

Logik und Grundlagen der Informatik

Stephan Schulz

stephan.schulz@dhbw-stuttgart.de



```
(define (fak x)
  (if (= x 0)
      1
      (* x (fak (- x 1)))
  )
)
```

$$(0 \in S \wedge \forall n \in \mathbb{N} : (n \in S \rightarrow n + 1 \in S)) \rightarrow \mathbb{N} \subseteq S$$

Inhaltsverzeichnis

Einführung

Mengenlehre

 Zahlenkonstruktion und Termalgebra

 Mengenoperationen

 Kartesische Produkte, Potenzmengen

 Relationen

 Funktionen

Funktionales Programmieren mit Scheme

Formale Logik

 Aussagenlogik

 Prädikatenlogik

Anhang: Kurzübersicht Scheme

Kursspezifisches Material

 TINF14B

 TINF14C

▶ Stephan Schulz

- ▶ Dipl.-Inform., U. Kaiserslautern, 1995
- ▶ Dr. rer. nat., TU München, 2000
- ▶ Visiting professor, U. Miami, 2002
- ▶ Visiting professor, U. West Indies, 2005
- ▶ Gastdozent (Hildesheim, Offenburg, . . .) seit 2009
- ▶ Praktische Erfahrung: Entwicklung von Flugsicherungssystemen
 - ▶ System engineer, 2005
 - ▶ Project manager, 2007
 - ▶ Product Manager, 2013
- ▶ Professor, DHBW Stuttgart, 2014

- ▶ Stephan Schulz
 - ▶ Dipl.-Inform., U. Kaiserslautern, 1995
 - ▶ Dr. rer. nat., TU München, 2000
 - ▶ Visiting professor, U. Miami, 2002
 - ▶ Visiting professor, U. West Indies, 2005
 - ▶ Gastdozent (Hildesheim, Offenburg, . . .) seit 2009
 - ▶ Praktische Erfahrung: Entwicklung von Flugsicherungssystemen
 - ▶ System engineer, 2005
 - ▶ Project manager, 2007
 - ▶ Product Manager, 2013
 - ▶ Professor, DHBW Stuttgart, 2014

Research: Logik & Deduktion

Rechnerausstattung

- ▶ Für die praktischen Übungen brauchen Sie mindestens einen Text-Editor und einen Scheme-Interpreter. Die notwendigen Programme sind mindestens für Linux und Mac OS-X frei verfügbar. Sie können eine aktuelle Linux-Installation auch unter einer virtuellen Maschine installieren.
 - ▶ Z.B. VirtualBox (<https://www.virtualbox.org>)
 - ▶ Betriebssystem: Z.B.. Ubuntu (<http://www.ubuntu.com>)
 - ▶ Paketmanager für OS-X: Fink (<http://fink.sourceforge.net>) oder MacPorts (<https://www.macports.org>)
- ▶ Scheme Interpreter: Guile
 - ▶ Unter Ubuntu/Debian: `sudo apt-get install guile-1.8`
 - ▶ Unter OS-X: `sudo fink install guile/sudo port install guile`
- ▶ Text-Editor:
 - ▶ Klassisch: `vi/vim`
 - ▶ Unschlagbar: `emacs`
 - ▶ Freundlich/modern: `gedit`
 - ▶ Auf OS-X installiert: `TextEdit`

Real Programmers

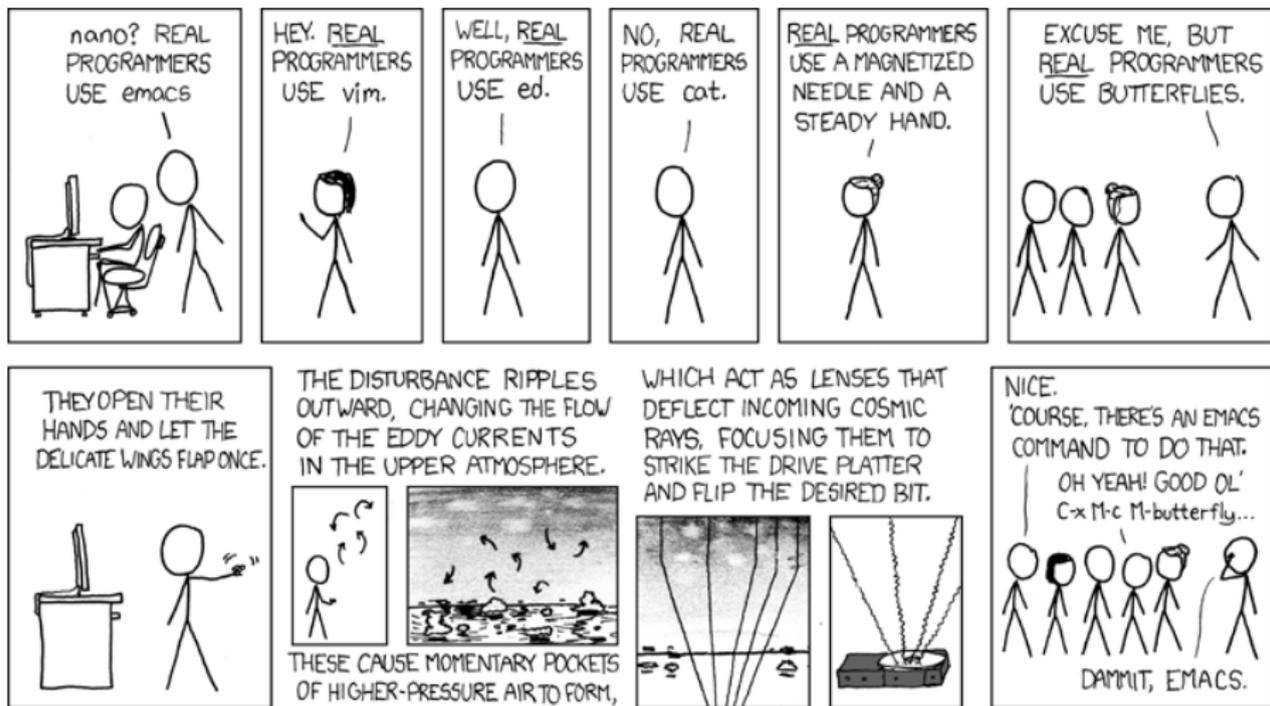


Image credit: <http://xkcd.com/378/>

Semesterübersicht

- ▶ Mengenlehre
 - ▶ Mengenbegriff und Operationen
 - ▶ Relationen, Funktionen, Ordnungen, . . .
- ▶ Aussagenlogik
 - ▶ Syntax und Semantik
 - ▶ Formalisierungsbeispiele
 - ▶ Kalküle
- ▶ Prädikatenlogik
 - ▶ Syntax und Semantik
 - ▶ Formalisierungsbeispiele/Korrektheit von Programmen
 - ▶ Kalküle
- ▶ Nichtprozedurale Programmiermodelle
 - ▶ Funktional: [Scheme](#)
 - ▶ (Logisch/Deklarativ: [Prolog](#))

- ▶ Allgemein
 - ▶ Dirk W. Hoffmann: [Theoretische Informatik](#)
 - ▶ Karl Stroetmann: [Theoretische Informatik I - Logik und Mengenlehre](#) (Skript der Vorlesung 2012/2013 an der DHBW), <http://www.lehre.dhbw-stuttgart.de/~stroetma/Logic/logik-2013.pdf>
- ▶ Interessante Klassiker
 - ▶ Bertrand Russel: [Introduction to Mathematical Philosophy](#) (1918), <http://www.gutenberg.org/ebooks/41654>
 - ▶ Raymond M. Smullyan: [First-Order Logic](#) (1968)
- ▶ Scheme
 - ▶ Kelsey, Clinger, Rees (editors): [Revised⁵ Report on the Algorithmic Language Scheme](#), <http://www.schemers.org/Documents/Standards/R5RS/r5rs.pdf>
 - ▶ G. J. Sussman and H. Abelson: [Structure and Interpretation of Computer Programs](#), Volltext unter CC BY-NC 3.0: <http://mitpress.mit.edu/sicp/>

