 und X ist X.“

Beispiel 5.49

Wissensbank

% Basisklausel

add(0,X,X).

% Rekursive Klausel

add(s(X),Y,s(Z)) :- add(X,Y,Z).

Intuitiv:

- „Die Summe von 0 und X ist X.“

Beispiel 5.49

Wissensbank

% Basisklausel

add(0,X,X).

% Rekursive Klausel

add(s(X),Y,s(Z)) :- add(X,Y,Z).

Intuitiv:

- „Die Summe von 0 und X ist X.“
- „Die Summe vom **Nachfolger** von X und Y ist der **Nachfolger** der Summe von X und Y.“

Beispiel 5.49

Wissensbank

% Basisklausel

add(0,X,X).

% Rekursive Klausel

add(s(X),Y,s(Z)) :- add(X,Y,Z).

Intuitiv:

- „Die Summe von 0 und X ist X.“
- „Die Summe vom **Nachfolger** von X und Y ist der **Nachfolger** der Summe von X und Y.“

Interpreter

?- add(s(s(0)),s(s(s(0))),Sum).

Beispiel 5.49

Wissensbank

```
% Basisklausel  
add(0,X,X).  
% Rekursive Klausel  
add(s(X),Y,s(Z)) :- add(X,Y,Z).
```

Intuitiv:

- „Die Summe von 0 und X ist X.“
- „Die Summe vom **Nachfolger** von X und Y ist der **Nachfolger** der Summe von X und Y.“

Interpreter

```
?- add(s(s(0)),s(s(s(0))),Sum).  
Sum = s(s(s(s(s(0))))).
```

Übung 5.50

Zeichnen Sie den Suchbaum für die Anfrage

```
?- add(s(s(0)),s(s(s(0))),Sum).
```

Übung 5.51

Gegeben sei die logische Notation für natürliche Zahlen aus Beispiel 5.48.

Schreiben Sie ein Prädikat `less/2`, das genau dann gilt, wenn das erste Argument echt kleiner ist als das zweite, also:

```
?- less(0,s(s(0))).  
true.  
?- less(s(0),s(0)).  
false.
```

Testen Sie Ihr Programm auch mit Variablen als Parametern.

Prädikatenlogik ■ $\wedge$ und $\vee$ sind kommutativ

Prädikatenlogik

- $\wedge$ und $\vee$ sind kommutativ
- $P(x) \wedge B(x, y) \vee C(z)$ ist äquivalent zu $C(z) \vee B(x, y) \wedge P(x)$

Prädikatenlogik

- $\wedge$ und $\vee$ sind kommutativ
- $P(x) \wedge B(x, y) \vee C(z)$ ist äquivalent zu $C(z) \vee B(x, y) \wedge P(x)$

Prolog

- Goals auf dem Stack werden **von oben nach unten** verarbeitet

Prädikatenlogik

- $\wedge$ und $\vee$ sind kommutativ
- $P(x) \wedge B(x, y) \vee C(z)$ ist äquivalent zu $C(z) \vee B(x, y) \wedge P(x)$

Prolog

- Goals auf dem Stack werden **von oben nach unten** verarbeitet
- Wissensbank wird **von oben nach unten** durchsucht

Prädikatenlogik

- $\wedge$ und $\vee$ sind kommutativ
- $P(x) \wedge B(x, y) \vee C(z)$ ist äquivalent zu $C(z) \vee B(x, y) \wedge P(x)$

Prolog

- Goals auf dem Stack werden **von oben nach unten** verarbeitet
- Wissensbank wird **von oben nach unten** durchsucht
- $\text{child}(X,Y), \text{descend}(Y,Z)$ und $\text{descend}(Y,Z), \text{child}(X,Y)$ sind **logisch** äquivalent, aber **prozedural** unterschiedlich

Prädikatenlogik

- $\wedge$ und $\vee$ sind kommutativ
- $P(x) \wedge B(x, y) \vee C(z)$ ist äquivalent zu $C(z) \vee B(x, y) \wedge P(x)$

Prolog

- Goals auf dem Stack werden **von oben nach unten** verarbeitet
- Wissensbank wird **von oben nach unten** durchsucht
- $\text{child}(X,Y), \text{descend}(Y,Z)$ und $\text{descend}(Y,Z), \text{child}(X,Y)$ sind **logisch** äquivalent, aber **prozedural** unterschiedlich

Wünschenswert für rekursive Prädikate:

- Basisklausel vor rekursiver Klausel
- Rekursives Goal als letztes im Rumpf (**tail-rekursiv**)

Beispiel 5.52 (Korrektes Prädikat)

```
descend(X,Y):- child(X,Y).  
descend(X,Y):- child(X,Z),descend(Z,Y).
```

`descend(X,Y).` $\rightsquigarrow$ 6 Antworten:

Beispiel 5.52 (Korrektes Prädikat)

```
descend(X,Y):- child(X,Y).  
descend(X,Y):- child(X,Z),descend(Z,Y).
```

`descend(X,Y).` $\rightsquigarrow$ 6 Antworten:

1 **X = anna, Y = betty**

6 **X = betty, Y = donna**

Beispiel 5.52 (Korrektes Prädikat)

```
descend(X,Y):- child(X,Y).  
descend(X,Y):- child(X,Z),descend(Z,Y).
```

descend(X,Y). $\rightsquigarrow$ 6 Antworten:

1 X = anna, Y = betty

6 X = betty, Y = donna

Beispiel 5.53 (Klauseln vertauscht)

```
descend(X,Y):- child(X,Z),descend(Z,Y).  
descend(X,Y):- child(X,Y).
```

Beispiel 5.52 (Korrektes Prädikat)

```
descend(X,Y):- child(X,Y).  
descend(X,Y):- child(X,Z),descend(Z,Y).
```

descend(X,Y). $\rightsquigarrow$ 6 Antworten:

- 1 **X = anna, Y = betty**
- 6 **X = betty, Y = donna**

Beispiel 5.53 (Klauseln vertauscht)

```
descend(X,Y):- child(X,Z),descend(Z,Y).  
descend(X,Y):- child(X,Y).
```

descend(X,Y). $\rightsquigarrow$ 6 Antworten:

- 1 **X = anna, Y = donna**
- 6 **X = carol, Y = donna.**

Beispiel 5.52 (Korrektes Prädikat)

```
descend(X,Y):- child(X,Y).  
descend(X,Y):- child(X,Z),descend(Z,Y).
```

descend(X,Y). $\rightsquigarrow$ 6 Antworten:

- 1 **X = anna, Y = betty**
- 6 **X = betty, Y = donna**

Beispiel 5.53 (Klauseln vertauscht)

```
descend(X,Y):- child(X,Z),descend(Z,Y).  
descend(X,Y):- child(X,Y).
```

descend(X,Y). $\rightsquigarrow$ 6 Antworten:

- 1 **X = anna, Y = donna**
- 6 **X = carol, Y = donna.**

Beispiel 5.54 (Goals vertauscht)

```
descend(X,Y):- child(X,Y).  
descend(X,Y):- descend(Z,Y),child(X,Z).
```

Beispiel 5.52 (Korrektes Prädikat)

```
descend(X,Y):- child(X,Y).  
descend(X,Y):- child(X,Z),descend(Z,Y).
```

descend(X,Y). $\rightsquigarrow$ 6 Antworten:

- 1 X = anna, Y = betty**
- 6 X = betty, Y = donna**

Beispiel 5.53 (Klauseln vertauscht)

```
descend(X,Y):- child(X,Z),descend(Z,Y).  
descend(X,Y):- child(X,Y).
```

descend(X,Y). $\rightsquigarrow$ 6 Antworten:

- 1 X = anna, Y = donna**
- 6 X = carol, Y = donna.**

Beispiel 5.54 (Goals vertauscht)

```
descend(X,Y):- child(X,Y).  
descend(X,Y):- descend(Z,Y),child(X,Z).
```

descend(X,Y). $\rightsquigarrow$ 6 Antworten, dann Fehler:

- 1 X = anna, Y = betty**
- 6 X = anna, Y = donna**

ERROR: Out of local stack

Beispiel 5.52 (Korrektes Prädikat)

```
descend(X,Y):- child(X,Y).  
descend(X,Y):- child(X,Z),descend(Z,Y).
```

descend(X,Y). $\rightsquigarrow$ 6 Antworten:

- 1 X = anna, Y = betty**
- 6 X = betty, Y = donna**

Beispiel 5.54 (Goals vertauscht)

```
descend(X,Y):- child(X,Y).  
descend(X,Y):- descend(Z,Y),child(X,Z).
```

descend(X,Y). $\rightsquigarrow$ 6 Antworten, dann Fehler:

- 1 X = anna, Y = betty**
- 6 X = anna, Y = donna**

ERROR: Out of local stack

Beispiel 5.53 (Klauseln vertauscht)

```
descend(X,Y):- child(X,Z),descend(Z,Y).  
descend(X,Y):- child(X,Y).
```

descend(X,Y). $\rightsquigarrow$ 6 Antworten:

- 1 X = anna, Y = donna**
- 6 X = carol, Y = donna.**

Beispiel 5.55 (Beides vertauscht)

```
descend(X,Y):- descend(Z,Y),child(X,Z).  
descend(X,Y):- child(X,Z).
```

Beispiel 5.52 (Korrektes Prädikat)

```
descend(X,Y):- child(X,Y).  
descend(X,Y):- child(X,Z),descend(Z,Y).
```

descend(X,Y). $\rightsquigarrow$ 6 Antworten:

- 1 X = anna, Y = betty**
- 6 X = betty, Y = donna**

Beispiel 5.54 (Goals vertauscht)

```
descend(X,Y):- child(X,Y).  
descend(X,Y):- descend(Z,Y),child(X,Z).
```

descend(X,Y). $\rightsquigarrow$ 6 Antworten, dann Fehler:

- 1 X = anna, Y = betty**
 - 6 X = anna, Y = donna**
- ERROR: Out of local stack**

Beispiel 5.53 (Klauseln vertauscht)

```
descend(X,Y):- child(X,Z),descend(Z,Y).  
descend(X,Y):- child(X,Y).
```

descend(X,Y). $\rightsquigarrow$ 6 Antworten:

- 1 X = anna, Y = donna**
- 6 X = carol, Y = donna.**

Beispiel 5.55 (Beides vertauscht)

```
descend(X,Y):- descend(Z,Y),child(X,Z).  
descend(X,Y):- child(X,Z).
```

descend(X,Y). $\rightsquigarrow$ Fehler:
ERROR: Out of local stack

- Rekursive Regel: Kopf-Prädikat taucht im Rumpf auf

- Rekursive Regel: Kopf-Prädikat taucht im Rumpf auf
- notwendig für leistungsfähige Programme

- Rekursive Regel: Kopf-Prädikat taucht im Rumpf auf
- notwendig für leistungsfähige Programme
- für Suchbaum: Variablen umbenennen

- Rekursive Regel: Kopf-Prädikat taucht im Rumpf auf
- notwendig für leistungsfähige Programme
- für Suchbaum: Variablen umbenennen
- kann zu nicht terminierenden Programmen führen

- Rekursive Regel: Kopf-Prädikat taucht im Rumpf auf
- notwendig für leistungsfähige Programme
- für Suchbaum: Variablen umbenennen
- kann zu nicht terminierenden Programmen führen
- Abhilfe:

- Rekursive Regel: Kopf-Prädikat taucht im Rumpf auf
- notwendig für leistungsfähige Programme
- für Suchbaum: Variablen umbenennen
- kann zu nicht terminierenden Programmen führen
- Abhilfe:
 - in Wissensbank: zuerst nicht-rekursive Klausel

- Rekursive Regel: Kopf-Prädikat taucht im Rumpf auf
- notwendig für leistungsfähige Programme
- für Suchbaum: Variablen umbenennen
- kann zu nicht terminierenden Programmen führen
- Abhilfe:
 - in Wissensbank: zuerst nicht-rekursive Klausel
 - im Regel-Rumpf: zuerst nicht-rekursive Goals ([Tail-Rekursion](#))

1. Einführung

2. Mengen

3. Aussagenlogik

4. Prädikatenlogik

5. Prolog

5.1 Terme

5.2 Fakten, Regeln und Anfragen

5.3 Unifikation, Termgleichheit und Arithmetik

5.4 Horn-Resolution und Suchbäume

5.5 Rekursion

5.6 Listen

5.7 Der Cut

[Head | Tail]

[Head | Tail]

Head ■ erstes Element der Liste

[Head | Tail]

- Head
- erstes Element der Liste
 - beliebiger Prolog-Term (Variable, Atom, Zahl, komplexer Term)

[Head | Tail]

- Head**
- erstes Element der Liste
 - beliebiger Prolog-Term (Variable, Atom, Zahl, komplexer Term)
- Tail**
- Rest der Liste

[Head | Tail]

- Head**
- erstes Element der Liste
 - beliebiger Prolog-Term (Variable, Atom, Zahl, komplexer Term)
- Tail**
- Rest der Liste
 - alles bis auf das erste Element

[Head | Tail]

- Head**
- erstes Element der Liste
 - beliebiger Prolog-Term (Variable, Atom, Zahl, komplexer Term)
- Tail**
- Rest der Liste
 - alles bis auf das erste Element
 - ist immer eine Liste

[Head | Tail]

- Head
 - erstes Element der Liste
 - beliebiger Prolog-Term (Variable, Atom, Zahl, komplexer Term)
- Tail
 - Rest der Liste
 - alles bis auf das erste Element
 - ist immer eine Liste
- []
 - leere Liste

[Head | Tail]

- Head
 - erstes Element der Liste
 - beliebiger Prolog-Term (Variable, Atom, Zahl, komplexer Term)
- Tail
 - Rest der Liste
 - alles bis auf das erste Element
 - ist immer eine Liste
- []
 - leere Liste
 - hat weder Head noch Tail

[Head | Tail]

- Head
 - erstes Element der Liste
 - beliebiger Prolog-Term (Variable, Atom, Zahl, komplexer Term)
- Tail
 - Rest der Liste
 - alles bis auf das erste Element
 - ist immer eine Liste
- []
 - leere Liste
 - hat weder Head noch Tail
 - ähnlich Null-Pointer

[Head | Tail]

- Head**
 - erstes Element der Liste
 - beliebiger Prolog-Term (Variable, Atom, Zahl, komplexer Term)
- Tail**
 - Rest der Liste
 - alles bis auf das erste Element
 - ist immer eine Liste
- []**
 - leere Liste
 - hat weder Head noch Tail
 - ähnlich Null-Pointer

Beispiel 5.56 (Liste mit 3 Elementen)

```
[ vincent | [ mia | [ jules | [ ] ] ] ]
```

ist die Liste bestehend aus den Atomen `vincent`, `mia` und `jules`.