Ziele der Vorlesung

- ▶ Vokabular (These von Sapir–Whorf)
 - ▶ Wie spricht man über Argumente?
 - ▶ Wie beschreibt man Algorithmen, Datenstrukturen und Programme abstrakt?
 - ▶ Was sind Syntax, Semantik, Interpretation, Modell, Wahrheit, Gültigkeit, Ableitbarkeit?
- ▶ Methodenkompetenz in
 - ▶ Modellierung von Systemen mit abstrakten Werkzeugen
 - ▶ Anwendungen von Logik und Deduktion
 - ▶ Argumentieren über Logik und Deduktion
 - ▶ Argumentieren über Programme und ihr Verhalten
 - ▶ Programmieren in Scheme/Prolog



Übung: Das MIU-Rätsel

- ▶ Wir betrachten ein formales System mit den folgenden Regeln:
 1. Alle Worte bestehen aus den Buchstaben M, I und U
 2. Wenn ein Wort mit I endet, darf man U anhängen
 3. III darf durch U ersetzt werden
 4. UU darf entfernt werden
 5. Das Teilwort nach einem M darf verdoppelt werden
 6. Eine Ableitung beginnt immer mit MI

Übung: Das MIU-Rätsel

- ▶ Wir betrachten ein formales System mit den folgenden Regeln:
 1. Alle Worte bestehen aus den Buchstaben M, I und U
 2. Wenn ein Wort mit I endet, darf man U anhängen
 3. III darf durch U ersetzt werden
 4. UU darf entfernt werden
 5. Das Teilwort nach einem M darf verdoppelt werden
 6. Eine Ableitung beginnt immer mit MI

Nach Douglas R. Hofstadter, *Gödel, Escher, Bach: ein Endloses Geflochtenes Band*

Übung: Das MIU-Rätsel (Formaler)

- ▶ Alle Worte bestehen aus den Buchstaben M, I und U
- ▶ x und y stehen für beliebige (Teil-)worte
- ▶ Eine Ableitung beginnt immer mit MI
- ▶ Ableitungsregeln:
 1. $xI \rightarrow xIU$
 2. $xIIIIy \rightarrow xUy$
 3. $xUUy \rightarrow xy$
 4. $Mx \rightarrow Mxx$
- ▶ Wir schreiben $x \vdash y$, wenn x durch eine Anwendung einer Regel in y überführen läßt

Übung: Das MIU-Rätsel (Formaler)

- ▶ Alle Worte bestehen aus den Buchstaben M, I und U
- ▶ x und y stehen für beliebige (Teil-)worte
- ▶ Eine Ableitung beginnt immer mit MI
- ▶ Ableitungsregeln:
 1. $xI \rightarrow xIU$
 2. $xIIIIy \rightarrow xUy$
 3. $xUUy \rightarrow xy$
 4. $Mx \rightarrow Mxx$
- ▶ Wir schreiben $x \vdash y$, wenn x durch eine Anwendung einer Regel in y überführen läßt
 - ▶ Beispiel: $MI \vdash_4 MII \vdash_4 MIIII \vdash_2 MUI \vdash_4 MUIUI$

Übung: Das MIU-Rätsel (Formaler)

- ▶ Alle Worte bestehen aus den Buchstaben M, I und U
- ▶ x und y stehen für beliebige (Teil-)worte
- ▶ Eine Ableitung beginnt immer mit MI
- ▶ Ableitungsregeln:
 1. $xI \rightarrow xIU$
 2. $xIIIIy \rightarrow xUy$
 3. $xUUy \rightarrow xy$
 4. $Mx \rightarrow Mxx$
- ▶ Wir schreiben $x \vdash y$, wenn x durch eine Anwendung einer Regel in y überführen läßt
 - ▶ Beispiel: $MI \vdash_4 MII \vdash_4 MIIII \vdash_2 MUI \vdash_4 MUIUI$
- ▶ Aufgabe: Geben Sie für die folgenden Worte Ableitungen an:
 1. MUIU
 2. MIIIII
 3. MUUUI
 4. MU

MIU Lösungen (1)

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

Lösungen

MIU Lösungen (1)

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

Lösungen

1. $MI \vdash MII \vdash MIIII \vdash MIIIIU \vdash MUIU$

MIU Lösungen (1)

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

Lösungen

1. $MI \vdash MII \vdash MIIII \vdash MIIIIU \vdash MUIU$
2. $MI \vdash MII \vdash MIIII \vdash MIIIIIIII \vdash MIIIIIIIIU \vdash MIIIIIIUU \vdash MIIIIII$

MIU Lösungen (1)

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

Lösungen

1. $MI \vdash MII \vdash MIIII \vdash MIIIIU \vdash MUIU$
2. $MI \vdash MII \vdash MIIII \vdash MIIIIIIII \vdash MIIIIIIIIU \vdash MIIIIIIUU \vdash MIIIIII$
3. $MI \vdash MIIII \vdash MIIIIIIII \vdash MUIIIII \vdash MUUII \vdash MUUIIUUII \vdash MUUIIIII \vdash MUUII$

MIU Lösungen (2)

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

Lösungen

MIU Lösungen (2)

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

Lösungen

4. MU kann nicht hergeleitet werden!

MIU Lösungen (2)

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

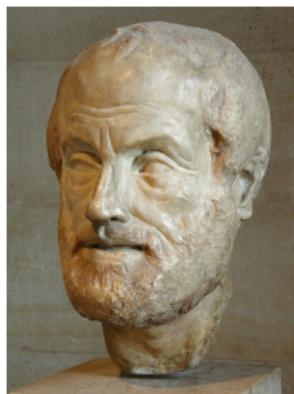
Lösungen

4. MU kann nicht hergeleitet werden!

- ▶ Beweis: Betrachte Anzahl der I in ableitbaren Worten
 - ▶ Es gilt die **Invariante**, dass $MI \vdash^* x$ impliziert, dass $|x|_I$ nicht glatt durch 3 teilbar ist
 - ▶ $|MI|_I = 1 \bmod 3 = 1$
 - ▶ Regeln 1 und 3 ändern die Anzahl der I nicht
 - ▶ Regel 4 verdoppelt die Anzahl der I
 - ▶ Regel 2 reduziert die Anzahl der I um 3
- ▶ In keinem der Fälle wird aus einem nicht-Vielfachen von 3 ein Vielfaches von 3. Aber $|MU|_I = 0$ und $0 \bmod 3 = 0$. Also kann MU nicht herleitbar sein.

**Vokabular
Methoden**

- ▶ Ziel
 - ▶ Formalisierung rationalen Denkens
 - ▶ Ursprünge: Aristoteles („Organon“, „De Interpretatione“), Al-Farabi, Boole, Frege, Russel&Whitehead („Principia Mathematica“), Gödel (Vollständigkeit und Unvollständigkeit), Davis ...
- ▶ Rolle der Logik in der Informatik
 - ▶ Grundlagen der Informatik und der Mathematik: Axiomatische Mengenlehre, Boolesche Schaltkreise
 - ▶ Anwendung innerhalb der Informatik: Spezifikation, Programmentwicklung, Programmverifikation
 - ▶ Werkzeug für Anwendungen der Informatik außerhalb der Informatik: Künstliche Intelligenz, Wissensrepräsentation



- ▶ Automatisierung rationalen Denkens
 - ▶ Eindeutige Spezifikationen
 - ▶ Syntax (was kann ich aufschreiben?)
 - ▶ Semantik (was bedeutet das geschriebene?)
 - ▶ Objektiv richtige Ableitung von neuem Wissen
 - ▶ Was bedeutet „Richtigkeit“?
 - ▶ Kann man Richtigkeit automatisch sicherstellen?
 - ▶ Kann man neue Fakten automatisch herleiten?

$$\frac{\forall \vdash \implies \exists}{\neg \exists \vdash}$$

Anwendungsbeispiel: Äquivalenz von Spezifikationen

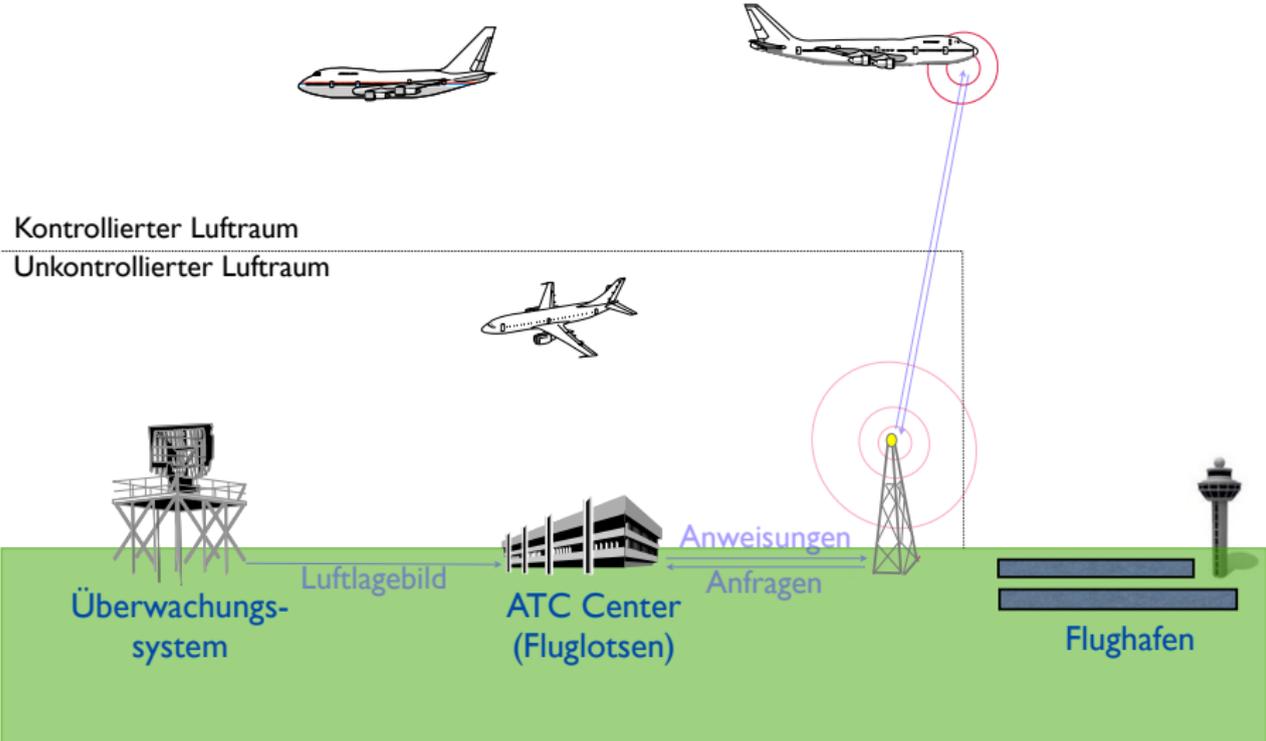
- ▶ Entwicklungsprozess (Auszug):
 - ▶ Kundenspezifikation
 - ▶ Systemspezifikation
 - ▶ Software-Design
 - ▶ Implementierung
- ▶ Problem: Äquivalenz der verschiedenen Ebenen
 - ▶ Manuelle Überprüfung aufwendig und fehleranfällig
 - ▶ Äquivalenz nicht immer offensichtlich
 - ▶ Ausgangsspezifikation nicht immer zur direkten Umsetzung geeignet
- ▶ Deduktionsmethoden können diesen Prozess unterstützen

Anwendungsbeispiel: Äquivalenz von Spezifikationen

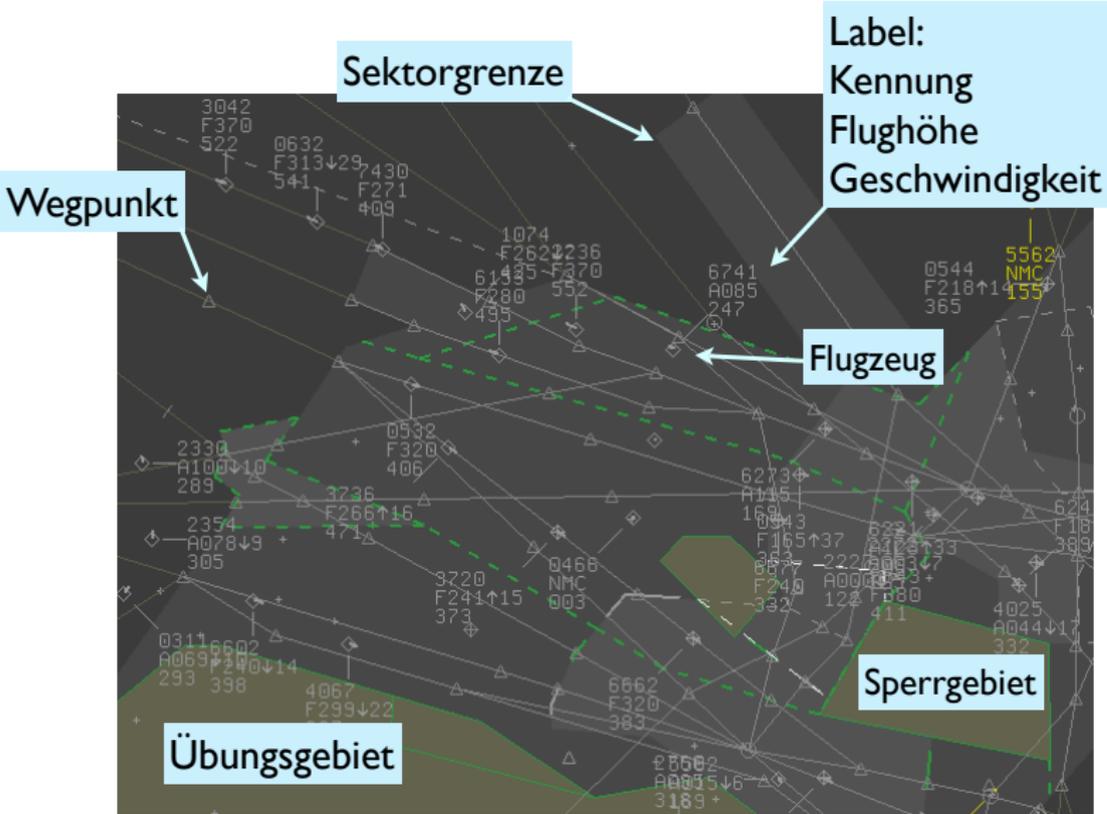
- ▶ Entwicklungsprozess (Auszug):
 - ▶ Kundenspezifikation
 - ▶ Systemspezifikation
 - ▶ Software-Design
 - ▶ Implementierung
- ▶ Problem: Äquivalenz der verschiedenen Ebenen
 - ▶ Manuelle Überprüfung aufwendig und fehleranfällig
 - ▶ Äquivalenz nicht immer offensichtlich
 - ▶ Ausgangsspezifikation nicht immer zur direkten Umsetzung geeignet
- ▶ Deduktionsmethoden können diesen Prozess unterstützen

Geht das auch konkreter?

Beispiel: Flugsicherung



Luftlagebild



Filterung vermeidet Überlastung

- ▶ Ziel: Fluglotse sieht nur relevanten Verkehr
- ▶ Beispiel:
 - ▶ Nur Flugzeuge im kontrollierten Luftraum
 - ▶ Alle Flugzeuge in der Nähe eines Flughafens
 - ▶ Ansonsten: Flugzeuge ab Flughöhe FL 100
 - ▶ Ausnahme: Militärflugzeuge im Übungsgebiet

Sehr variable Forderungen der Lotsen



Hart-kodierte Lösungen ungeeignet

- ▶ Filter wird durch logischen Ausdruck beschrieben
- ▶ Elementarfilter (Auswertung nach Luftlage):
 - ▶ Höhenband
 - ▶ Id-Code (Mode-3/A) in Liste
 - ▶ Geographische Filter (Polygon)
- ▶ Kombinationen durch logische Operatoren

Flugzeug wird genau dann angezeigt, wenn der Filterausdruck zu „wahr“ ausgewertet wird

Realisierung im Luftraum UAE (Beispiel)

- ▶ Generische Filter:
 - ▶ $\text{inregion}(X, \langle \text{polygon} \rangle)$: X ist im beschriebenen Polygon
 - ▶ $\text{altitude}(X, \text{lower}, \text{upper})$: Flughöhe (in FL) von X ist zwischen lower (einschließlich) und upper (ausschließlich)
 - ▶ $\text{modeA}(X, \langle \text{code-list} \rangle)$: Kennung von X ist in gegebener Liste
- ▶ Spezialisierte Einzelfilter:
 - ▶ $\forall X : \text{a-d-app} \leftrightarrow \text{inregion}(X, [\text{abu-dhabi-koord}])$
 - ▶ $\forall X : \text{dub-app} \leftrightarrow \text{inregion}(X, [\text{dubai-koord}])$
 - ▶ $\forall X : \text{milregion} \leftrightarrow \text{inregion}(X, [\text{training-koord}])$
 - ▶ $\forall X : \text{lowairspace} \leftrightarrow \text{altitude}(X, 0, 100)$
 - ▶ $\forall X : \text{uppairspace} \leftrightarrow \text{altitude}(X, 100, 900)$
 - ▶ $\forall X : \text{military}(X) \leftrightarrow \text{modeA}(X, [\text{mil-code-list}])$

- ▶ Filterlösung: Stelle Flugzeug X genau dann da, wenn

$$\begin{aligned} & ((a-d-app(X) \wedge lowairspace(X)) \\ \vee & \quad (dub-app(X) \wedge lowairspace(X)) \\ \vee & \quad \quad \quad uppairspace(X)) \\ \wedge & \quad (\neg milregion(X) \vee \neg military(X)) \end{aligned}$$

- ▶ mit den gegebenen Definitionen und der durch die aktuelle Luftlage definierte **Interpretation** zu „wahr“ evaluiert wird.