Aufzählung der Elemente

Aufzählung der Elemente

- [vincent, mia, jules]

Aufzählung der Elemente

- `[vincent, mia, jules]`
- oft besser lesbar als Head-Tail-Notation

Aufzählung der Elemente

- `[vincent, mia, jules]`
- oft besser lesbar als Head-Tail-Notation
- Notationen können gemischt werden: `[vincent, mia | [jules]]`

Aufzählung der Elemente

- `[vincent, mia, jules]`
- oft besser lesbar als Head-Tail-Notation
- Notationen können gemischt werden: `[vincent, mia | [jules]]`

Beispiel 5.57

```
?- [Head | Tail] = [vincent,mia,jules].
```

Aufzählung der Elemente

- `[vincent, mia, jules]`
- oft besser lesbar als Head-Tail-Notation
- Notationen können gemischt werden: `[vincent, mia | [jules]]`

Beispiel 5.57

```
?- [Head | Tail] = [vincent,mia,jules].  
Head = vincent, Tail = [mia,jules].
```

Aufzählung der Elemente

- `[vincent, mia, jules]`
- oft besser lesbar als Head-Tail-Notation
- Notationen können gemischt werden: `[vincent, mia | [jules]]`

Beispiel 5.57

```
?- [Head | Tail] = [vincent,mia,jules].  
Head = vincent, Tail = [mia,jules].  
?- [Head | Tail] = [mia].
```

Aufzählung der Elemente

- `[vincent, mia, jules]`
- oft besser lesbar als Head-Tail-Notation
- Notationen können gemischt werden: `[vincent, mia | [jules]]`

Beispiel 5.57

```
?- [Head | Tail] = [vincent,mia,jules].  
Head = vincent, Tail = [mia,jules].  
?- [Head | Tail] = [mia].  
Head = mia, Tail = [].
```

Aufzählung der Elemente

- `[vincent, mia, jules]`
- oft besser lesbar als Head-Tail-Notation
- Notationen können gemischt werden: `[vincent, mia | [jules]]`

Beispiel 5.57

```
?- [Head | Tail] = [vincent,mia,jules].  
Head = vincent, Tail = [mia,jules].  
?- [Head | Tail] = [mia].  
Head = mia, Tail = [].  
?- [Head | Tail] = [].
```

Aufzählung der Elemente

- `[vincent, mia, jules]`
- oft besser lesbar als Head-Tail-Notation
- Notationen können gemischt werden: `[vincent, mia | [jules]]`

Beispiel 5.57

```
?- [Head | Tail] = [vincent,mia,jules].  
Head = vincent, Tail = [mia,jules].  
?- [Head | Tail] = [mia].  
Head = mia, Tail = [].  
?- [Head | Tail] = [].  
false.
```


Aufzählung der Elemente

- `[vincent, mia, jules]`
- oft besser lesbar als Head-Tail-Notation
- Notationen können gemischt werden: `[vincent, mia | [jules]]`

Beispiel 5.57

```
?- [Head | Tail] = [vincent,mia,jules].  
Head = vincent, Tail = [mia,jules].  
?- [Head | Tail] = [mia].  
Head = mia, Tail = [].  
?- [Head | Tail] = [].  
false.  
?- [First, Second | Rest] = [vincent, mia, jules, marsellus].
```

Aufzählung der Elemente

- `[vincent, mia, jules]`
- oft besser lesbar als Head-Tail-Notation
- Notationen können gemischt werden: `[vincent, mia | [jules]]`

Beispiel 5.57

```
?- [Head | Tail] = [vincent,mia,jules].  
Head = vincent, Tail = [mia,jules].  
?- [Head | Tail] = [mia].  
Head = mia, Tail = [].  
?- [Head | Tail] = [].  
false.  
?- [First, Second | Rest] = [vincent, mia, jules, marsellus].  
First = vincent, Second = mia, Rest = [jules, marsellus].
```

Aufzählung der Elemente

- `[vincent, mia, jules]`
- oft besser lesbar als Head-Tail-Notation
- Notationen können gemischt werden: `[vincent, mia | [jules]]`

Beispiel 5.57

```
?- [Head | Tail] = [vincent,mia,jules].  
Head = vincent, Tail = [mia,jules].  
?- [Head | Tail] = [mia].  
Head = mia, Tail = [].  
?- [Head | Tail] = [].  
false.  
?- [First, Second | Rest] = [vincent, mia, jules, marsellus].  
First = vincent, Second = mia, Rest = [jules, marsellus].  
?- [First, Second] = [mia].
```

Aufzählung der Elemente

- `[vincent, mia, jules]`
- oft besser lesbar als Head-Tail-Notation
- Notationen können gemischt werden: `[vincent, mia | [jules]]`

Beispiel 5.57

```
?- [Head | Tail] = [vincent,mia,jules].  
Head = vincent, Tail = [mia,jules].  
?- [Head | Tail] = [mia].  
Head = mia, Tail = [].  
?- [Head | Tail] = [].  
false.  
?- [First, Second | Rest] = [vincent, mia, jules, marsellus].  
First = vincent, Second = mia, Rest = [jules, marsellus].  
?- [First, Second] = [mia].  
false.
```

Aufzählung der Elemente

- `[vincent, mia, jules]`
- oft besser lesbar als Head-Tail-Notation
- Notationen können gemischt werden: `[vincent, mia | [jules]]`

Beispiel 5.57

```
?- [Head | Tail] = [vincent,mia,jules].  
Head = vincent, Tail = [mia,jules].  
?- [Head | Tail] = [mia].  
Head = mia, Tail = [].  
?- [Head | Tail] = [].  
false.  
?- [First, Second | Rest] = [vincent, mia, jules, marsellus].  
First = vincent, Second = mia, Rest = [jules, marsellus].  
?- [First, Second] = [mia].  
false.  
?- [First, Second] = [vincent,mia,jules].
```

Aufzählung der Elemente

- `[vincent, mia, jules]`
- oft besser lesbar als Head-Tail-Notation
- Notationen können gemischt werden: `[vincent, mia | [jules]]`

Beispiel 5.57

```
?- [Head | Tail] = [vincent,mia,jules].  
Head = vincent, Tail = [mia,jules].  
?- [Head | Tail] = [mia].  
Head = mia, Tail = [].  
?- [Head | Tail] = [].  
false.  
?- [First, Second | Rest] = [vincent, mia, jules, marsellus].  
First = vincent, Second = mia, Rest = [jules, marsellus].  
?- [First, Second] = [mia].  
false.  
?- [First, Second] = [vincent,mia,jules].  
false.
```

[[] | []]

- erstes Element: leere Liste

[[] | []]

- erstes Element: leere Liste
- Rest: leere Liste

[[] | []]

- erstes Element: leere Liste

- Rest: leere Liste

~> Liste, die als einziges Element die leere Liste enthält

[[] | []]

- erstes Element: leere Liste
- Rest: leere Liste

~> Liste, die als einziges Element die leere Liste enthält

~> [[]]

[[] | []]

- erstes Element: leere Liste
- Rest: leere Liste

~> Liste, die als einziges Element die leere Liste enthält

~> [[]]

`[ [ ] | [ ] ]`

- erstes Element: leere Liste
- Rest: leere Liste

↪ Liste, die als einziges Element die leere Liste enthält

↪ `[ [ ] ]`

`[mia, [vincent, marsellus], [butch, [father(butch), mother(butch)]], [ ]]`

- erstes Element: mia

```
[ [ ] | [ ] ]
```

- erstes Element: leere Liste
- Rest: leere Liste

↪ Liste, die als einziges Element die leere Liste enthält

↪ [[]]

```
[mia, [vincent, marsellus], [butch, [father(butch), mother(butch)]], [ ]]
```

- erstes Element: mia
- zweites Element: Liste [vincent, marsellus]

```
[ [ ] | [ ] ]
```

- erstes Element: leere Liste
- Rest: leere Liste

~> Liste, die als einziges Element die leere Liste enthält

~> [[]]

```
[mia, [vincent, marsellus], [butch, [father(butch), mother(butch)]], [ ]]
```

- erstes Element: mia
- zweites Element: Liste [vincent, marsellus]
- drittes Element: Liste [butch, [father(butch), mother(butch)]]

```
[ [ ] | [ ] ]
```

- erstes Element: leere Liste
- Rest: leere Liste

~> Liste, die als einziges Element die leere Liste enthält

~> [[]]

```
[mia, [vincent, marsellus], [butch, [father(butch), mother(butch)]], [ ]]
```

- erstes Element: mia
- zweites Element: Liste [vincent, marsellus]
- drittes Element: Liste [butch, [father(butch), mother(butch)]]
- viertes Element: leere Liste

Übung 5.58

Sind die folgenden Ausdrücke syntaktisch korrekte Listen?

Wenn ja, wie viele Elemente haben sie jeweils?

- 1 `[1 | [2,3,4]]`
- 2 `[1,2,3 | []]`
- 3 `[1 | 2,3,4]`
- 4 `[1 | [2 | [3 | [4]]]]`
- 5 `[1,2,3,4 | []]`
- 6 `[[1,2] | 4]`
- 7 `[[1,2], [3,4] | [5,6,7]]`

Übung 5.59

Welche Antworten gibt der Interpreter auf die folgenden Anfragen?

- 1 `[a,b,c] = [a,[b,c]] .`
- 2 `[a,b,c] = [a|[b,c]] .`
- 3 `[a,b,c] = [a,b,[c]] .`
- 4 `[a,b,c] = [a,b|[c]] .`
- 5 `[a,b,c] = [a,b,c,[]] .`
- 6 `[a,b,c] = [a,b,c|[]] .`
- 7 `[] = _ .`
- 8 `[] = [_] .`

Menge [mia, vincent, yolanda, butch]

Menge [mia, vincent, yolanda, butch]

Folge [1,1,2,3,5,8,13,21]

Menge [mia, vincent, yolanda, butch]

Folge [1,1,2,3,5,8,13,21]

Vektor [1,0,3,7]

Menge [mia, vincent, yolanda, butch]

Folge [1,1,2,3,5,8,13,21]

Vektor [1,0,3,7]

Matrix [[1,0,0],[0,1,0],[0,0,1]]

Menge [mia, vincent, yolanda, butch]

Folge [1,1,2,3,5,8,13,21]

Vektor [1,0,3,7]

Matrix [[1,0,0],[0,1,0],[0,0,1]]

Objekt ['Marsellus','Wallace','3.8.1968','Los Angeles']

Menge [mia, vincent, yolanda, butch]

Folge [1,1,2,3,5,8,13,21]

Vektor [1,0,3,7]

Matrix [[1,0,0],[0,1,0],[0,0,1]]

Objekt ['Marsellus','Wallace','3.8.1968','Los Angeles']

Menge von Objekten [['Vincent','Vega'],['Jules','Winnfield']]

Menge [mia, vincent, yolanda, butch]

Folge [1,1,2,3,5,8,13,21]

Vektor [1,0,3,7]

Matrix [[1,0,0],[0,1,0],[0,0,1]]

Objekt ['Marsellus','Wallace','3.8.1968','Los Angeles']

Menge von Objekten [['Vincent','Vega'],['Jules','Winnfield']]

Satz [zed,is,dead]

- zweistellig

Das Prädikat member

- zweistellig
- erstes Argument: Term T

- zweistellig
- erstes Argument: Term `T`
- zweites Argument: Liste `L`

- zweistellig
- erstes Argument: Term `T`
- zweites Argument: Liste `L`
- Resultat: `true`, wenn `T` ein Element von `L` ist, sonst `false`

- zweistellig
- erstes Argument: Term T
- zweites Argument: Liste L
- Resultat: `true`, wenn T ein Element von L ist, sonst `false`

Beispiel 5.60

```
?- member(a,[a]).
```

- zweistellig
- erstes Argument: Term T
- zweites Argument: Liste L
- Resultat: `true`, wenn T ein Element von L ist, sonst `false`

Beispiel 5.60

```
?- member(a,[a]).  
true.
```

- zweistellig
- erstes Argument: Term T
- zweites Argument: Liste L
- Resultat: `true`, wenn T ein Element von L ist, sonst `false`

Beispiel 5.60

```
?- member(a,[a]).  
true.  
?- member(a,[b,c,d]).
```

- zweistellig
- erstes Argument: Term T
- zweites Argument: Liste L
- Resultat: `true`, wenn T ein Element von L ist, sonst `false`

Beispiel 5.60

```
?- member(a,[a]).  
true.  
?- member(a,[b,c,d]).  
false.
```

- zweistellig
- erstes Argument: Term T
- zweites Argument: Liste L
- Resultat: `true`, wenn T ein Element von L ist, sonst `false`

Beispiel 5.60

```
?- member(a,[a]).  
true.  
?- member(a,[b,c,d]).  
false.  
?- member(d,[b,c,d]).
```


- zweistellig
- erstes Argument: Term T
- zweites Argument: Liste L
- Resultat: `true`, wenn T ein Element von L ist, sonst `false`

Beispiel 5.60

```
?- member(a,[a]).  
true.  
?- member(a,[b,c,d]).  
false.  
?- member(d,[b,c,d]).  
true.
```

- zweistellig
- erstes Argument: Term T
- zweites Argument: Liste L
- Resultat: `true`, wenn T ein Element von L ist, sonst `false`

Beispiel 5.60

```
?- member(a,[a]).  
true.  
?- member(a,[b,c,d]).  
false.  
?- member(d,[b,c,d]).  
true.  
?- member(a, [[a],b]).
```

- zweistellig
- erstes Argument: Term T
- zweites Argument: Liste L
- Resultat: `true`, wenn T ein Element von L ist, sonst `false`

Beispiel 5.60

```
?- member(a,[a]).  
true.  
?- member(a,[b,c,d]).  
false.  
?- member(d,[b,c,d]).  
true.  
?- member(a, [[a],b]).  
false.
```

- zweistellig
- erstes Argument: Term T
- zweites Argument: Liste L
- Resultat: `true`, wenn T ein Element von L ist, sonst `false`

Beispiel 5.60

```
?- member(a,[a]).  
true.  
?- member(a,[b,c,d]).  
false.  
?- member(d,[b,c,d]).  
true.  
?- member(a, [[a],b]).  
false.  
?- member(b, [[a],b]).
```

- zweistellig
- erstes Argument: Term T
- zweites Argument: Liste L
- Resultat: `true`, wenn T ein Element von L ist, sonst `false`

Beispiel 5.60

```
?- member(a,[a]).  
true.  
?- member(a,[b,c,d]).  
false.  
?- member(d,[b,c,d]).  
true.  
?- member(a, [[a],b]).  
false.  
?- member(b, [[a],b]).  
true.
```

- zweistellig
- erstes Argument: Term T
- zweites Argument: Liste L
- Resultat: `true`, wenn T ein Element von L ist, sonst `false`

Beispiel 5.60

```
?- member(a,[a]).  
true.  
?- member(a,[b,c,d]).  
false.  
?- member(d,[b,c,d]).  
true.  
?- member(a, [[a],b]).  
false.  
?- member(b, [[a],b]).  
true.
```

```
?- member(X,[a,b,[c,d],father(butch),Y]).
```

- zweistellig
- erstes Argument: Term T
- zweites Argument: Liste L
- Resultat: `true`, wenn T ein Element von L ist, sonst `false`

Beispiel 5.60

```
?- member(a,[a]).  
true.  
?- member(a,[b,c,d]).  
false.  
?- member(d,[b,c,d]).  
true.  
?- member(a, [[a],b]).  
false.  
?- member(b, [[a],b]).  
true.
```

```
?- member(X,[a,b,[c,d],father(butch),Y]).  
X = a ;
```

Das Prädikat member

- zweistellig
- erstes Argument: Term T
- zweites Argument: Liste L
- Resultat: `true`, wenn T ein Element von L ist, sonst `false`

Beispiel 5.60

```
?- member(a,[a]).  
true.  
?- member(a,[b,c,d]).  
false.  
?- member(d,[b,c,d]).  
true.  
?- member(a, [[a],b]).  
false.  
?- member(b, [[a],b]).  
true.
```

```
?- member(X,[a,b,[c,d],father(butch),Y]).  
X = a ;  
X = b ;
```


Das Prädikat member

- zweistellig
- erstes Argument: Term T
- zweites Argument: Liste L
- Resultat: `true`, wenn T ein Element von L ist, sonst `false`

Beispiel 5.60

```
?- member(a,[a]).  
true.  
?- member(a,[b,c,d]).  
false.  
?- member(d,[b,c,d]).  
true.  
?- member(a, [[a],b]).  
false.  
?- member(b, [[a],b]).  
true.
```

```
?- member(X,[a,b,[c,d],father(butch),Y]).  
X = a ;  
X = b ;  
X = [c,d] ;
```

- zweistellig
- erstes Argument: Term T
- zweites Argument: Liste L
- Resultat: `true`, wenn T ein Element von L ist, sonst `false`

Beispiel 5.60

```
?- member(a,[a]).  
true.  
?- member(a,[b,c,d]).  
false.  
?- member(d,[b,c,d]).  
true.  
?- member(a, [[a],b]).  
false.  
?- member(b, [[a],b]).  
true.
```

```
?- member(X,[a,b,[c,d],father(butch),Y]).  
X = a ;  
X = b ;  
X = [c,d] ;  
X = father(butch) ;
```

- zweistellig
- erstes Argument: Term T
- zweites Argument: Liste L
- Resultat: `true`, wenn T ein Element von L ist, sonst `false`

Beispiel 5.60

```
?- member(a,[a]).  
true.  
?- member(a,[b,c,d]).  
false.  
?- member(d,[b,c,d]).  
true.  
?- member(a, [[a],b]).  
false.  
?- member(b, [[a],b]).  
true.
```

```
?- member(X,[a,b,[c,d],father(butch),Y]).  
X = a ;  
X = b ;  
X = [c,d] ;  
X = father(butch) ;  
X = Y.
```

- zweistellig
- erstes Argument: Term T
- zweites Argument: Liste L
- Resultat: `true`, wenn T ein Element von L ist, sonst `false`

Beispiel 5.60

```
?- member(a,[a]).  
true.  
?- member(a,[b,c,d]).  
false.  
?- member(d,[b,c,d]).  
true.  
?- member(a, [[a],b]).  
false.  
?- member(b, [[a],b]).  
true.
```

```
?- member(X,[a,b,[c,d],father(butch),Y]).  
X = a ;  
X = b ;  
X = [c,d] ;  
X = father(butch) ;  
X = Y.  
?- member(X,[]).
```

- zweistellig
- erstes Argument: Term T
- zweites Argument: Liste L
- Resultat: `true`, wenn T ein Element von L ist, sonst `false`

Beispiel 5.60

```
?- member(a,[a]).  
true.  
?- member(a,[b,c,d]).  
false.  
?- member(d,[b,c,d]).  
true.  
?- member(a, [[a],b]).  
false.  
?- member(b, [[a],b]).  
true.
```

```
?- member(X,[a,b,[c,d],father(butch),Y]).  
X = a ;  
X = b ;  
X = [c,d] ;  
X = father(butch) ;  
X = Y.  
?- member(X,[]).  
false.
```

- zweistellig
- erstes Argument: Term T
- zweites Argument: Liste L
- Resultat: `true`, wenn T ein Element von L ist, sonst `false`

Beispiel 5.60

```
?- member(a,[a]).  
true.  
?- member(a,[b,c,d]).  
false.  
?- member(d,[b,c,d]).  
true.  
?- member(a, [[a],b]).  
false.  
?- member(b, [[a],b]).  
true.
```

```
?- member(X,[a,b,[c,d],father(butch),Y]).  
X = a ;  
X = b ;  
X = [c,d] ;  
X = father(butch) ;  
X = Y.  
?- member(X,[]).  
false.  
?- member(X,a).
```

- zweistellig
- erstes Argument: Term T
- zweites Argument: Liste L
- Resultat: `true`, wenn T ein Element von L ist, sonst `false`

Beispiel 5.60

```
?- member(a,[a]).  
true.  
?- member(a,[b,c,d]).  
false.  
?- member(d,[b,c,d]).  
true.  
?- member(a, [[a],b]).  
false.  
?- member(b, [[a],b]).  
true.
```

```
?- member(X,[a,b,[c,d],father(butch),Y]).  
X = a ;  
X = b ;  
X = [c,d] ;  
X = father(butch) ;  
X = Y.  
?- member(X,[]).  
false.  
?- member(X,a).  
false.
```

```
member(X, [H|T]) :- X = H.
```



```
member(X, [H|T]) :- X = H.  
member(X, [H|T]) :- member(X, T).
```

```
member(X, [H|T]) :- X = H.  
member(X, [H|T]) :- member(X, T).
```

X ist ein Element von L, wenn

- X das erste Element von L ist oder

```
member(X, [H|T]) :- X = H.  
member(X, [H|T]) :- member(X, T).
```

X ist ein Element von L, wenn

- X das erste Element von L ist oder
- X ein Element des Rests der Liste ist.

```
member(X, [H|T]) :- X = H.  
member(X, [H|T]) :- member(X, T).
```

X ist ein Element von L, wenn

- X das erste Element von L ist oder
- X ein Element des Rests der Liste ist.

Optimierungen:

- Unifikation im Regelkopf

```
member(X, [X|T]).  
member(X, [H|T]) :- member(X, T).
```

X ist ein Element von L, wenn

- X das erste Element von L ist oder
- X ein Element des Rests der Liste ist.

Optimierungen:

- Unifikation im Regelkopf

```
member(X, [X|T]).  
member(X, [H|T]) :- member(X, T).
```

X ist ein Element von L, wenn

- X das erste Element von L ist oder
- X ein Element des Rests der Liste ist.

Optimierungen:

- Unifikation im Regelkopf
- Anonyme Variable vermeidet Warnung **Singleton Variable**

```
member(X, [X|_]).  
member(X, [_|T]) :- member(X, T).
```

X ist ein Element von L, wenn

- X das erste Element von L ist oder
- X ein Element des Rests der Liste ist.

Optimierungen:

- Unifikation im Regelkopf
- Anonyme Variable vermeidet Warnung **Singleton Variable**

Übung 5.61

Zeichnen Sie den Suchbaum für die Anfrage

```
?- member(X,[a,father(butch),[c,d]]).
```


- zweistellig

Das Prädikat length

- zweistellig
- erstes Argument: Liste

Das Prädikat `length`

- zweistellig
- erstes Argument: Liste
- zweites Argument: Länge der Liste

Das Prädikat `length`

- zweistellig
- erstes Argument: Liste
- zweites Argument: Länge der Liste

```
length([],0).
```

Das Prädikat length

- zweistellig
- erstes Argument: Liste
- zweites Argument: Länge der Liste

```
length([],0).  
length([_|T],X) :- length(T,Y), X is Y + 1.
```

Das Prädikat length

- zweistellig
- erstes Argument: Liste
- zweites Argument: Länge der Liste

```
length([],0).  
length([_|T],X) :- length(T,Y), X is Y + 1.
```

Beispiel 5.62

```
?- length([],X).
```

Das Prädikat length

- zweistellig
- erstes Argument: Liste
- zweites Argument: Länge der Liste

```
length([],0).  
length([_|T],X) :- length(T,Y), X is Y + 1.
```

Beispiel 5.62

```
?- length([],X).  
X = 0.
```

Das Prädikat length

- zweistellig
- erstes Argument: Liste
- zweites Argument: Länge der Liste

```
length([],0).  
length([_|T],X) :- length(T,Y), X is Y + 1.
```

Beispiel 5.62

```
?- length([],X).  
X = 0.  
?- length([a,b,c],X).
```


Das Prädikat length

- zweistellig
- erstes Argument: Liste
- zweites Argument: Länge der Liste

```
length([],0).  
length([_|T],X) :- length(T,Y), X is Y + 1.
```

Beispiel 5.62

```
?- length([],X).  
X = 0.  
?- length([a,b,c],X).  
X = 3.
```

Das Prädikat length

- zweistellig
- erstes Argument: Liste
- zweites Argument: Länge der Liste

```
length([],0).  
length([_|T],X) :- length(T,Y), X is Y + 1.
```

Beispiel 5.62

```
?- length([],X).  
X = 0.  
?- length([a,b,c],X).  
X = 3.  
?- length([a,[b,c]],X).
```

Das Prädikat length

- zweistellig
- erstes Argument: Liste
- zweites Argument: Länge der Liste

```
length([],0).  
length([_|T],X) :- length(T,Y), X is Y + 1.
```

Beispiel 5.62

```
?- length([],X).  
X = 0.  
?- length([a,b,c],X).  
X = 3.  
?- length([a,[b,c]],X).  
X = 2.
```

Das Prädikat length

- zweistellig
- erstes Argument: Liste
- zweites Argument: Länge der Liste

```
length([],0).  
length([_|T],X) :- length(T,Y), X is Y + 1.
```

Beispiel 5.62

```
?- length([],X).  
X = 0.  
?- length([a,b,c],X).  
X = 3.  
?- length([a,[b,c]],X).  
X = 2.
```

```
?- length([a,b],2).
```

Das Prädikat length

- zweistellig
- erstes Argument: Liste
- zweites Argument: Länge der Liste

```
length([],0).  
length([_|T],X) :- length(T,Y), X is Y + 1.
```

Beispiel 5.62

```
?- length([],X).  
X = 0.  
?- length([a,b,c],X).  
X = 3.  
?- length([a,[b,c]],X).  
X = 2.
```

```
?- length([a,b],2).  
true.
```

Das Prädikat length

- zweistellig
- erstes Argument: Liste
- zweites Argument: Länge der Liste

```
length([],0).  
length([_|T],X) :- length(T,Y), X is Y + 1.
```

Beispiel 5.62

```
?- length([],X).  
X = 0.  
?- length([a,b,c],X).  
X = 3.  
?- length([a,[b,c]],X).  
X = 2.
```

```
?- length([a,b],2).  
true.  
?- length(X,3).
```

Das Prädikat length

- zweistellig
- erstes Argument: Liste
- zweites Argument: Länge der Liste

```
length([],0).  
length([_|T],X) :- length(T,Y), X is Y + 1.
```

Beispiel 5.62

```
?- length([],X).  
X = 0.  
?- length([a,b,c],X).  
X = 3.  
?- length([a,[b,c]],X).  
X = 2.
```

```
?- length([a,b],2).  
true.  
?- length(X,3).  
X = [_G578, _G584, _G590].
```

Beispiel 5.63

Wissensbank

```
a2b([], []).  
a2b([a|A], [b|B]) :- a2b(A, B).
```


Beispiel 5.63

Wissensbank

```
a2b([], []).  
a2b([a|A], [b|B]) :- a2b(A, B).
```

a2b antwortet **true**, wenn

- beide Argumente Listen sind,

Beispiel 5.63

Wissensbank

```
a2b([], []).  
a2b([a|A], [b|B]) :- a2b(A, B).
```

a2b antwortet **true**, wenn

- beide Argumente Listen sind,
- beide Listen gleich lang sind,

Beispiel 5.63

Wissensbank

```
a2b([], []).  
a2b([a|A], [b|B]) :- a2b(A, B).
```

a2b antwortet **true**, wenn

- beide Argumente Listen sind,
- beide Listen gleich lang sind,
- die erste Liste nur aus a besteht,

Beispiel 5.63

Wissensbank

```
a2b([], []).  
a2b([a|A], [b|B]) :- a2b(A, B).
```

a2b antwortet **true**, wenn

- beide Argumente Listen sind,
- beide Listen gleich lang sind,
- die erste Liste nur aus a besteht,
- die zweite Liste nur aus b besteht.

Beispiel 5.63

Wissensbank

```
a2b([], []).  
a2b([a|A],[b|B]):- a2b(A,B).
```

a2b antwortet **true**, wenn

- beide Argumente Listen sind,
- beide Listen gleich lang sind,
- die erste Liste nur aus a besteht,
- die zweite Liste nur aus b besteht.

Interpreter

```
?- a2b([a,a,a],[b,b,b]).
```

Beispiel 5.63

Wissensbank

```
a2b([], []).  
a2b([a|A], [b|B]) :- a2b(A, B).
```

a2b antwortet **true**, wenn

- beide Argumente Listen sind,
- beide Listen gleich lang sind,
- die erste Liste nur aus a besteht,
- die zweite Liste nur aus b besteht.

Interpreter

```
?- a2b([a,a,a], [b,b,b]).  
true.
```

Beispiel 5.63

Wissensbank

```
a2b([], []).  
a2b([a|A], [b|B]) :- a2b(A, B).
```

a2b antwortet **true**, wenn

- beide Argumente Listen sind,
- beide Listen gleich lang sind,
- die erste Liste nur aus a besteht,
- die zweite Liste nur aus b besteht.

Interpreter

```
?- a2b([a,a,a], [b,b,b]).  
true.  
?- a2b([a,a,a], [b,b]).
```

Beispiel 5.63

Wissensbank

```
a2b([], []).  
a2b([a|A],[b|B]):- a2b(A,B).
```

a2b antwortet **true**, wenn

- beide Argumente Listen sind,
- beide Listen gleich lang sind,
- die erste Liste nur aus a besteht,
- die zweite Liste nur aus b besteht.

Interpreter

```
?- a2b([a,a,a],[b,b,b]).  
true.  
?- a2b([a,a,a],[b,b]).  
false.
```


Beispiel 5.63

Wissensbank

```
a2b([], []).  
a2b([a|A],[b|B]):- a2b(A,B).
```

a2b antwortet **true**, wenn

- beide Argumente Listen sind,
- beide Listen gleich lang sind,
- die erste Liste nur aus a besteht,
- die zweite Liste nur aus b besteht.

Interpreter

```
?- a2b([a,a,a],[b,b,b]).  
true.  
?- a2b([a,a,a],[b,b]).  
false.  
?- a2b(a,b).
```

Beispiel 5.63

Wissensbank

```
a2b([], []).  
a2b([a|A],[b|B]):- a2b(A,B).
```

a2b antwortet **true**, wenn

- beide Argumente Listen sind,
- beide Listen gleich lang sind,
- die erste Liste nur aus a besteht,
- die zweite Liste nur aus b besteht.

Interpreter

```
?- a2b([a,a,a],[b,b,b]).  
true.  
?- a2b([a,a,a],[b,b]).  
false.  
?- a2b(a,b).  
false.
```

Beispiel 5.63

Wissensbank

```
a2b([], []).  
a2b([a|A],[b|B]):- a2b(A,B).
```

a2b antwortet **true**, wenn

- beide Argumente Listen sind,
- beide Listen gleich lang sind,
- die erste Liste nur aus a besteht,
- die zweite Liste nur aus b besteht.

Interpreter

```
?- a2b([a,a,a],[b,b,b]).  
true.  
?- a2b([a,a,a],[b,b]).  
false.  
?- a2b(a,b).  
false.  
?- a2b([a,t],[b,b]).
```

Beispiel 5.63

Wissensbank

```
a2b([], []).  
a2b([a|A],[b|B]):- a2b(A,B).
```

a2b antwortet **true**, wenn

- beide Argumente Listen sind,
- beide Listen gleich lang sind,
- die erste Liste nur aus a besteht,
- die zweite Liste nur aus b besteht.

Interpreter

```
?- a2b([a,a,a],[b,b,b]).  
true.  
?- a2b([a,a,a],[b,b]).  
false.  
?- a2b(a,b).  
false.  
?- a2b([a,t],[b,b]).  
false.
```

Beispiel 5.63

Wissensbank

```
a2b([], []).  
a2b([a|A],[b|B]):- a2b(A,B).
```

a2b antwortet **true**, wenn

- beide Argumente Listen sind,
- beide Listen gleich lang sind,
- die erste Liste nur aus a besteht,
- die zweite Liste nur aus b besteht.

Interpreter

```
?- a2b([a,a,a],[b,b,b]).  
true.  
?- a2b([a,a,a],[b,b]).  
false.  
?- a2b(a,b).  
false.  
?- a2b([a,t],[b,b]).  
false.  
?- a2b([a,a,a],X).
```

Beispiel 5.63

Wissensbank

```
a2b([], []).  
a2b([a|A],[b|B]):- a2b(A,B).
```

a2b antwortet **true**, wenn

- beide Argumente Listen sind,
- beide Listen gleich lang sind,
- die erste Liste nur aus a besteht,
- die zweite Liste nur aus b besteht.