- ▶ Implementierungsdetails:
 - ▶ Höhenfilter sind billig (2 Vergleiche)
 - ▶ ID-Filter: Zugriff auf große Tabelle
 - ▶ Geographische Filter: Teuer, sphärische Geometrie
 - ▶ Positiv: Kurzschlussauswertung der Booleschen Operatoren
 - ▶ Auswertung des zweiten Arguments nur, wenn notwendig
- ▶ Optimierte Version:

$$\begin{aligned} & ((\text{uppairspace}(X) \\ \vee & \quad \text{dub-app}(X) \\ \vee & \quad \text{a-d-app}(X)) \\ \wedge & \quad (\neg \text{military}(X) \vee \neg \text{milregion}(X))) \end{aligned}$$

Äquivalenz?

$$\begin{aligned} & ((a\text{-d-app}(X) \wedge \text{lowairspace}(X)) \\ \vee & (\text{dub-app}(X) \wedge \text{lowairspace}(X)) \\ \vee & \text{uppairspace}(X)) \\ \wedge & (\neg \text{milregion}(X) \vee \neg \text{military}(X)) \end{aligned}$$

gegen

$$\begin{aligned} & ((\text{uppairspace}(X) \\ \vee & \text{dub-app}(X) \\ \vee & \text{a-d-app}(X)) \\ \wedge & (\neg \text{military}(X) \vee \neg \text{milregion}(X))) \end{aligned}$$

Formalisierung in TPTP-Syntax

```
fof(filter_equiv , conjecture , (  
% Naive version: Display aircraft in the Abu Dhabi Approach area  
% lower airspace, display aircraft in the Dubai Approach area in  
% airspace, display all aircraft in upper airspace, except for  
% aircraft in military training region if they are actual milit  
% aircraft.  
    (![X]:((( a_d_app(X) & lowairspace(X)  
              |(dub_app(X) & lowairspace(X))  
              |uppairspace(X))  
            & (~milregion(X)|~military(X))))  
=>  
% Optimized version: Display all aircraft in either Approach, d  
% aircraft in upper airspace, except military aircraft in the n  
% training region  
    (![X]:((uppairspace(X) | dub_app(X) | a_d_app(X)) &  
            (~military(X) | ~milregion(X))))).
```

- ▶ Frage: Sind ursprüngliche und optimierte Version äquivalent?

- ▶ Frage: Sind ursprüngliche und optimierte Version äquivalent?

```
# Initializing proof state
```

```
# Scanning for AC axioms
```

```
...
```

```
# No proof found!
```

```
# SZS status CounterSatisfiable
```

- ▶ Automatischer Beweisversuch schlägt fehl (nach 1664 Schritten/0.04 s)

- ▶ Frage: Sind ursprüngliche und optimierte Version äquivalent?

```
# Initializing proof state
```

```
# Scanning for AC axioms
```

```
...
```

```
# No proof found!
```

```
# SZS status CounterSatisfiable
```

- ▶ Automatischer Beweisversuch schlägt fehl (nach 1664 Schritten/0.04s)
- ▶ Analyse: $\text{lowairspace}(X)$ oder $\text{uppairspace}(X)$ sind die einzigen Möglichkeiten - aber das ist nicht spezifiziert!

Formalisierung in TPTP-Syntax

```
% All aircraft are either in lower or in upper airspace
fof(low_up_is_exhaustive, axiom,
    (![X]:(lowairspace(X)|uppairspace(X)))).
```

```
fof(filter_equiv, conjecture, (
% Naive version: Display aircraft in the Abu Dhabi Approach area i
% lower airspace, display aircraft in the Dubai Approach area i
% airspace, display all aircraft in upper airspace, except for
% aircraft in military training region if they are actual milit
% aircraft.
```

```
    (![X]:(((a_d_app(X) & lowairspace(X))
              |(dub_app(X) & lowairspace(X))
              |uppairspace(X))
          & (~milregion(X)|~military(X))))
```

\Leftrightarrow

```
% Optimized version: Display all aircraft in either Approach, d
% aircraft in upper airspace, except military aircraft in the n
% training region
```

```
    (![X]:((uppairspace(X) | dub_app(X) | a_d_app(X)) &
          (~military(X) | ~milregion(X))))).
```

- ▶ Frage: Sind ursprüngliche und optimierte Version äquivalent?

```
# Initializing proof state
```

```
# Scanning for AC axioms
```

```
...
```

```
# Proof found!
```

```
# SZS status Theorem
```

- ▶ Mit ergänzter Spezifikation ist automatischer Beweisversuch erfolgreich (229 Schritte/0.038 s)

- ▶ Problem: Flexible Filterspezifikation mit klarer Semantik für Darstellung von Flugzeugen
- ▶ Lösung: Spezifikation mit symbolischer Logik
 - ▶ Mächtig
 - ▶ Dynamisch konfigurierbar
 - ▶ Gut verstandene Semantik
 - ▶ Automatische Verifikation möglich

Hausaufgabe

- ▶ Installieren Sie auf Ihrem Laptop eine Umgebung, in der Sie Scheme-Programme entwickeln und ausführen können.
- ▶ Orientieren Sie sich dabei an Seite 4.

Zurück B

Zurück C

Mengenlehre

Eine **Definition** ist eine genaue Beschreibung eines Objektes oder Konzepts.

- ▶ Definitionen können einfach oder komplex sein
- ▶ Definitionen müssen präzise sein - es muss klar sein, welche Objekte oder Konzepte beschrieben werden
- ▶ Oft steckt hinter einer Definition eine Intuition - die Definition versucht, ein „reales“ Konzept formal zu beschreiben
 - ▶ Hilfreich für das Verständnis - aber gefährlich! Nur die Definition an sich zählt für formale Argumente

Definition: Ein Beweis ist ein Argument, das einen verständigen und unvoreingenommenen Empfänger von der unbestreitbaren Wahrheit einer Aussage überzeugt.

- ▶ Oft mindestens semi-formal
- ▶ Aussage ist fast immer ein Konditional (d.h. eine bedingte Aussage)

Definition: Ein Beweis ist ein Argument, das einen verständigen und unvoreingenommenen Empfänger von der unbestreitbaren Wahrheit einer Aussage überzeugt.

- ▶ Oft mindestens semi-formal
- ▶ Aussage ist fast immer ein Konditional (d.h. eine bedingte Aussage)
 - ▶ ... aber die Annahmen sind für semi-formale Beweise oft implizit

Definition: Ein Beweis ist ein Argument, das einen verständigen und unvoreingenommenen Empfänger von der unbestreitbaren Wahrheit einer Aussage überzeugt.

- ▶ Oft mindestens semi-formal
- ▶ Aussage ist fast immer ein Konditional (d.h. eine bedingte Aussage)
 - ▶ ... aber die Annahmen sind für semi-formale Beweise oft implizit
 - ▶ Z.B. Eigenschaften von natürlichen Zahlen, Bedeutung von Symbolen, ...