Interpreter

```
?- a2b([a,a,a],[b,b,b]).  
true.  
?- a2b([a,a,a],[b,b]).  
false.  
?- a2b(a,b).  
false.  
?- a2b([a,t],[b,b]).  
false.  
?- a2b([a,a,a],X).  
X = [b,b,b].
```

Beispiel 5.63

Wissensbank

```
a2b([], []).  
a2b([a|A],[b|B]):- a2b(A,B).
```

a2b antwortet **true**, wenn

- beide Argumente Listen sind,
- beide Listen gleich lang sind,
- die erste Liste nur aus a besteht,
- die zweite Liste nur aus b besteht.

Interpreter

```
?- a2b([a,a,a],[b,b,b]).  
true.  
?- a2b([a,a,a],[b,b]).  
false.  
?- a2b(a,b).  
false.  
?- a2b([a,t],[b,b]).  
false.  
?- a2b([a,a,a],X).  
X = [b,b,b].  
?- a2b([a|T],[b,b,b]).
```

Beispiel 5.63

Wissensbank

```
a2b([], []).  
a2b([a|A],[b|B]):- a2b(A,B).
```

a2b antwortet **true**, wenn

- beide Argumente Listen sind,
- beide Listen gleich lang sind,
- die erste Liste nur aus a besteht,
- die zweite Liste nur aus b besteht.

Interpreter

```
?- a2b([a,a,a],[b,b,b]).  
true.  
?- a2b([a,a,a],[b,b]).  
false.  
?- a2b(a,b).  
false.  
?- a2b([a,t],[b,b]).  
false.  
?- a2b([a,a,a],X).  
X = [b,b,b].  
?- a2b([a|T],[b,b,b]).  
T = [a,a].
```


Übung 5.64

Zeichnen Sie den Suchbaum für die Anfrage

Interpreter

```
?- a2b([a,a,a],X).
```

Übung 5.65

Schreiben Sie ein zweistelliges Prädikat `twice`, dessen erstes Argument eine Liste ist, und dessen zweites Argument eine Liste ist, die jedes Element der ersten Liste zweimal enthält.

Interpreter

```
?- twice([1,b,mia],[1,1,b,b,mia,mia]).  
true.  
?- twice([a,b,c],X).  
X = [a,a,b,b,c,c].
```

- dreistellig

- dreistellig
- Resultat: `true`, wenn alle Argumente Listen sind und die Konkatenation von erstem und zweitem Argument gleich dem dritten Argument ist

- dreistellig
- Resultat: `true`, wenn alle Argumente Listen sind und die Konkatenation von erstem und zweitem Argument gleich dem dritten Argument ist
- kann zum Aneinanderhängen und Aufteilen von Listen verwendet werden

- dreistellig
- Resultat: `true`, wenn alle Argumente Listen sind und die Konkatenation von erstem und zweitem Argument gleich dem dritten Argument ist
- kann zum Aneinanderhängen und Aufteilen von Listen verwendet werden

- dreistellig
- Resultat: `true`, wenn alle Argumente Listen sind und die Konkatenation von erstem und zweitem Argument gleich dem dritten Argument ist
- kann zum Aneinanderhängen und Aufteilen von Listen verwendet werden

```
append([ ],L,L).  
append([H|T],L,[H|R]) :- append(T,L,R).
```

- dreistellig
- Resultat: `true`, wenn alle Argumente Listen sind und die Konkatenation von erstem und zweitem Argument gleich dem dritten Argument ist
- kann zum Aneinanderhängen und Aufteilen von Listen verwendet werden

```
append([ ],L,L).  
append([H|T],L,[H|R]) :- append(T,L,R).
```

- Die Konkatenation der leeren Liste mit einer beliebigen Liste `L` ist gleich `L`.

- dreistellig
- Resultat: `true`, wenn alle Argumente Listen sind und die Konkatenation von erstem und zweitem Argument gleich dem dritten Argument ist
- kann zum Aneinanderhängen und Aufteilen von Listen verwendet werden

```
append([ ],L,L).  
append([H|T],L,[H|R]) :- append(T,L,R).
```

- Die Konkatenation der leeren Liste mit einer beliebigen Liste `L` ist gleich `L`.
- Die Konkatenation einer nichtleeren Liste `[H|T]` mit einer Liste `L` hat

- dreistellig
- Resultat: `true`, wenn alle Argumente Listen sind und die Konkatenation von erstem und zweitem Argument gleich dem dritten Argument ist
- kann zum Aneinanderhängen und Aufteilen von Listen verwendet werden

```
append([ ],L,L).  
append([H|T],L,[H|R]) :- append(T,L,R).
```

- Die Konkatenation der leeren Liste mit einer beliebigen Liste `L` ist gleich `L`.
- Die Konkatenation einer nichtleeren Liste `[H|T]` mit einer Liste `L` hat
 - als Kopf `H`

- dreistellig
- Resultat: `true`, wenn alle Argumente Listen sind und die Konkatenation von erstem und zweitem Argument gleich dem dritten Argument ist
- kann zum Aneinanderhängen und Aufteilen von Listen verwendet werden

```
append([ ],L,L).  
append([H|T],L,[H|R]) :- append(T,L,R).
```

- Die Konkatenation der leeren Liste mit einer beliebigen Liste `L` ist gleich `L`.
- Die Konkatenation einer nichtleeren Liste `[H|T]` mit einer Liste `L` hat
 - als Kopf `H`
 - als Rumpf die [Konkatenation von `T` und `L`](#)

- dreistellig
- Resultat: `true`, wenn alle Argumente Listen sind und die Konkatenation von erstem und zweitem Argument gleich dem dritten Argument ist
- kann zum Aneinanderhängen und Aufteilen von Listen verwendet werden

```
append([ ],L,L).  
append([H|T],L,[H|R]) :- append(T,L,R).
```

- Die Konkatenation der leeren Liste mit einer beliebigen Liste `L` ist gleich `L`.
- Die Konkatenation einer nichtleeren Liste `[H|T]` mit einer Liste `L` hat
 - als Kopf `H`
 - als Rumpf die [Konkatenation von `T` und `L`](#)
- analog zum Prädikat `add/3` in Beispiel 5.49

Beispiel 5.66

Wissensbank

```
append([ ],L,L).  
append([H|T],L,[H|R]):-  
    append(T,L,R).
```

```
append([a,b,c],[1,2,3],A)
```

Beispiel 5.66

Wissensbank

```
append([ ],L,L).  
append([H|T],L,[H|R]):-  
    append(T,L,R).
```

```
append([a,b,c],[1,2,3],A)
```

```
| H1 = a, T1 = [b,c]  
| L1 = [1,2,3], A = [a|R1]
```

```
append([b,c],[1,2,3],R1)
```

Beispiel 5.66

Wissensbank

```
append([ ],L,L).  
append([H|T],L,[H|R]):-  
    append(T,L,R).
```

```
append([a,b,c],[1,2,3],A)
```

```
| H1 = a, T1 = [b,c]  
| L1 = [1,2,3], A = [a|R1]
```

```
append([b,c],[1,2,3],R1)
```

```
| H2 = b, T2 = [c]  
| L2 = [1,2,3], R1 = [b|R2]
```

```
append([c],[1,2,3],R2)
```

Beispiel 5.66

Wissensbank

```
append([ ],L,L).  
append([H|T],L,[H|R]):-  
    append(T,L,R).
```

```
append([a,b,c],[1,2,3],A)
```

```
| H1 = a, T1 = [b,c]  
| L1 = [1,2,3], A = [a|R1]
```

```
append([b,c],[1,2,3],R1)
```

```
| H2 = b, T2 = [c]  
| L2 = [1,2,3], R1 = [b|R2]
```

```
append([c],[1,2,3],R2)
```

```
| H3 = c, T3 = [ ]  
| L3 = [1,2,3], R2 = [c|R3]
```

```
append([ ],[1,2,3],R3)
```

Beispiel 5.66

Wissensbank

```
append([ ],L,L).  
append([H|T],L,[H|R]):-  
    append(T,L,R).
```

append([a,b,c],[1,2,3],A)

| H1 = a, T1 = [b,c]
| L1 = [1,2,3], A = [a|R1]

append([b,c],[1,2,3],R1)

| H2 = b, T2 = [c]
| L2 = [1,2,3], R1 = [b|R2]

append([c],[1,2,3],R2)

| H3 = c, T3 = []
| L3 = [1,2,3], R2 = [c|R3]

append([],[1,2,3],R3)

| L4 = [1,2,3], R3 = L4



Beispiel 5.66

Wissensbank

```
append([ ],L,L).  
append([H|T],L,[H|R]):-  
    append(T,L,R).
```

append([a,b,c],[1,2,3],A)

H1 = a, T1 = [b,c]
L1 = [1,2,3], A = [a|R1]

append([b,c],[1,2,3],R1)

H2 = b, T2 = [c]
L2 = [1,2,3], R1 = [b|R2]

append([c],[1,2,3],R2)

H3 = c, T3 = []
L3 = [1,2,3], R2 = [c|R3]

append([],[1,2,3],R3)

L4 = [1,2,3], R3 = L4



A = [a,b,c,1,2,3]

Beispiel 5.66

Wissensbank

```
append([ ],L,L).  
append([H|T],L,[H|R]):-  
    append(T,L,R).
```

$A = [a|R1]$
 $= [a|[b|R2]]$
 $= [a|[b|[c|R3]]]$
 $= [a|[b|[c|[1,2,3]]]]$
 $= [a,b,c,1,2,3]$

append([a,b,c],[1,2,3],A)

$H1 = a, T1 = [b,c]$
 $L1 = [1,2,3], A = [a|R1]$

append([b,c],[1,2,3],R1)

$H2 = b, T2 = [c]$
 $L2 = [1,2,3], R1 = [b|R2]$

append([c],[1,2,3],R2)

$H3 = c, T3 = []$
 $L3 = [1,2,3], R2 = [c|R3]$

append([],[1,2,3],R3)

$L4 = [1,2,3], R3 = L4$



$A = [a,b,c,1,2,3]$

Beispiel 5.67

■ Konkatenation von Listen

```
?- append([a,b],[c,d],X).  
X = [a,b,c,d].
```

Beispiel 5.67

■ Konkatenation von Listen

```
?- append([a,b],[c,d],X).  
X = [a,b,c,d].
```

■ Differenz-Listen

```
?- append([a,b],X,[a,b,c,d]).  
X = [c,d].  
?- append(X,[c,d],[a,b,c,d]).  
X = [a,b] ;  
false.
```

Beispiel 5.67

■ Konkatenation von Listen

```
?- append([a,b],[c,d],X).  
X = [a,b,c,d].
```

■ Differenz-Listen

```
?- append([a,b],X,[a,b,c,d]).  
X = [c,d].  
?- append(X,[c,d],[a,b,c,d]).  
X = [a,b] ;  
false.
```

■ Aufteilungsmöglichkeiten für Listen

```
?- append(X,Y,[a,b,c,d]).  
X = [ ], Y = [a,b,c,d] ;  
X = [a], Y = [b,c,d] ;  
X = [a,b], Y = [c,d] ;  
X = [a,b,c], Y = [d] ;  
X = [a,b,c,d], Y = [ ] ;  
false.
```

Wissensbank

```
prefix(Pre,List):-  
    append(Pre,_,List).
```

```
suffix(Suf,List):-  
    append(_,Suf,List).
```

```
sublist(Sub,List):-  
    prefix(Pre,List),  
    suffix(Sub,Pre).
```

Wissensbank

```
prefix(Pre,List):-  
    append(Pre,_,List).  
  
suffix(Suf,List):-  
    append(_,Suf,List).  
  
sublist(Sub,List):-  
    prefix(Pre,List),  
    suffix(Sub,Pre).
```

Interpreter

```
?- prefix(X,[a,b,c]).  
X = [] ;  
X = [a] ;  
X = [a, b] ;  
X = [a, b, c] ;  
false.
```

Wissensbank

```
prefix(Pre,List):-  
    append(Pre,_,List).  
  
suffix(Suf,List):-  
    append(_,Suf,List).  
  
sublist(Sub,List):-  
    prefix(Pre,List),  
    suffix(Sub,Pre).
```

Interpreter

```
?- prefix(X,[a,b,c]).  
X = [] ;  
X = [a] ;  
X = [a, b] ;  
X = [a, b, c] ;  
false.  
  
?- suffix(X,[a,b,c]).  
X = [a, b, c] ;  
X = [b, c] ;  
X = [c] ;  
X = [] ;  
false.
```


Wissensbank

```
prefix(Pre,List):-  
    append(Pre,_,List).  
  
suffix(Suf,List):-  
    append(_,Suf,List).  
  
sublist(Sub,List):-  
    prefix(Pre,List),  
    suffix(Sub,Pre).
```

Interpreter

```
?- prefix(X,[a,b,c]).  
X = [] ;  
X = [a] ;  
X = [a, b] ;  
X = [a, b, c] ;  
false.  
?- suffix(X,[a,b,c]).  
X = [a, b, c] ;  
X = [b, c] ;  
X = [c] ;  
X = [] ;  
false.
```

Interpreter

```
?- sublist(X,[a,b,c]).  
X = [] ;  
X = [a] ;  
X = [] ;  
X = [a, b] ;  
X = [b] ;  
X = [] ;  
X = [a, b, c] ;  
X = [b, c] ;  
X = [c] ;  
X = [] ;  
false.
```

Übung 5.68

Eine Straße besteht aus drei Häusern.

- Sie haben die Farben Rot, Grün und Blau.
- Sie werden von einem Engländer, einem Japaner und einem Spanier bewohnt.
- In jedem Haus gibt es ein Haustier: Eine Schnecke, einen Jaguar und ein Zebra.

Wir wissen:

- Der Engländer wohnt im roten Haus.
- Der Jaguar wohnt beim Spanier.
- Der Japaner wohnt rechts vom Halter der Schnecke.
- Der Halter der Schnecke wohnt rechts vom grünen Haus.

Wer hat das Zebra?

Tips:

- Nutzen Sie Listen, um die Straße und deren Häuser zu repräsentieren.
- Nutzen Sie die Prädikate `member` und `sublist`, um die Aussagen zu codieren.