Mengenbegriff von Georg Cantor



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, 1895

Definition: Eine Menge ist eine Sammlung von Objekten, betrachtet als Einheit

- ▶ Die Objekte heißen auch Elemente der Menge. Elemente können beliebige Objekte sein:
- ▶ Zahlen
- ▶ Worte
- ▶ Andere Mengen (!)
- ▶ Listen, Paare, Funktionen, ...
- ▶ ... aber auch Personen, Fahrzeuge, Kurse an der DHBW, ...

Definition: Eine Menge ist eine Sammlung von Objekten, betrachtet als Einheit

- ▶ Die Objekte heißen auch Elemente der Menge. Elemente können beliebige Objekte sein:
- ▶ Zahlen
- ▶ Worte
- ▶ Andere Mengen (!)
- ▶ Listen, Paare, Funktionen, ...
- ▶ ... aber auch Personen, Fahrzeuge, Kurse an der DHBW, ...

Die Menge **aller** im Moment betrachteten Objekte heißt manchmal **Universum**, **Bereich** oder **(universelle) Trägermenge**. Dabei ist etwas Vorsicht notwendig (mehr später).

Definition von Mengen

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

Basiseigenschaften von Mengen

- ▶ Mengen sind ungeordnet
 - ▶ $\{a, b, c\} = \{b, c, a\} = \{c, a, b\}$
 - ▶ Geordnet sind z.B. [Listen](#)
 - ▶ Aber: Wir können eine externe Ordnung zu einer Menge definieren (später)!
- ▶ Jedes Element kommt in einer Menge maximal einmal vor
 - ▶ $\{1, 1, 1\}$ hat ein Element
 - ▶ Mehrfaches Vorkommen des gleichen Elements erlauben z.B. [Multimengen](#)

Teilmengen, Mengengleichheit

Definition: Eine Menge M_1 heißt **Teilmenge** von M_2 , wenn für alle $x \in M_1$ auch $x \in M_2$ gilt.

- ▶ Schreibweise: $M_1 \subseteq M_2$

Teilmengen, Mengengleichheit

Definition: Eine Menge M_1 heißt **Teilmenge** von M_2 , wenn für alle $x \in M_1$ auch $x \in M_2$ gilt.

► Schreibweise: $M_1 \subseteq M_2$

Definition: Zwei Mengen M_1 und M_2 sind einander gleich, wenn sie die selben Elemente enthalten. Formal: Für alle Elemente x gilt: $x \in M_1$ gdw. $x \in M_2$.

► Schreibweise: $M_1 = M_2$

Teilmengen, Mengengleichheit

Definition: Eine Menge M_1 heißt **Teilmenge** von M_2 , wenn für alle $x \in M_1$ auch $x \in M_2$ gilt.

► Schreibweise: $M_1 \subseteq M_2$

Definition: Zwei Mengen M_1 und M_2 sind einander gleich, wenn sie die selben Elemente enthalten. Formal: Für alle Elemente x gilt: $x \in M_1$ gdw. $x \in M_2$.

► Schreibweise: $M_1 = M_2$

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

Teilmengen, Mengengleichheit

Definition: Eine Menge M_1 heißt **Teilmenge** von M_2 , wenn für alle $x \in M_1$ auch $x \in M_2$ gilt.

► Schreibweise: $M_1 \subseteq M_2$

Definition: Zwei Mengen M_1 und M_2 sind einander gleich, wenn sie die selben Elemente enthalten. Formal: Für alle Elemente x gilt: $x \in M_1$ gdw. $x \in M_2$.

► Schreibweise: $M_1 = M_2$

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“

Echte Teilmengen, Obermengen

Definition: Eine Menge M_1 heißt **echte Teilmenge** von M_2 , wenn $M_1 \subseteq M_2$ und $M_1 \neq M_2$.

- ▶ Schreibweise: $M_1 \subset M_2$

Definition: Eine Menge M_1 heißt **echte Teilmenge** von M_2 , wenn $M_1 \subseteq M_2$ und $M_1 \neq M_2$.

- ▶ Schreibweise: $M_1 \subset M_2$
- ▶ Analog definiere 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: Eine Menge M_1 heißt **echte Teilmenge** von M_2 , wenn $M_1 \subseteq M_2$ und $M_1 \neq M_2$.

- ▶ Schreibweise: $M_1 \subset M_2$
- ▶ Analog definiere 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.

Echte Teilmengen, Obermengen

Definition: Eine Menge M_1 heißt **echte Teilmenge** von M_2 , wenn $M_1 \subseteq M_2$ und $M_1 \neq M_2$.

- ▶ Schreibweise: $M_1 \subset M_2$
- ▶ Analog definiere 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.

Notationalalarm: Manche Autoren verwenden \subset mit der Bedeutung \subseteq und \subsetneq statt \subset .

Einige wichtige Mengen

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

Einige wichtige Mengen

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



*Die ganzen Zahlen
hat der liebe Gott
gemacht, alles
andere ist
Menschenwerk.*

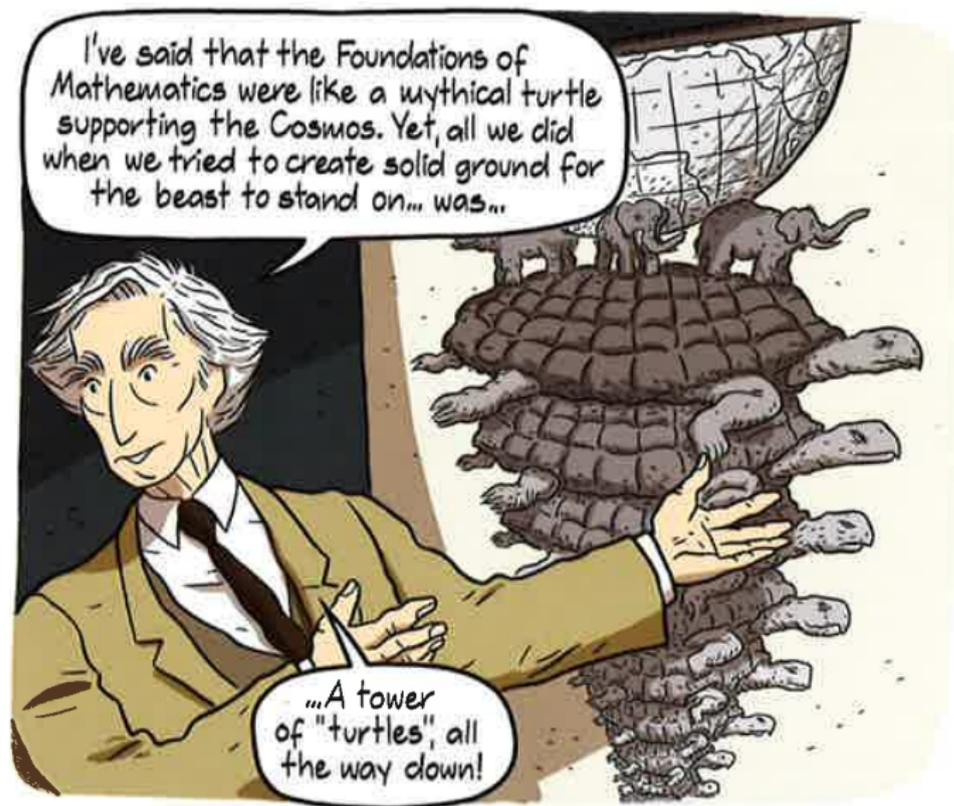
Leopold Kronecker
(1823–1891)

Übung: Mengenbeschreibungen

- ▶ Geben Sie formale Beschreibungen für die folgenden Mengen:
 - ▶ Alle geraden Zahlen
 - ▶ Alle Quadratzahlen
 - ▶ Alle Primzahlen