- [Head | Tail]

- [Head | Tail]
Head erstes Element

- [Head | Tail]

- Head erstes Element

- Tail Liste der restlichen Elemente

- [Head | Tail]
 - Head erstes Element
 - Tail Liste der restlichen Elemente
- leere Liste: []

- [Head | Tail]
 - Head erstes Element
 - Tail Liste der restlichen Elemente
- leere Liste: []
- Rekursive Bearbeitung

- [Head | Tail]
 - Head erstes Element
 - Tail Liste der restlichen Elemente
- leere Liste: []
- Rekursive Bearbeitung
 - Basisklausel: Bearbeitung der leeren Liste

- [Head | Tail]

- Head erstes Element

- Tail Liste der restlichen Elemente

- leere Liste: []

- Rekursive Bearbeitung

- Basisklausel: Bearbeitung der leeren Liste

- Rekursive Klausel: Bearbeitung des Kopfs, rekursive Bearbeitung des Rumpfs

- [Head | Tail]

- Head erstes Element

- Tail Liste der restlichen Elemente

- leere Liste: []

- Rekursive Bearbeitung

- Basisklausel: Bearbeitung der leeren Liste

- Rekursive Klausel: Bearbeitung des Kopfs, rekursive Bearbeitung des Rumpfs

- wichtige Prädikate: member, length, append

1. Einführung

2. Mengen

3. Aussagenlogik

4. Prädikatenlogik

5. Prolog

5.1 Terme

5.2 Fakten, Regeln und Anfragen

5.3 Unifikation, Termgleichheit und Arithmetik

5.4 Horn-Resolution und Suchbäume

5.5 Rekursion

5.6 Listen

5.7 Der Cut

Beispiel 5.69 (Betrag)

Wissensbank

```
abs(X,X) :- X >= 0.
```

```
abs(X,Y) :- X < 0, Y is -X.
```

Beispiel 5.69 (Betrag)

Wissensbank

```
abs(X,X) :- X >= 0.  
abs(X,Y) :- X < 0, Y is -X.
```

Interpreter

```
?- abs(-5,R).
```

Beispiel 5.69 (Betrag)

Wissensbank

```
abs(X,X) :- X >= 0.  
abs(X,Y) :- X < 0, Y is -X.
```

Interpreter

```
?- abs(-5,R).  
R = 5.
```

Beispiel 5.69 (Betrag)

Wissensbank

```
abs(X,X) :- X >= 0.  
abs(X,Y) :- X < 0, Y is -X.
```

Interpreter

```
?- abs(-5,R).  
R = 5.
```

```
?- abs(5,R).
```

Beispiel 5.69 (Betrag)

Wissensbank

```
abs(X,X) :- X >= 0.  
abs(X,Y) :- X < 0, Y is -X.
```

Interpreter

```
?- abs(-5,R).  
R = 5.
```

```
?- abs(5,R).  
R = 5 ;
```


Beispiel 5.69 (Betrag)

Wissensbank

```
abs(X,X) :- X >= 0.  
abs(X,Y) :- X < 0, Y is -X.
```

Interpreter

```
?- abs(-5,R).  
R = 5.
```

```
?- abs(5,R).  
R = 5 ;  
false.
```

Beispiel 5.69 (Betrag)

Wissensbank

```
abs(X,X) :- X >= 0.  
abs(X,Y) :- X < 0, Y is -X.
```

```
abs(5,R)
```

Interpreter

```
?- abs(-5,R).  
R = 5.
```

```
?- abs(5,R).  
R = 5 ;  
false.
```

Beispiel 5.69 (Betrag)

Wissensbank

```
abs(X,X) :- X >= 0.  
abs(X,Y) :- X < 0, Y is -X.
```

Interpreter

```
?- abs(-5,R).  
R = 5.
```

```
?- abs(5,R).  
R = 5 ;  
false.
```

abs(5,R)

X = 5
R = X

X = 5
R = Y

5 >= 0

Beispiel 5.69 (Betrag)

Wissensbank

```
abs(X,X) :- X >= 0.  
abs(X,Y) :- X < 0, Y is -X.
```

Interpreter

```
?- abs(-5,R).  
R = 5.
```

```
?- abs(5,R).  
R = 5 ;  
false.
```

abs(5,R)

X = 5
R = X

X = 5
R = Y

5 >= 0



R = 5

Beispiel 5.69 (Betrag)

Wissensbank

```
abs(X,X) :- X >= 0.  
abs(X,Y) :- X < 0, Y is -X.
```

Interpreter

```
?- abs(-5,R).  
R = 5.
```

```
?- abs(5,R).  
R = 5 ;  
false.
```

abs(5,R)

X = 5
R = X

X = 5
R = Y

5 >= 0

5 < 0
Y is - 5



R = 5

Beispiel 5.69 (Betrag)

Wissensbank

```
abs(X,X) :- X >= 0.  
abs(X,Y) :- X < 0, Y is -X.
```

Interpreter

```
?- abs(-5,R).  
R = 5.
```

```
?- abs(5,R).  
R = 5 ;  
false.
```

abs(5,R)

X = 5
R = X

X = 5
R = Y

5 >= 0

5 < 0
Y is - 5

X



R = 5

Beispiel 5.69 (Betrag)

Wissensbank

```
abs(X,X) :- X >= 0.  
abs(X,Y) :- X < 0, Y is -X.
```

Interpreter

```
?- abs(-5,R).  
R = 5.
```

```
?- abs(5,R).  
R = 5 ;  
false.
```

abs(5,R)

X = 5
R = X

X = 5
R = Y

5 >= 0

5 < 0
Y is - 5

X



R = 5

- Verzweigungspunkt für **jeden** Aufruf von abs

Beispiel 5.69 (Betrag)

Wissensbank

```
abs(X,X) :- X >= 0.  
abs(X,Y) :- X < 0, Y is -X.
```

Interpreter

```
?- abs(-5,R).  
R = 5.
```

```
?- abs(5,R).  
R = 5 ;  
false.
```

abs(5,R)

X = 5
R = X

X = 5
R = Y

5 >= 0

5 < 0
Y is - 5

X



R = 5

- Verzweigungspunkt für **jeden** Aufruf von abs
- Wenn folgendes Goal fehlschlägt $\leadsto$ Backtracking
Z.B. abs(5,X), X > 10, ...

Beispiel 5.69 (Betrag)

Wissensbank

```
abs(X,X) :- X >= 0.  
abs(X,Y) :- X < 0, Y is -X.
```

Interpreter

```
?- abs(-5,R).  
R = 5.
```

```
?- abs(5,R).  
R = 5 ;  
false.
```

abs(5,R)

X = 5
R = X

X = 5
R = Y

5 >= 0

5 < 0
Y is - 5

X



R = 5

- Verzweigungspunkt für **jeden** Aufruf von abs
- Wenn folgendes Goal fehlschlägt $\leadsto$ Backtracking
Z.B. abs(5,X), X > 10, ...
- Backtracking ist sinnlos, da Alternativen sich ausschließen

Beispiel 5.69 (Betrag)

Wissensbank

```
abs(X,X) :- X >= 0.  
abs(X,Y) :- X < 0, Y is -X.
```

Interpreter

```
?- abs(-5,R).  
R = 5.
```

```
?- abs(5,R).  
R = 5 ;  
false.
```

abs(5,R)

X = 5
R = X

X = 5
R = Y

5 >= 0

5 < 0
Y is - 5

X



R = 5

- Verzweigungspunkt für **jeden** Aufruf von abs
- Wenn folgendes Goal fehlschlägt $\leadsto$ Backtracking
Z.B. abs(5,X), X > 10, ...
- Backtracking ist sinnlos, da Alternativen sich ausschließen
 - Platzverschwendung durch Verzweigungspunkt

Beispiel 5.69 (Betrag)

Wissensbank

```
abs(X,X) :- X >= 0.  
abs(X,Y) :- X < 0, Y is -X.
```

Interpreter

```
?- abs(-5,R).  
R = 5.
```

```
?- abs(5,R).  
R = 5 ;  
false.
```

abs(5,R)

X = 5
R = X

X = 5
R = Y

5 >= 0

5 < 0
Y is - 5

X



✓

R = 5

- Verzweigungspunkt für **jeden** Aufruf von abs
- Wenn folgendes Goal fehlschlägt $\leadsto$ Backtracking
Z.B. abs(5,X), X > 10, ...
- Backtracking ist sinnlos, da Alternativen sich ausschließen
 - Platzverschwendung durch Verzweigungspunkt
 - Zeitverschwendung durch Backtracking

Beispiel 5.69 (Betrag)

Wissensbank

```
abs(X,X) :- X >= 0.  
abs(X,Y) :- X < 0, Y is -X.
```

Interpreter

```
?- abs(-5,R).  
R = 5.
```

```
?- abs(5,R).  
R = 5 ;  
false.
```

abs(5,R)

X = 5
R = X

X = 5
R = Y

5 >= 0

5 < 0
Y is - 5

X



✓

R = 5

- Verzweigungspunkt für **jeden** Aufruf von abs
- Wenn folgendes Goal fehlschlägt $\leadsto$ Backtracking
Z.B. abs(5,X), X > 10, ...
- Backtracking ist sinnlos, da Alternativen sich ausschließen
 - Platzverschwendung durch Verzweigungspunkt
 - Zeitverschwendung durch Backtracking
- Wünschenswert: Verzweigungspunkte verhindern

Definition 5.70 (Eltern-Goal, Subgoal)

Gegeben sei eine Regel mit Kopf K und Rumpf R .

Das Goal in K heißt **Eltern-Goal** der Goals in R ; die Goals in R heißen **Subgoals** von K .

Definition 5.70 (Eltern-Goal, Subgoal)

Gegeben sei eine Regel mit Kopf K und Rumpf R .

Das Goal in K heißt **Eltern-Goal** der Goals in R ; die Goals in R heißen **Subgoals** von K .

Beispiel 5.71

Wissensbank

$a(X) :- b(X), c(X), d(X).$

Definition 5.70 (Eltern-Goal, Subgoal)

Gegeben sei eine Regel mit Kopf K und Rumpf R .
Das Goal in K heißt **Eltern-Goal** der Goals in R ; die Goals in R heißen **Subgoals** von K .

Beispiel 5.71

Wissensbank

$a(X) :- b(X), c(X), d(X).$

- $a(X)$ ist Eltern-Goal von $b(X)$, $c(X)$ und $d(X)$.

Definition 5.70 (Eltern-Goal, Subgoal)

Gegeben sei eine Regel mit Kopf K und Rumpf R .

Das Goal in K heißt **Eltern-Goal** der Goals in R ; die Goals in R heißen **Subgoals** von K .

Beispiel 5.71

Wissensbank

$a(X) :- b(X), c(X), d(X).$

- $a(X)$ ist Eltern-Goal von $b(X)$, $c(X)$ und $d(X)$.
- $b(X)$, $c(X)$ und $d(X)$ sind Subgoals von $a(X)$.

- Name: `Cut`

- Name: `Cut`
- nullstellig

- Name: `Cut`
- nullstellig
- immer erfolgreich

- Name: `Cut`
- nullstellig
- immer erfolgreich
- entfernt `Verzweigungspunkte`, die nach dem Aufruf des Eltern-Goals angelegt wurden

- Name: **Cut**
- nullstellig
- immer erfolgreich
- entfernt **Verzweigungspunkte**, die nach dem Aufruf des Eltern-Goals angelegt wurden
 - kein Backtracking vor den Cut

- Name: **Cut**
- nullstellig
- immer erfolgreich
- entfernt **Verzweigungspunkte**, die nach dem Aufruf des Eltern-Goals angelegt wurden
 - kein Backtracking vor den Cut
 - bestehende Variablenbindungen werden fixiert

- Name: **Cut**
- nullstellig
- immer erfolgreich
- entfernt **Verzweigungspunkte**, die nach dem Aufruf des Eltern-Goals angelegt wurden
 - kein Backtracking vor den Cut
 - bestehende Variablenbindungen werden fixiert
 - keine alternativen Unifikationen

- Name: **Cut**
- nullstellig
- immer erfolgreich
- entfernt **Verzweigungspunkte**, die nach dem Aufruf des Eltern-Goals angelegt wurden
 - kein Backtracking vor den Cut
 - bestehende Variablenbindungen werden fixiert
 - keine alternativen Unifikationen
 - keine alternativen Klauseln zur Erfüllung eines Goals

- Name: **Cut**
- nullstellig
- immer erfolgreich
- entfernt **Verzweigungspunkte**, die nach dem Aufruf des Eltern-Goals angelegt wurden
 - kein Backtracking vor den Cut
 - bestehende Variablenbindungen werden fixiert
 - keine alternativen Unifikationen
 - keine alternativen Klauseln zur Erfüllung eines Goals
- prozedural, nicht deklarativ

- Name: **Cut**
- nullstellig
- immer erfolgreich
- entfernt **Verzweigungspunkte**, die nach dem Aufruf des Eltern-Goals angelegt wurden
 - kein Backtracking vor den Cut
 - bestehende Variablenbindungen werden fixiert
 - keine alternativen Unifikationen
 - keine alternativen Klauseln zur Erfüllung eines Goals
- prozedural, nicht deklarativ
- ermöglicht neue Programmkonstrukte (Negation)

- Name: **Cut**
- nullstellig
- immer erfolgreich
- entfernt **Verzweigungspunkte**, die nach dem Aufruf des Eltern-Goals angelegt wurden
 - kein Backtracking vor den Cut
 - bestehende Variablenbindungen werden fixiert
 - keine alternativen Unifikationen
 - keine alternativen Klauseln zur Erfüllung eines Goals
- prozedural, nicht deklarativ
- ermöglicht neue Programmkonstrukte (Negation)
- kann Effizienz erhöhen

Beispiel 5.72 (Betrag mit Cut)

Wissensbank

```
abs(X,X) :- X >= 0, !.  
abs(X,Y) :- X < 0, Y is -X.
```

Beispiel 5.72 (Betrag mit Cut)

Wissensbank

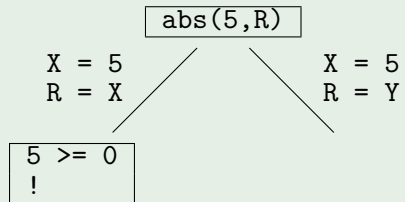
```
abs(X,X) :- X >= 0, !.  
abs(X,Y) :- X < 0, Y is -X.
```

abs(5,R)

Beispiel 5.72 (Betrag mit Cut)

Wissensbank

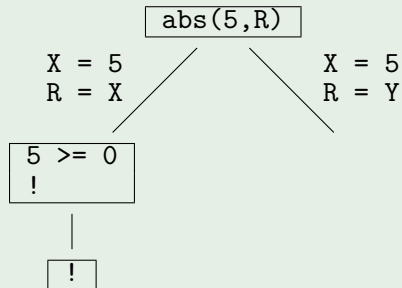
```
abs(X,X) :- X >= 0, !.  
abs(X,Y) :- X < 0, Y is -X.
```



Beispiel 5.72 (Betrag mit Cut)

Wissensbank

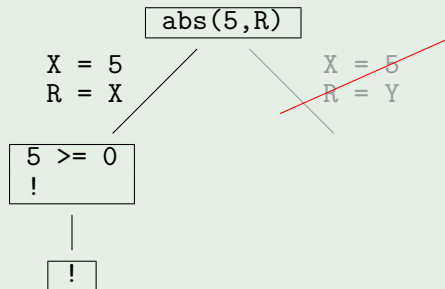
```
abs(X,X) :- X >= 0, !.  
abs(X,Y) :- X < 0, Y is -X.
```



Beispiel 5.72 (Betrag mit Cut)

Wissensbank

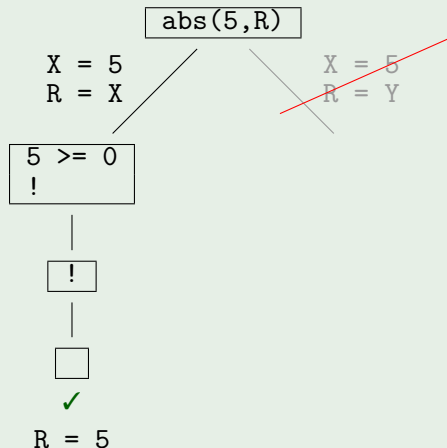
```
abs(X,X) :- X >= 0, !.  
abs(X,Y) :- X < 0, Y is -X.
```



Beispiel 5.72 (Betrag mit Cut)

Wissensbank

```
abs(X,X) :- X >= 0, !.  
abs(X,Y) :- X < 0, Y is -X.
```

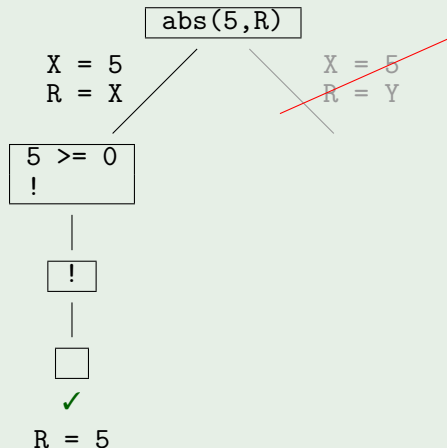


Beispiel 5.72 (Betrag mit Cut)

Wissensbank

```
abs(X,X) :- X >= 0, !.  
abs(X,Y) :- X < 0, Y is -X.
```

Beim Erreichen des Cut:



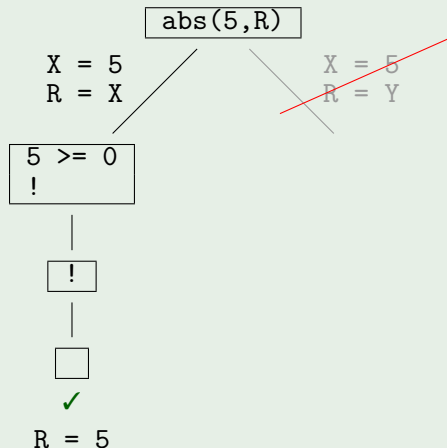
Beispiel 5.72 (Betrag mit Cut)

Wissensbank

```
abs(X,X) :- X >= 0, !.  
abs(X,Y) :- X < 0, Y is -X.
```

Beim Erreichen des Cut:

- Verzweigungspunkt gelöscht



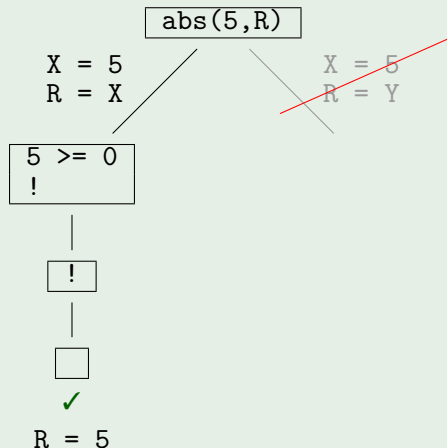
Beispiel 5.72 (Betrag mit Cut)

Wissensbank

```
abs(X,X) :- X >= 0, !.  
abs(X,Y) :- X < 0, Y is -X.
```

Beim Erreichen des Cut:

- Verzweigungspunkt gelöscht
- Variablenbindungen fixiert



Beispiel 5.72 (Betrag mit Cut)

Wissensbank

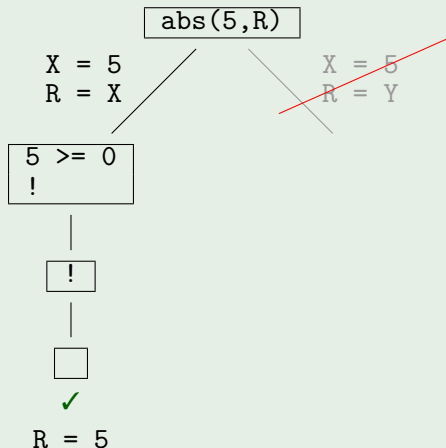
```
abs(X,X) :- X >= 0, !.  
abs(X,Y) :- X < 0, Y is -X.
```

Beim Erreichen des Cut:

- Verzweigungspunkt gelöscht
- Variablenbindungen fixiert

Interpreter

```
?- abs(5,R).  
R = 5.
```



Übung 5.73

Das einstellige Prädikat `number` gibt genau dann `true` zurück, wenn das Argument eine Zahl ist:

Interpreter

```
?- number(1).  
true.  
?- number(X).  
false.  
?- number([1]).  
false.  
?- number(1+2).  
false.  
?- X is 1 + 2, number(X).  
X = 3.
```

Schreiben Sie ein zweistelliges Prädikat `filter`, das als erstes Argument eine Liste erhält und im zweiten Argument die Liste derjenigen Elemente zurückgibt, die keine Zahlen sind.

- nullstellig

- nullstellig
- löst Backtracking aus

- nullstellig
- löst Backtracking aus
- kann zusammen mit `Cut` zum Beschreiben von Ausnahmen verwendet werden

- nullstellig
- löst Backtracking aus
- kann zusammen mit Cut zum Beschreiben von Ausnahmen verwendet werden

Beispiel 5.74 („Vincent mag alle Burger außer dem Big Kahuna“)

Wissensbank

```
likes(vincent, kahuna) :- !, fail.  
likes(vincent, X) :- burger(X).
```

```
burger(bigMac).  
burger(kahuna).  
burger(royale).
```

- nullstellig
- löst Backtracking aus
- kann zusammen mit Cut zum Beschreiben von Ausnahmen verwendet werden

Beispiel 5.74 („Vincent mag alle Burger außer dem Big Kahuna“)

Wissensbank

```
likes(vincent, kahuna) :- !, fail.  
likes(vincent, X) :- burger(X).
```

```
burger(bigMac).  
burger(kahuna).  
burger(royale).
```

Interpreter

```
?- likes(vincent, bigMac).
```

- nullstellig
- löst Backtracking aus
- kann zusammen mit Cut zum Beschreiben von Ausnahmen verwendet werden

Beispiel 5.74 („Vincent mag alle Burger außer dem Big Kahuna“)

Wissensbank

```
likes(vincent, kahuna) :- !, fail.  
likes(vincent, X) :- burger(X).
```

```
burger(bigMac).  
burger(kahuna).  
burger(royale).
```

Interpreter

```
?- likes(vincent, bigMac).  
true.
```

- nullstellig
- löst Backtracking aus
- kann zusammen mit Cut zum Beschreiben von Ausnahmen verwendet werden

Beispiel 5.74 („Vincent mag alle Burger außer dem Big Kahuna“)

Wissensbank

```
likes(vincent, kahuna) :- !, fail.  
likes(vincent, X) :- burger(X).
```

```
burger(bigMac).  
burger(kahuna).  
burger(royale).
```

Interpreter

```
?- likes(vincent, bigMac).  
true.  
?- likes(vincent, kahuna).
```

- nullstellig
- löst Backtracking aus
- kann zusammen mit Cut zum Beschreiben von Ausnahmen verwendet werden

Beispiel 5.74 („Vincent mag alle Burger außer dem Big Kahuna“)

Wissensbank

```
likes(vincent, kahuna) :- !, fail.  
likes(vincent, X) :- burger(X).
```

```
burger(bigMac).  
burger(kahuna).  
burger(royale).
```

Interpreter

```
?- likes(vincent, bigMac).  
true.  
?- likes(vincent, kahuna).  
false.
```

- nullstellig
- löst Backtracking aus
- kann zusammen mit Cut zum Beschreiben von Ausnahmen verwendet werden

Beispiel 5.74 („Vincent mag alle Burger außer dem Big Kahuna“)

Wissensbank

```
likes(vincent, kahuna) :- !, fail.  
likes(vincent, X) :- burger(X).
```

```
burger(bigMac).  
burger(kahuna).  
burger(royale).
```

Interpreter

```
?- likes(vincent, bigMac).  
true.  
?- likes(vincent, kahuna).  
false.  
?- likes(vincent, royale).
```

- nullstellig
- löst Backtracking aus
- kann zusammen mit Cut zum Beschreiben von Ausnahmen verwendet werden

Beispiel 5.74 („Vincent mag alle Burger außer dem Big Kahuna“)

Wissensbank

```
likes(vincent, kahuna) :- !, fail.  
likes(vincent, X) :- burger(X).
```

```
burger(bigMac).  
burger(kahuna).  
burger(royale).
```

Interpreter

```
?- likes(vincent, bigMac).  
true.  
?- likes(vincent, kahuna).  
false.  
?- likes(vincent, royale).  
true.
```


- nullstellig
- löst Backtracking aus
- kann zusammen mit Cut zum Beschreiben von Ausnahmen verwendet werden

Beispiel 5.74 („Vincent mag alle Burger außer dem Big Kahuna“)

Wissensbank

```
likes(vincent, kahuna) :- !, fail.  
likes(vincent, X) :- burger(X).
```

```
burger(bigMac).  
burger(kahuna).  
burger(royale).
```

Interpreter

```
?- likes(vincent, bigMac).  
true.  
?- likes(vincent, kahuna).  
false.  
?- likes(vincent, royale).  
true.  
?- likes(vincent, B).
```

- nullstellig
- löst Backtracking aus
- kann zusammen mit Cut zum Beschreiben von Ausnahmen verwendet werden

Beispiel 5.74 („Vincent mag alle Burger außer dem Big Kahuna“)

Wissensbank

```
likes(vincent, kahuna) :- !, fail.  
likes(vincent, X) :- burger(X).
```

```
burger(bigMac).  
burger(kahuna).  
burger(royale).
```

Interpreter

```
?- likes(vincent, bigMac).  
true.  
?- likes(vincent, kahuna).  
false.  
?- likes(vincent, royale).  
true.  
?- likes(vincent, B).  
false.
```

Closed World Assumption Wahr ist nur, was aus der Wissensbank folgt.

Closed World Assumption Wahr ist nur, was aus der Wissensbank folgt.

Negation as Failure Was nicht aus der Wissensbank folgt, ist falsch.

Closed World Assumption Wahr ist nur, was aus der Wissensbank folgt.

Negation as Failure Was nicht aus der Wissensbank folgt, ist falsch.

Realisierung mit Cut und fail:

```
not(Goal) :- Goal, !, fail.  
not(Goal).
```

Closed World Assumption Wahr ist nur, was aus der Wissensbank folgt.

Negation as Failure Was nicht aus der Wissensbank folgt, ist falsch.

Realisierung mit Cut und fail:

```
not(Goal) :- Goal, !, fail.  
not(Goal).
```

■ Wenn Goal aus KB folgt $\rightsquigarrow$ false.

Closed World Assumption Wahr ist nur, was aus der Wissensbank folgt.

Negation as Failure Was nicht aus der Wissensbank folgt, ist falsch.

Realisierung mit Cut und fail:

```
not(Goal) :- Goal, !, fail.  
not(Goal).
```

- Wenn Goal aus KB folgt $\rightsquigarrow$ false.
- Sonst $\rightsquigarrow$ true.

Closed World Assumption Wahr ist nur, was aus der Wissensbank folgt.

Negation as Failure Was nicht aus der Wissensbank folgt, ist falsch.

Realisierung mit Cut und fail:

```
not(Goal) :- Goal, !, fail.  
not(Goal).
```

- Wenn Goal aus KB folgt $\rightsquigarrow$ false.
- Sonst $\rightsquigarrow$ true.

Unterschiede zu logischer Negation

- $\varphi \models \neg\psi$ gdw $\varphi \wedge \psi$ **unerfüllbar** ist.

Closed World Assumption Wahr ist nur, was aus der Wissensbank folgt.

Negation as Failure Was nicht aus der Wissensbank folgt, ist falsch.

Realisierung mit Cut und fail:

```
not(Goal) :- Goal, !, fail.  
not(Goal).
```

- Wenn Goal aus KB folgt $\rightsquigarrow$ false.
- Sonst $\rightsquigarrow$ true.

Unterschiede zu logischer Negation

- $\varphi \models \neg\psi$ gdw $\varphi \wedge \psi$ **unerfüllbar** ist.
- Aus KB folgt $\text{not}(\text{Goal})$, wenn $\text{KB} \wedge \neg\text{Goal}$ **erfüllbar** ist.

Closed World Assumption Wahr ist nur, was aus der Wissensbank folgt.

Negation as Failure Was nicht aus der Wissensbank folgt, ist falsch.

Realisierung mit Cut und fail:

```
not(Goal) :- Goal, !, fail.  
not(Goal).
```

- Wenn Goal aus KB folgt $\rightsquigarrow$ false.
- Sonst $\rightsquigarrow$ true.

Unterschiede zu logischer Negation

- $\varphi \models \neg\psi$ gdw $\varphi \wedge \psi$ **unerfüllbar** ist.
- Aus KB folgt $\text{not}(\text{Goal})$, wenn $\text{KB} \wedge \neg\text{Goal}$ **erfüllbar** ist.
- Negierte Fakten sind nicht möglich (**not(happy(butch))**.)

Beispiel 5.75

Wissensbank

```
likes(vincent,X):-  
    burger(X),  
    not(X = kahuna).
```

```
burger(bigMac).  
burger(kahuna).  
burger(royale).  
not(Goal) :- Goal, !, fail.  
not(Goal).
```

Beispiel 5.75

`likes(vincent,B)`

Wissensbank

```
likes(vincent,X):-  
    burger(X),  
    not(X = kahuna).
```

```
burger(bigMac).  
burger(kahuna).  
burger(royale).  
not(Goal) :- Goal, !, fail.  
not(Goal).
```

Beispiel 5.75

```
likes(vincent,B)
```

```
| X = B
```

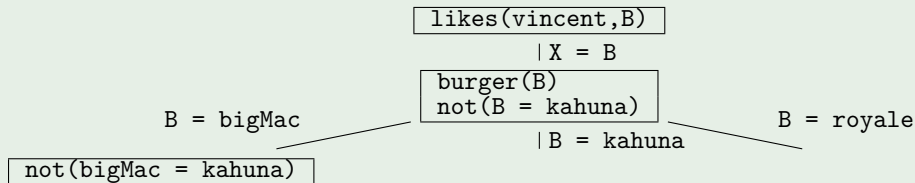
```
burger(B)  
not(B = kahuna)
```

Wissensbank

```
likes(vincent,X):-  
    burger(X),  
    not(X = kahuna).
```

```
burger(bigMac).  
burger(kahuna).  
burger(royale).  
not(Goal) :- Goal, !, fail.  
not(Goal).
```

Beispiel 5.75

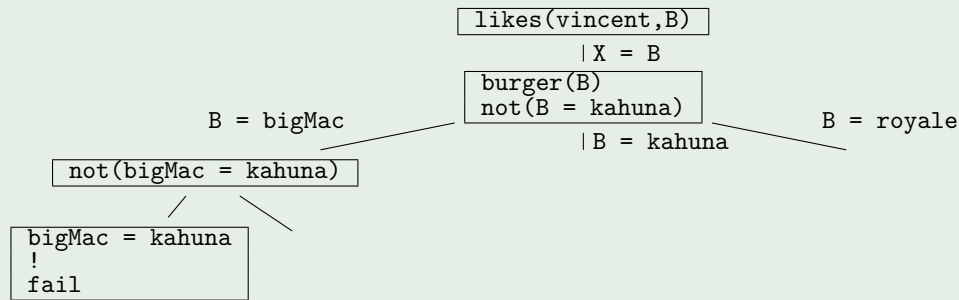


Wissensbank

```
likes(vincent,X):-
    burger(X),
    not(X = kahuna).
```

```
burger(bigMac).
burger(kahuna).
burger(royale).
not(Goal) :- Goal, !, fail.
not(Goal).
```

Beispiel 5.75

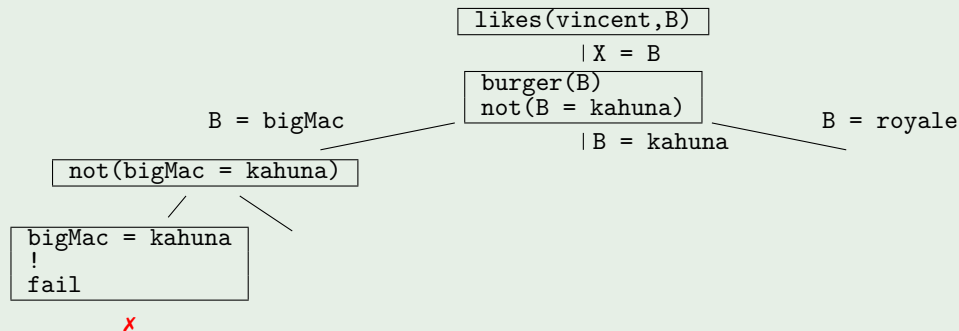


Wissensbank

```
likes(vincent, X):-  
    burger(X),  
    not(X = kahuna).
```

```
burger(bigMac).  
burger(kahuna).  
burger(royale).  
not(Goal) :- Goal, !, fail.  
not(Goal).
```