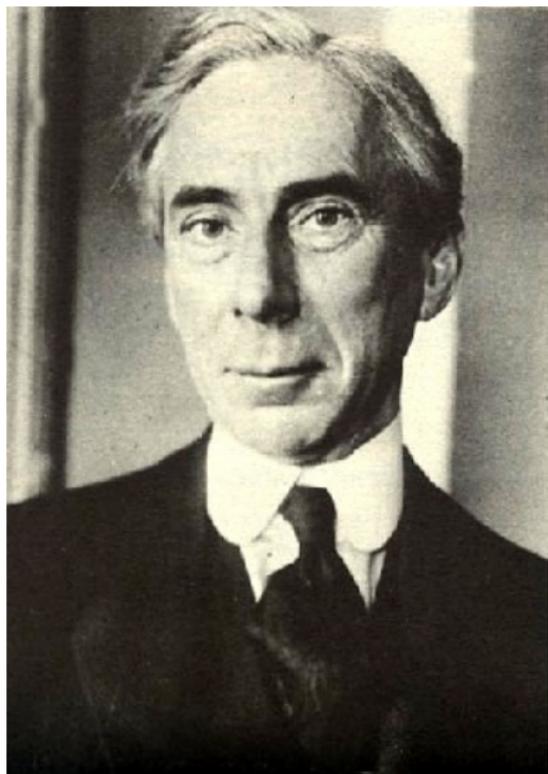
Zahlenkonstruktion und Termalgebra

Die Grundlagenkrise



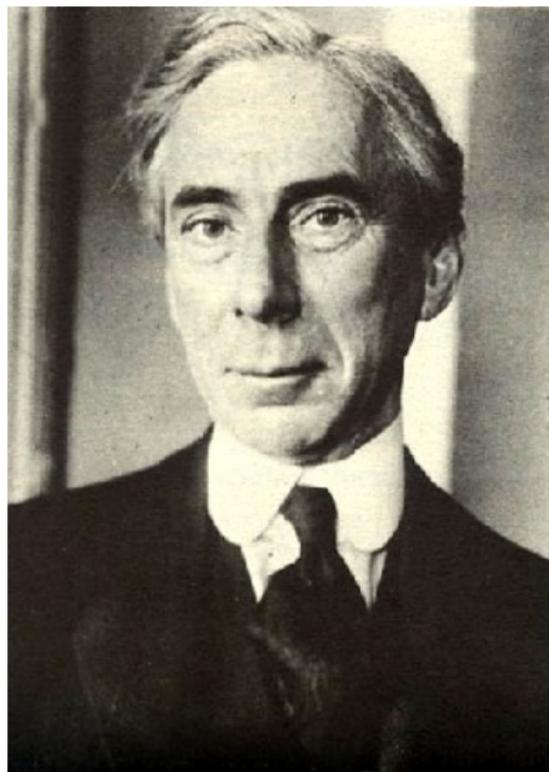
Aus: A. Doxidadis, C.H. Papadimitriou, *Logicomix - An Epic Search for Truth*

Die Grundlagenkrise



Bertrand Russel (1872–1970)

Die Grundlagenkrise



Bertrand Russel (1872–1970)



Alfred North Whitehead (1861-1947)

Mengenlehre ist die Grundlage der Mathematik (1)

Wir definieren eine Familie von Mengen wie folgt:

$M_0 = \{\}$ Leere Menge

$M_1 = \{\{\}\}$ Menge, die (nur) $\{\}$ enthält

$M_2 = \{\{\{\}\}\}$ usw.

$M_3 = \{\{\{\{\}\}\}\}$ usf.

...

$M_{i+1} = \{M_i\}$ Nachfolger enthält (nur) den Vorgänger

Mengenlehre ist die Grundlage der Mathematik (1)

Wir definieren eine Familie von Mengen wie folgt:

$M_0 = \{\}$ Leere Menge

$M_1 = \{\{\}\}$ Menge, die (nur) $\{\}$ enthält

$M_2 = \{\{\{\}\}\}$ usw.

$M_3 = \{\{\{\{\}\}\}\}$ usf.

...

$M_{i+1} = \{M_i\}$ Nachfolger enthält (nur) den Vorgänger

- ▶ Alle M_k sind verschieden!
- ▶ Außer M_0 enthält jedes M_k genau ein Element
- ▶ Wir können die Konstruktion beliebig fortsetzen

Mengenlehre ist die Grundlage der Mathematik (2)

Andere Sicht:

0 =	$\{\}$	Leere Menge
1 =	$\{\{\}\}$	Menge, die (nur) $\{\}$ enthält
2 =	$\{\{\{\}\}\}$	usw.
3 =	$\{\{\{\{\}\}\}\}$	usf.
...

Mengenlehre ist die Grundlage der Mathematik (3)

Und bequemere Schreibweise:

0 =	0	Leere Menge
1 =	$s(0)$	Menge, die (nur) $\{ \}$ enthält
2 =	$s(s(0))$	usw.
3 =	$s(s(s(0)))$	usf.
...

► Wir geben folgende Regeln an:

- $a(x, 0) = x$
- $a(x, s(y)) = s(a(x, y))$

► Beispielrechnung:

$$a(s(0), s(s(0))) =$$

Mengenlehre ist die Grundlage der Mathematik (3)

Und bequemere Schreibweise:

0 =	0	Leere Menge
1 =	$s(0)$	Menge, die (nur) $\{ \}$ enthält
2 =	$s(s(0))$	usw.
3 =	$s(s(s(0)))$	usf.
...

► Wir geben folgende Regeln an:

- $a(x, 0) = x$
- $a(x, s(y)) = s(a(x, y))$

► Beispielrechnung:

$$\begin{aligned} a(s(0), s(s(0))) &= s(a(s(0), s(0))) \\ &= \end{aligned}$$

Mengenlehre ist die Grundlage der Mathematik (3)

Und bequemere Schreibweise:

$0 =$	0	Leere Menge
$1 =$	$s(0)$	Menge, die (nur) $\{ \}$ enthält
$2 =$	$s(s(0))$	usw.
$3 =$	$s(s(s(0)))$	usf.
\dots	\dots	\dots

► Wir geben folgende Regeln an:

- $a(x, 0) = x$
- $a(x, s(y)) = s(a(x, y))$

► Beispielrechnung:

$$\begin{aligned} a(s(0), s(s(0))) &= s(a(s(0), s(0))) \\ &= s(s(a(s(0), 0))) \\ &= \end{aligned}$$

Mengenlehre ist die Grundlage der Mathematik (3)

Und bequemere Schreibweise:

$0 =$	0	Leere Menge
$1 =$	$s(0)$	Menge, die (nur) $\{ \}$ enthält
$2 =$	$s(s(0))$	usw.
$3 =$	$s(s(s(0)))$	usf.
\dots	\dots	\dots

► Wir geben folgende Regeln an:

- $a(x, 0) = x$
- $a(x, s(y)) = s(a(x, y))$

► Beispielrechnung:

$$\begin{aligned} a(s(0), s(s(0))) &= s(a(s(0), s(0))) \\ &= s(s(a(s(0), 0))) \\ &= s(s(s(0))) \end{aligned}$$

Mengenlehre ist die Grundlage der Mathematik (3)

Und bequemere Schreibweise:

0 =	0	Leere Menge
1 =	$s(0)$	Menge, die (nur) $\{ \}$ enthält
2 =	$s(s(0))$	usw.
3 =	$s(s(s(0)))$	usf.
...

► Wir geben folgende Regeln an:

- $a(x, 0) = x$
- $a(x, s(y)) = s(a(x, y))$

► Beispielrechnung:

$$\begin{aligned} a(s(0), s(s(0))) &= s(a(s(0), s(0))) \\ &= s(s(a(s(0), 0))) \\ &= s(s(s(0))) \end{aligned}$$

... oder auch $1 + 2 = 3$

Mengenlehre ist die Grundlage der Mathematik (3)

Und bequemere Schreibweise:

$0 =$	0	Leere Menge
$1 =$	$s(0)$	Menge, die (nur) $\{ \}$ enthält
$2 =$	$s(s(0))$	usw.
$3 =$	$s(s(s(0)))$	usf.
\dots	\dots	\dots

► Wir geben folgende Regeln an:

- $a(x, 0) = x$
- $a(x, s(y)) = s(a(x, y))$

► Beispielrechnung:

$$\begin{aligned} a(s(0), s(s(0))) &= s(a(s(0), s(0))) \\ &= s(s(a(s(0), 0))) \\ &= s(s(s(0))) \end{aligned}$$

... oder auch $1 + 2 = 3$



Übung: Multiplikation rekursiv

- ▶ Erweitern Sie das vorgestellte System um ein Funktionssymbol m und geeigneten Regeln, so dass m der Multiplikation auf den natürlichen Zahlen entspricht.