Beispiel 5.75

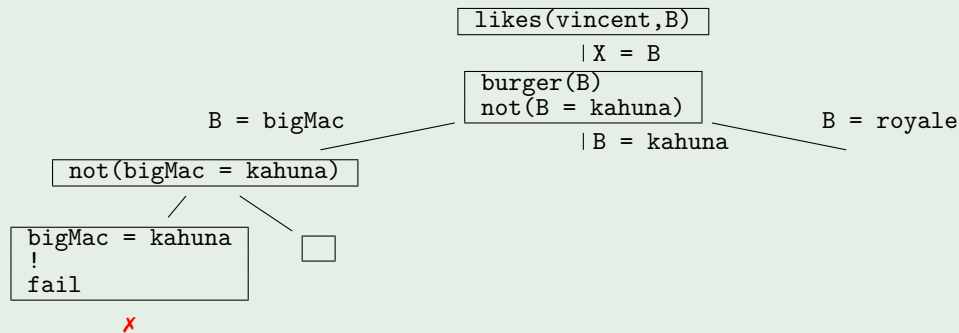


Wissensbank

```
likes(vincent, X):-  
    burger(X),  
    not(X = kahuna).
```

```
burger(bigMac).  
burger(kahuna).  
burger(royale).  
not(Goal) :- Goal, !, fail.  
not(Goal).
```


Beispiel 5.75

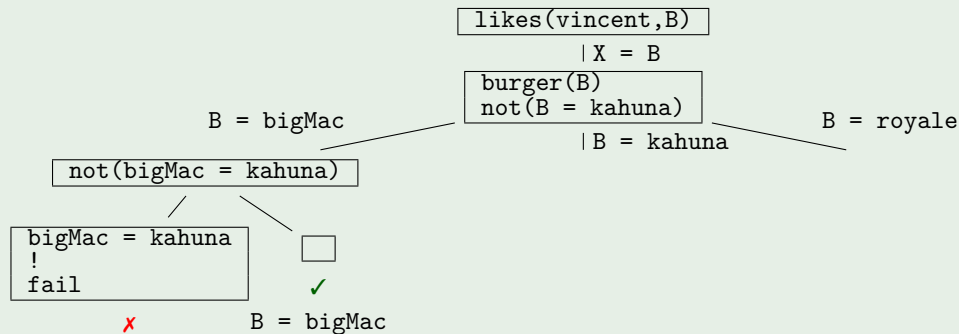


Wissensbank

```
likes(vincent, X) :-  
    burger(X),  
    not(X = kahuna).
```

```
burger(bigMac).  
burger(kahuna).  
burger(royale).  
not(Goal) :- Goal, !, fail.  
not(Goal).
```

Beispiel 5.75

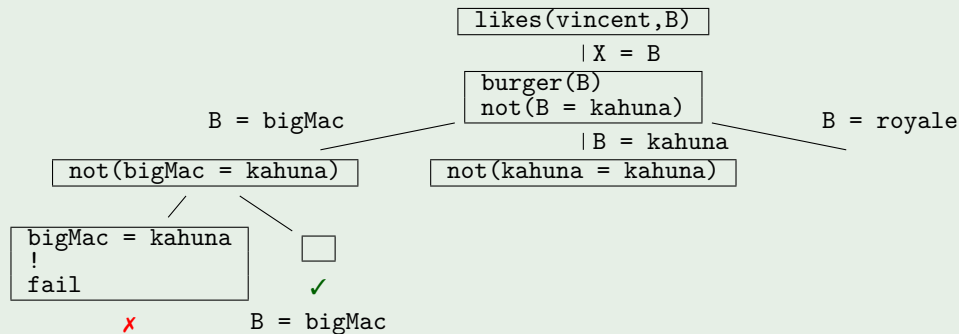


Wissensbank

```
likes(vincent, X) :-  
    burger(X),  
    not(X = kahuna).
```

```
burger(bigMac).  
burger(kahuna).  
burger(royale).  
not(Goal) :- Goal, !, fail.  
not(Goal).
```

Beispiel 5.75

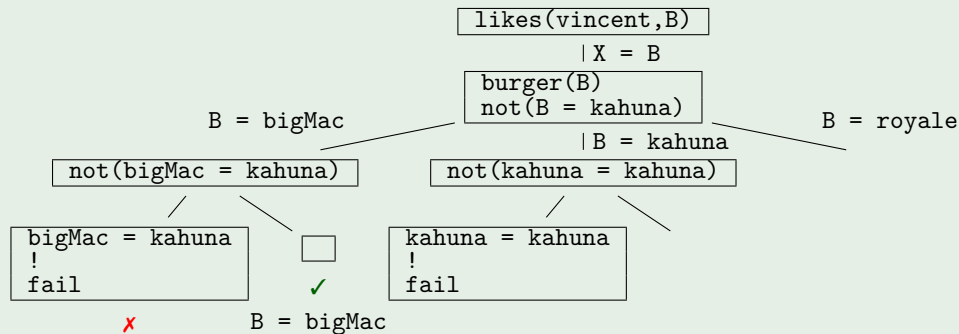


Wissensbank

```
likes(vincent,X):-
    burger(X),
    not(X = kahuna).
```

```
burger(bigMac).
burger(kahuna).
burger(royale).
not(Goal) :- Goal, !, fail.
not(Goal).
```

Beispiel 5.75

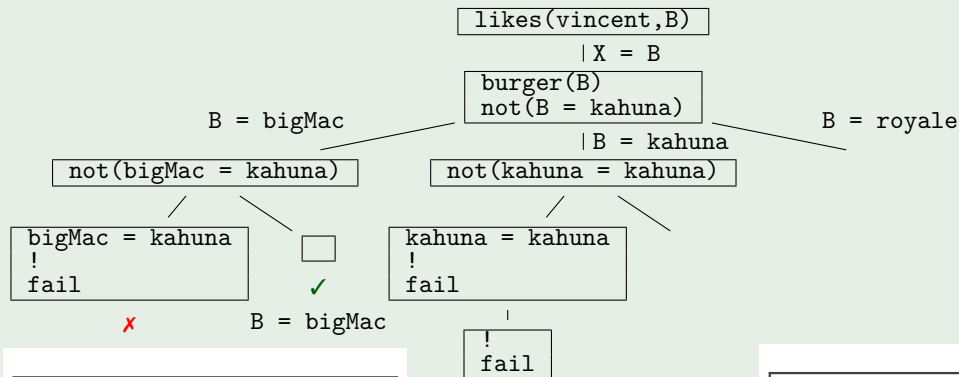


Wissensbank

```
likes(vincent,X):-
    burger(X),
    not(X = kahuna).
```

```
burger(bigMac).
burger(kahuna).
burger(royale).
not(Goal) :- Goal, !, fail.
not(Goal).
```

Beispiel 5.75

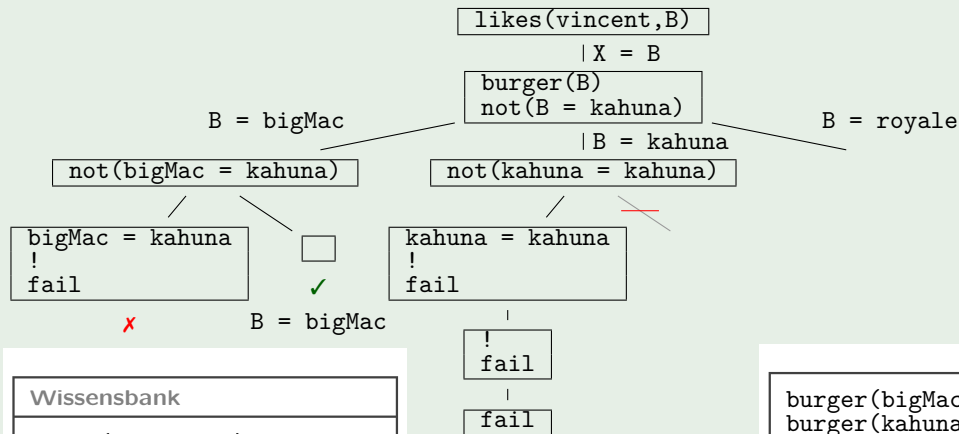


Wissensbank

```
likes(vincent,X):-
    burger(X),
    not(X = kahuna).
```

```
burger(bigMac).
burger(kahuna).
burger(royale).
not(Goal) :- Goal, !, fail.
not(Goal).
```

Beispiel 5.75

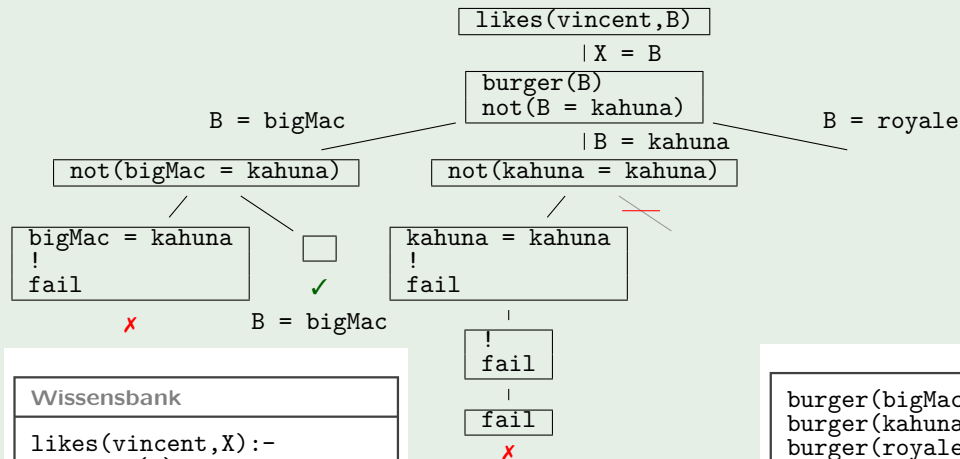


Wissensbank

```
likes(vincent, X):-  
    burger(X),  
    not(X = kahuna).
```

```
burger(bigMac).  
burger(kahuna).  
burger(royale).  
not(Goal) :- Goal, !, fail.  
not(Goal).
```

Beispiel 5.75

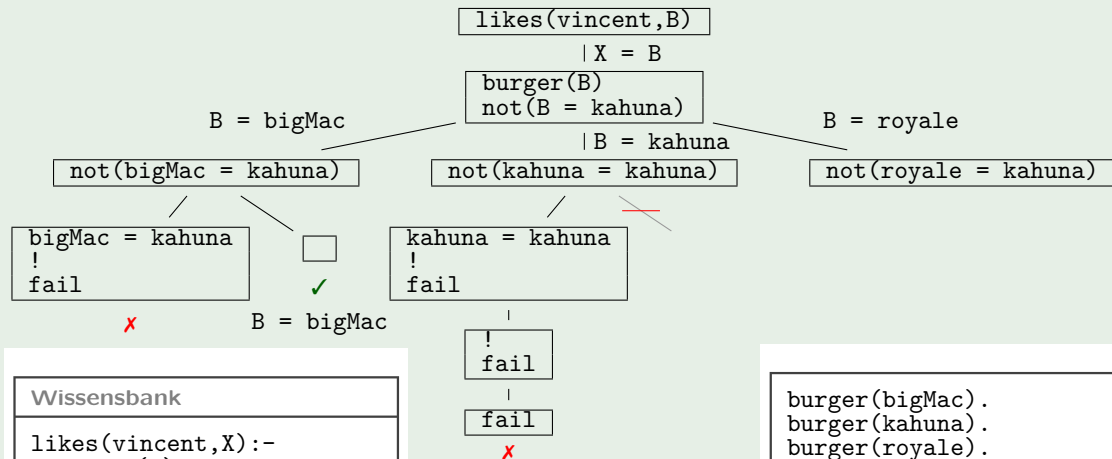


Wissensbank

```
likes(vincent, X):-  
    burger(X),  
    not(X = kahuna).
```

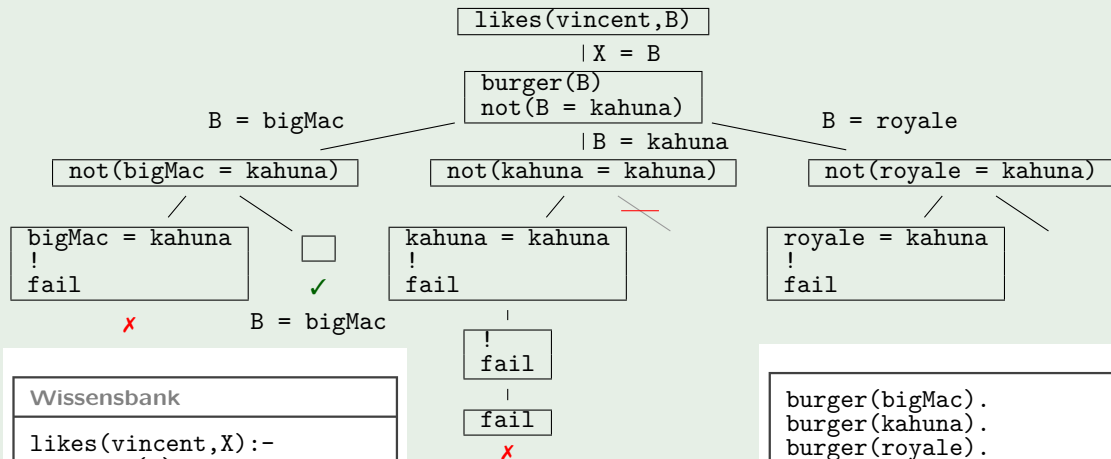
```
burger(bigMac).  
burger(kahuna).  
burger(royale).  
not(Goal) :- Goal, !, fail.  
not(Goal).
```

Beispiel 5.75



```
burger(bigMac).  
burger(kahuna).  
burger(royale).  
not(Goal) :- Goal, !, fail.  
not(Goal).
```