Übung: Konstruktion der negativen Zahlen

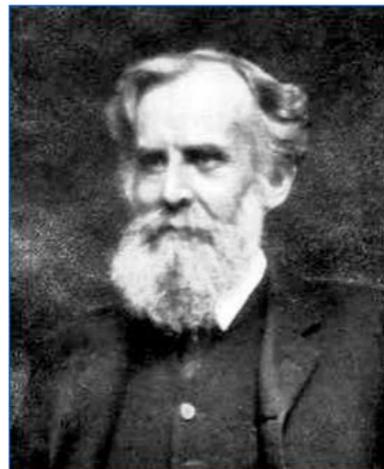
- ▶ Erweitern Sie die rekursiven Definitionen von p und m (Plus und Mal) auf $s, 0$ -Termen, um auch die negativen Zahlen behandeln zu können.
- ▶ Hinweis: Benutzen sie p („Vorgänger von“) und n (Negation)

Zurück C

Mengenoperationen

Venn-Diagramme

- ▶ Graphische Mengendarstellung
 - ▶ Mengen sind zusammenhängende Flächen
 - ▶ Überlappungen visualisieren gemeinsame Elemente
- ▶ Zeigen **alle** möglichen Beziehungen

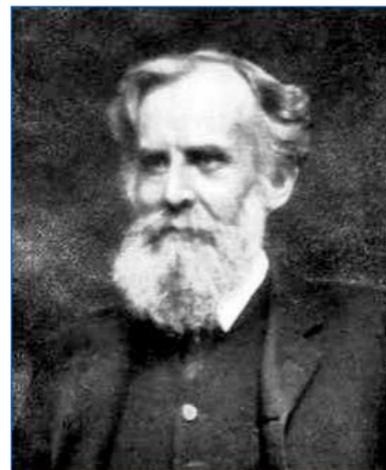
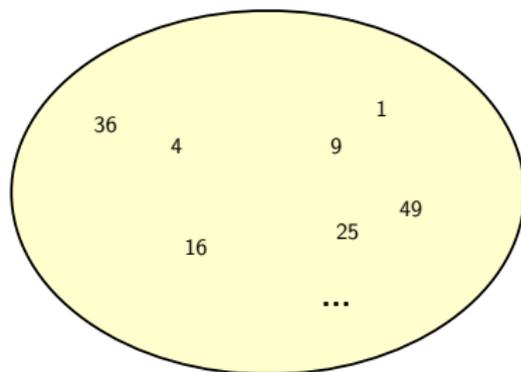


John Venn
(1834–1923)

Venn-Diagramme

- ▶ Graphische Mengendarstellung
 - ▶ Mengen sind zusammenhängende Flächen
 - ▶ Überlappungen visualisieren gemeinsame Elemente
- ▶ Zeigen **alle** möglichen Beziehungen

Quadratzahlen

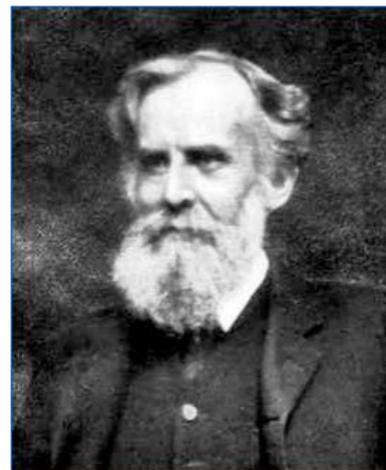
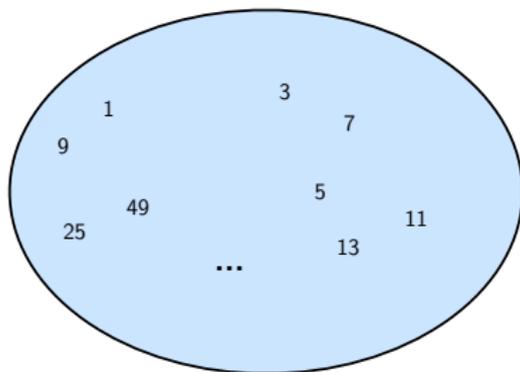


John Venn
(1834–1923)

Venn-Diagramme

- ▶ Graphische Mengendarstellung
 - ▶ Mengen sind zusammenhängende Flächen
 - ▶ Überlappungen visualisieren gemeinsame Elemente
- ▶ Zeigen **alle** möglichen Beziehungen

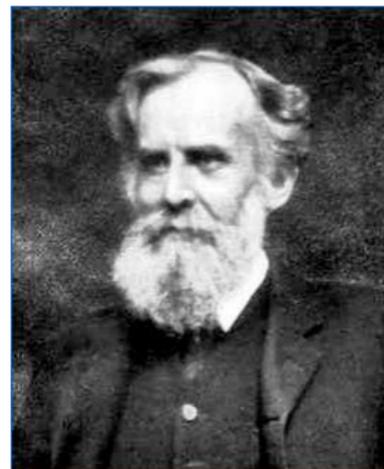
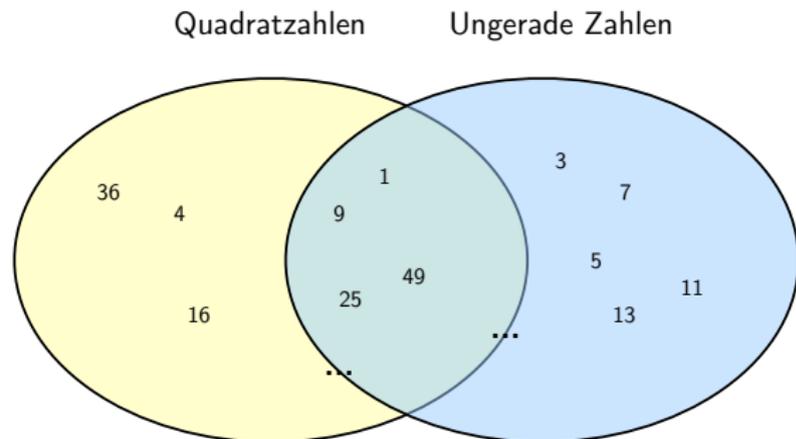
Ungerade Zahlen



John Venn
(1834–1923)

Venn-Diagramme

- ▶ Graphische Mengendarstellung
 - ▶ Mengen sind zusammenhängende Flächen
 - ▶ Überlappungen visualisieren gemeinsame Elemente
- ▶ Zeigen **alle** möglichen Beziehungen



John Venn
(1834–1923)

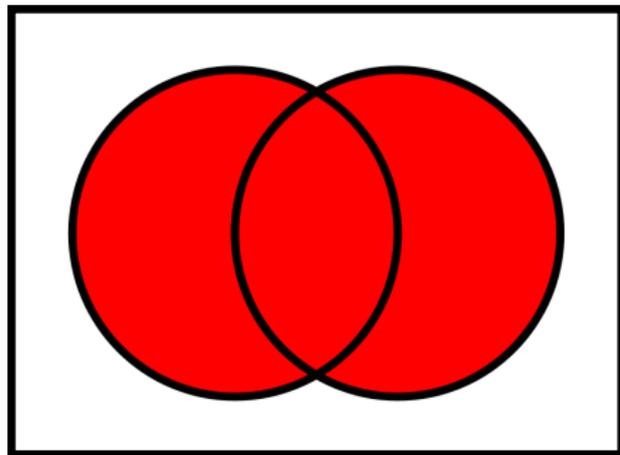
Mengenoperationen

Wir nehmen im folgenden an, dass alle betrachteten Mengen Teilmengen einer gemeinsamen **Trägermenge** T sind.