Beispiel 5.75

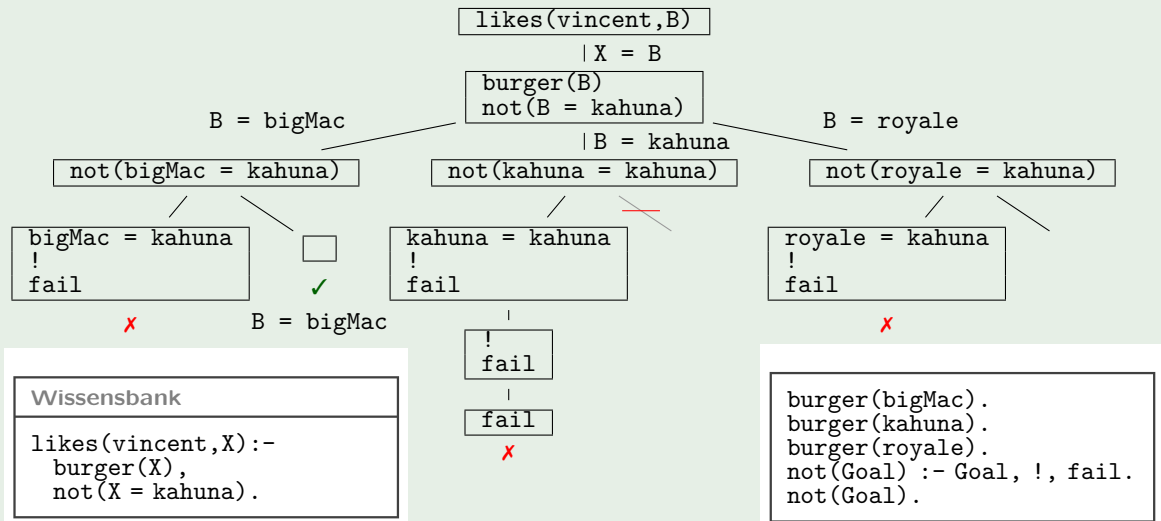


Wissensbank

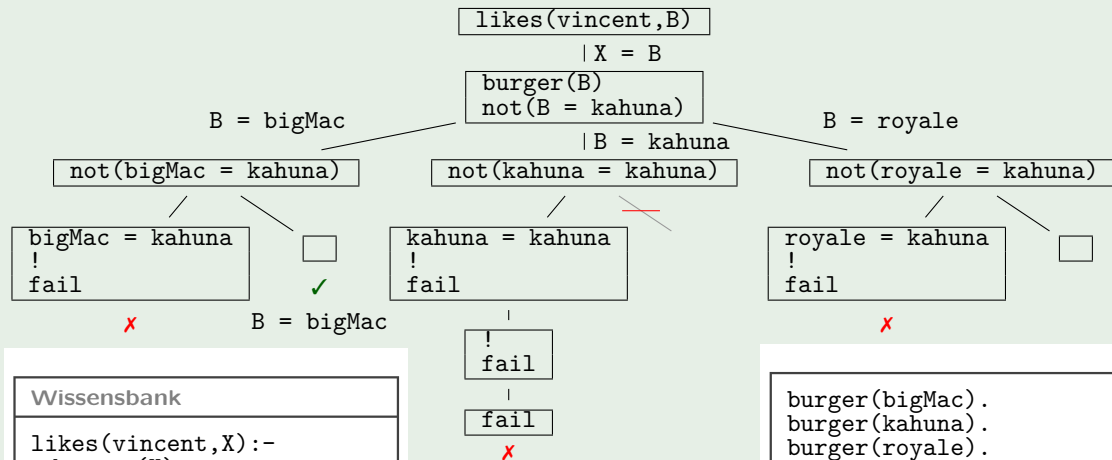
```
likes(vincent,X):-
    burger(X),
    not(X = kahuna).
```

```
burger(bigMac).
burger(kahuna).
burger(royale).
not(Goal) :- Goal, !, fail.
not(Goal).
```

Beispiel 5.75



Beispiel 5.75

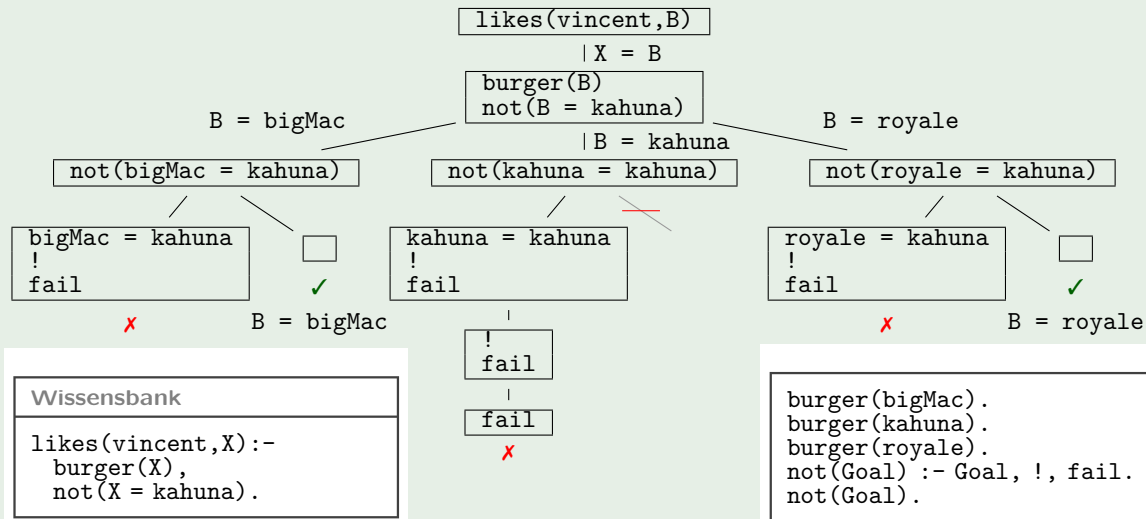


Wissensbank

```
likes(vincent, X) :-  
    burger(X),  
    not(X = kahuna).
```

```
burger(bigMac).  
burger(kahuna).  
burger(royale).  
not(Goal) :- Goal, !, fail.  
not(Goal).
```

Beispiel 5.75



Beispiel 5.76

Wissensbank

```
likes(vincent,X):- burger(X),  
    not(X = kahuna).  
burger(bigMac).  
burger(kahuna).
```

Eltern-Goal: **not**.

Beispiel 5.76

Wissensbank

```
likes(vincent,X):- burger(X),  
    not(X = kahuna).  
burger(bigMac).  
burger(kahuna).
```

Eltern-Goal: **not**.

Interpreter

```
?- likes(vincent,bigMac).  
true.
```

Beispiel 5.76

Wissensbank

```
likes(vincent,X):- burger(X),  
    not(X = kahuna).  
burger(bigMac).  
burger(kahuna).
```

Eltern-Goal: **not**.

Interpreter

```
?- likes(vincent,bigMac).  
true.  
?- likes(vincent,kahuna).  
false.
```

Beispiel 5.76

Wissensbank

```
likes(vincent,X):- burger(X),  
    not(X = kahuna).  
burger(bigMac).  
burger(kahuna).
```

Eltern-Goal: **not**.

Interpreter

```
?- likes(vincent,bigMac).  
true.  
?- likes(vincent,kahuna).  
false.  
?- likes(vincent,B).  
B = bigMac.
```


Beispiel 5.76

Wissensbank

```
likes(vincent,X):- burger(X),  
    not(X = kahuna).  
burger(bigMac).  
burger(kahuna).
```

Eltern-Goal: **not**.

Wissensbank

```
likes(vincent,kahuna):- !, fail.  
likes(vincent,X):- burger(X).  
burger(bigMac).  
burger(kahuna).
```

Eltern-Goal: **likes**.

Interpreter

```
?- likes(vincent,bigMac).  
true.  
?- likes(vincent,kahuna).  
false.  
?- likes(vincent,B).  
B = bigMac.
```

Beispiel 5.76

Wissensbank

```
likes(vincent,X):- burger(X),  
    not(X = kahuna).  
burger(bigMac).  
burger(kahuna).
```

Eltern-Goal: **not**.

Interpreter

```
?- likes(vincent,bigMac).  
true.  
?- likes(vincent,kahuna).  
false.  
?- likes(vincent,B).  
B = bigMac.
```

Wissensbank

```
likes(vincent,kahuna):- !, fail.  
likes(vincent,X):- burger(X).  
burger(bigMac).  
burger(kahuna).
```

Eltern-Goal: **likes**.

Interpreter

```
?- likes(vincent,bigMac).  
true.
```

Beispiel 5.76

Wissensbank

```
likes(vincent,X):- burger(X),  
    not(X = kahuna).  
burger(bigMac).  
burger(kahuna).
```

Eltern-Goal: **not**.

Interpreter

```
?- likes(vincent,bigMac).  
true.  
?- likes(vincent,kahuna).  
false.  
?- likes(vincent,B).  
B = bigMac.
```

Wissensbank

```
likes(vincent,kahuna):- !, fail.  
likes(vincent,X):- burger(X).  
burger(bigMac).  
burger(kahuna).
```

Eltern-Goal: **likes**.

Interpreter

```
?- likes(vincent,bigMac).  
true.  
?- likes(vincent,kahuna).  
false.
```

Beispiel 5.76

Wissensbank

```
likes(vincent,X):- burger(X),  
    not(X = kahuna).  
burger(bigMac).  
burger(kahuna).
```

Eltern-Goal: **not**.

Interpreter

```
?- likes(vincent,bigMac).  
true.  
?- likes(vincent,kahuna).  
false.  
?- likes(vincent,B).  
B = bigMac.
```

Wissensbank

```
likes(vincent,kahuna):- !, fail.  
likes(vincent,X):- burger(X).  
burger(bigMac).  
burger(kahuna).
```

Eltern-Goal: **likes**.

Interpreter

```
?- likes(vincent,bigMac).  
true.  
?- likes(vincent,kahuna).  
false.  
?- likes(vincent,B).  
false.
```

Übung 5.77

Im folgenden Programm ist im Prädikat `likes` die Reihenfolge der beiden Goals vertauscht:

```
likes(vincent,X):- not(X = kahuna), burger(X).
```

Wie beantwortet der Interpreter nun die folgende Anfrage?

```
?- likes(vincent,B).
```

Zeichnen Sie ggf. den Berechnungsbaum.

- entfernt Verzweigungspunkte

- entfernt Verzweigungspunkte
 - vom Aufruf des Eltern-Goals

- entfernt Verzweigungspunkte
 - vom Aufruf des Eltern-Goals
 - bis zum Cut

- entfernt Verzweigungspunkte
 - vom Aufruf des Eltern-Goals
 - bis zum Cut
- kann zusammen mit `fail` Ausnahmen codieren

- entfernt Verzweigungspunkte
 - vom Aufruf des Eltern-Goals
 - bis zum Cut
- kann zusammen mit `fail` Ausnahmen codieren
- Prädikat `not` beschreibt *negation as failure*

- entfernt Verzweigungspunkte
 - vom Aufruf des Eltern-Goals
 - bis zum Cut
- kann zusammen mit `fail` Ausnahmen codieren
- Prädikat `not` beschreibt *negation as failure*
 - Cut verhindert Backtracking

- entfernt Verzweigungspunkte
 - vom Aufruf des Eltern-Goals
 - bis zum Cut
- kann zusammen mit `fail` Ausnahmen codieren
- Prädikat `not` beschreibt *negation as failure*
 - Cut verhindert Backtracking
 - `fail` verursacht Scheitern des aktuellen Zweigs

- entfernt Verzweigungspunkte
 - vom Aufruf des Eltern-Goals
 - bis zum Cut
- kann zusammen mit `fail` Ausnahmen codieren
- Prädikat `not` beschreibt **negation as failure**
 - Cut verhindert Backtracking
 - `fail` verursacht Scheitern des aktuellen Zweigs
 - nicht äquivalent zu logischer Negation

- deklarativ

- deklarativ
 - Beschreibung der Domäne in der Wissensbank

- deklarativ

- Beschreibung der Domäne in der Wissensbank
- Anfragen im Interpreter

- deklarativ
 - Beschreibung der Domäne in der Wissensbank
 - Anfragen im Interpreter
- einfacher Term: Atom, Variable oder Zahl

- deklarativ
 - Beschreibung der Domäne in der Wissensbank
 - Anfragen im Interpreter
- einfacher Term: Atom, Variable oder Zahl
- komplexer Term: Funktor und Argumente

- deklarativ
 - Beschreibung der Domäne in der Wissensbank
 - Anfragen im Interpreter
- einfacher Term: Atom, Variable oder Zahl
- komplexer Term: Funktor und Argumente
- Goals, Fakten, Regeln und Anfragen

- deklarativ
 - Beschreibung der Domäne in der Wissensbank
 - Anfragen im Interpreter
- einfacher Term: Atom, Variable oder Zahl
- komplexer Term: Funktor und Argumente
- Goals, Fakten, Regeln und Anfragen
- Unifikation (=) bindet Variablen, == nicht

- deklarativ
 - Beschreibung der Domäne in der Wissensbank
 - Anfragen im Interpreter
- einfacher Term: Atom, Variable oder Zahl
- komplexer Term: Funktor und Argumente
- Goals, Fakten, Regeln und Anfragen
- Unifikation (=) bindet Variablen, == nicht
- Arithmetik: is, ==, =\=, ...

- deklarativ
 - Beschreibung der Domäne in der Wissensbank
 - Anfragen im Interpreter
- einfacher Term: Atom, Variable oder Zahl
- komplexer Term: Funktor und Argumente
- Goals, Fakten, Regeln und Anfragen
- Unifikation (=) bindet Variablen, == nicht
- Arithmetik: is, ==, =\=, ...
- Suchbaum

- deklarativ
 - Beschreibung der Domäne in der Wissensbank
 - Anfragen im Interpreter
- einfacher Term: Atom, Variable oder Zahl
- komplexer Term: Funktor und Argumente
- Goals, Fakten, Regeln und Anfragen
- Unifikation (=) bindet Variablen, == nicht
- Arithmetik: is, ==, =\=, ...
- Suchbaum
 - Bearbeite Stack von oben nach unten

- deklarativ
 - Beschreibung der Domäne in der Wissensbank
 - Anfragen im Interpreter
- einfacher Term: Atom, Variable oder Zahl
- komplexer Term: Funktor und Argumente
- Goals, Fakten, Regeln und Anfragen
- Unifikation (=) bindet Variablen, == nicht
- Arithmetik: is, ==, =\=, ...
- Suchbaum
 - Bearbeite Stack von oben nach unten
 - Unifiziere erstes Goal mit allen Regelköpfen

- deklarativ
 - Beschreibung der Domäne in der Wissensbank
 - Anfragen im Interpreter
- einfacher Term: Atom, Variable oder Zahl
- komplexer Term: Funktor und Argumente
- Goals, Fakten, Regeln und Anfragen
- Unifikation (=) bindet Variablen, == nicht
- Arithmetik: is, ==, ==\, ...
- Suchbaum
 - Bearbeite Stack von oben nach unten
 - Unifiziere erstes Goal mit allen Regelköpfen
 - Verzweigungspunkte, Backtracking

- deklarativ
 - Beschreibung der Domäne in der Wissensbank
 - Anfragen im Interpreter
- einfacher Term: Atom, Variable oder Zahl
- komplexer Term: Funktor und Argumente
- Goals, Fakten, Regeln und Anfragen
- Unifikation (=) bindet Variablen, == nicht
- Arithmetik: is, ==, =\=, ...
- Suchbaum
 - Bearbeite Stack von oben nach unten
 - Unifiziere erstes Goal mit allen Regelköpfen
 - Verzweigungspunkte, Backtracking
- Rekursion: Prädikat verwendet sich selbst; Standard-Programmierparadigma

- deklarativ
 - Beschreibung der Domäne in der Wissensbank
 - Anfragen im Interpreter
- einfacher Term: Atom, Variable oder Zahl
- komplexer Term: Funktor und Argumente
- Goals, Fakten, Regeln und Anfragen
- Unifikation (=) bindet Variablen, == nicht
- Arithmetik: is, ==, \=, ...
- Suchbaum
 - Bearbeite Stack von oben nach unten
 - Unifiziere erstes Goal mit allen Regelköpfen
 - Verzweigungspunkte, Backtracking
- Rekursion: Prädikat verwendet sich selbst; Standard-Programmierparadigma
- Listen: Standard-Datenstruktur

- deklarativ
 - Beschreibung der Domäne in der Wissensbank
 - Anfragen im Interpreter
- einfacher Term: Atom, Variable oder Zahl
- komplexer Term: Funktor und Argumente
- Goals, Fakten, Regeln und Anfragen
- Unifikation (=) bindet Variablen, == nicht
- Arithmetik: is, ==, \=, ...
- Suchbaum
 - Bearbeite Stack von oben nach unten
 - Unifiziere erstes Goal mit allen Regelköpfen
 - Verzweigungspunkte, Backtracking
- Rekursion: Prädikat verwendet sich selbst; Standard-Programmierparadigma
- Listen: Standard-Datenstruktur
- Cut verhindert Backtracking