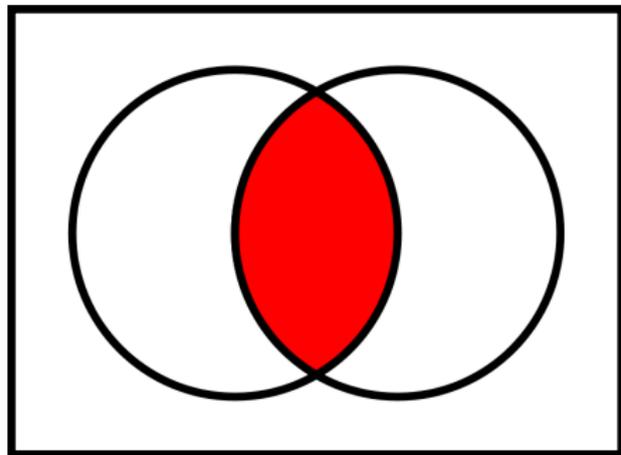
Wichtige Mengenoperationen sind:

- ▶ Vereinigung
- ▶ Schnitt
- ▶ Differenz
- ▶ Symmetrische Differenz
- ▶ Komplement

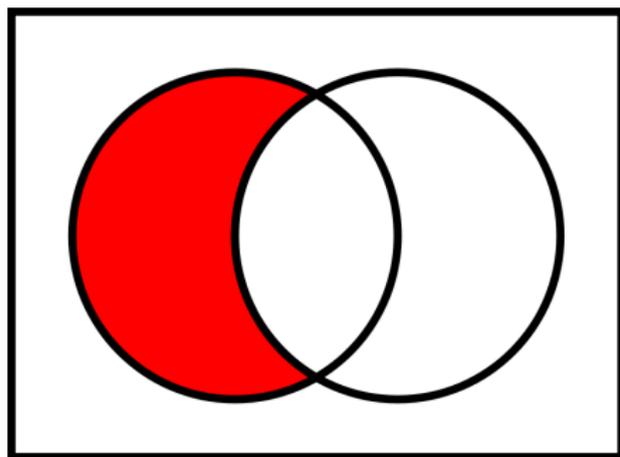
Vereinigungsmenge



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

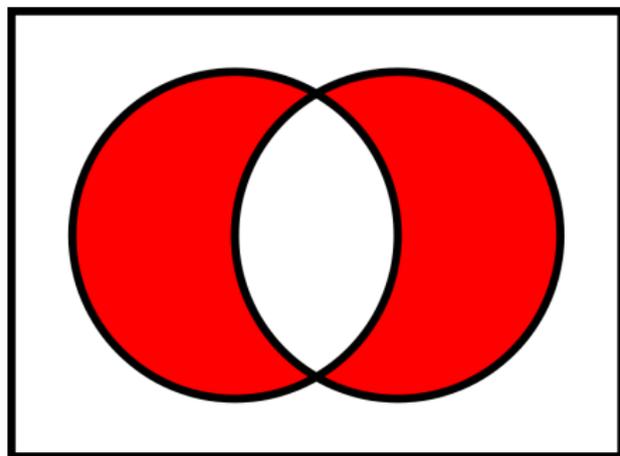


- ▶ $M_1 \cap M_2 = \{x \mid x \in M_1 \text{ und } x \in M_2\}$
- ▶ $x \in M_1 \cap M_2$ gdw. $x \in M_1$ und $x \in M_2$



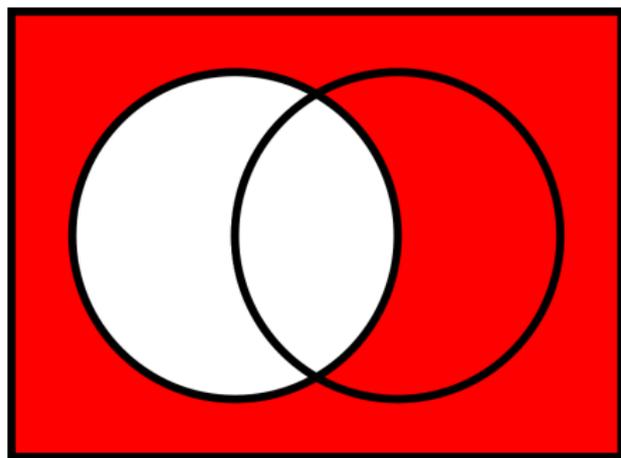
- ▶ $M_1 \setminus M_2 = \{x \mid x \in M_1 \text{ und } x \notin M_2\}$
- ▶ $x \in M_1 \setminus M_2$ gdw. $x \in M_1$ und $x \notin M_2$

Symmetrische Differenz



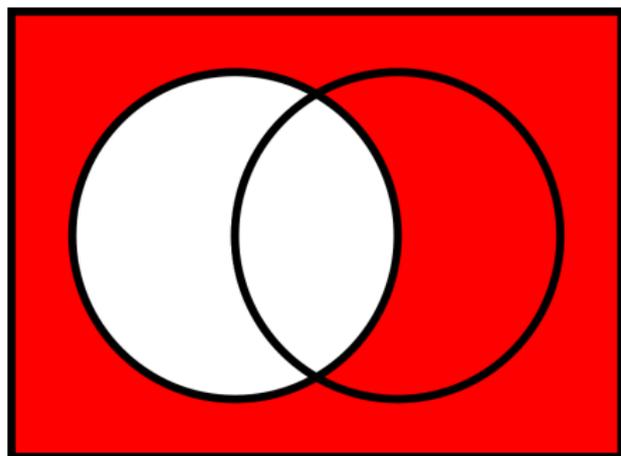
- ▶ $M_1 \Delta M_2 = \{x \mid x \in M_1 \text{ und } x \notin M_2\} \cup \{x \mid x \in M_2 \text{ und } x \notin M_1\}$
- ▶ $x \in M_1 \Delta M_2$ gdw. $x \in M_1$ oder $x \in M_2$, aber nicht $x \in M_1$ und $x \in M_2$

Komplement



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

Komplement



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

Hier ist die implizite Annahme der Trägermenge T (symbolisiert durch den eckigen Kasten) besonders wichtig!

Übung Mengenoperationen

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

Diskussion: Übung Mengenoperationen

- ▶ 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.
 - ▶ $M_1 \cup M_2$

Diskussion: Übung Mengenoperationen

- ▶ 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.
 - ▶ $M_1 \cup M_2$
 - ▶ Die Menge der ungeraden Zahlen und der Vielfachen von 3

Diskussion: Übung Mengenoperationen

- ▶ 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.
 - ▶ $M_1 \cup M_2$
 - ▶ Die Menge der ungeraden Zahlen und der Vielfachen von 3
 - ▶ $M_1 \cup M_2 = \{x \mid \exists i \in \mathbb{N} : x = 3i \text{ oder } x = 2i + 1\}$

Diskussion: Übung Mengenoperationen

- ▶ 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.
 - ▶ $M_1 \cup M_2$
 - ▶ Die Menge der ungeraden Zahlen und der Vielfachen von 3
 - ▶ $M_1 \cup M_2 = \{x \mid \exists i \in \mathbb{N} : x = 3i \text{ oder } x = 2i + 1\} = \{6i + k \mid i \in \mathbb{N}, k \in \{0, 1, 3, 5\}\}$
 - ▶ $M_1 \cap M_2$

Diskussion: Übung Mengenoperationen

- ▶ 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.
 - ▶ $M_1 \cup M_2$
 - ▶ Die Menge der ungeraden Zahlen und der Vielfachen von 3
 - ▶ $M_1 \cup M_2 = \{x \mid \exists i \in \mathbb{N} : x = 3i \text{ oder } x = 2i + 1\} = \{6i + k \mid i \in \mathbb{N}, k \in \{0, 1, 3, 5\}\}$
 - ▶ $M_1 \cap M_2$
 - ▶ Die Menge der ungeraden Vielfachen von 3

Diskussion: Übung Mengenoperationen

- ▶ 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.
 - ▶ $M_1 \cup M_2$
 - ▶ Die Menge der ungeraden Zahlen und der Vielfachen von 3
 - ▶ $M_1 \cup M_2 = \{x \mid \exists i \in \mathbb{N} : x = 3i \text{ oder } x = 2i + 1\} = \{6i + k \mid i \in \mathbb{N}, k \in \{0, 1, 3, 5\}\}$
 - ▶ $M_1 \cap M_2$
 - ▶ Die Menge der ungeraden Vielfachen von 3
 - ▶ $M_1 \cap M_2 = \{6i + 3 \mid i \in \mathbb{N}\}$
 - ▶ $M_1 \setminus M_2$

Diskussion: Übung Mengenoperationen

- ▶ 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.
 - ▶ $M_1 \cup M_2$
 - ▶ Die Menge der ungeraden Zahlen und der Vielfachen von 3
 - ▶ $M_1 \cup M_2 = \{x \mid \exists i \in \mathbb{N} : x = 3i \text{ oder } x = 2i + 1\} = \{6i + k \mid i \in \mathbb{N}, k \in \{0, 1, 3, 5\}\}$
 - ▶ $M_1 \cap M_2$
 - ▶ Die Menge der ungeraden Vielfachen von 3
 - ▶ $M_1 \cap M_2 = \{6i + 3 \mid i \in \mathbb{N}\}$
 - ▶ $M_1 \setminus M_2$
 - ▶ Die Menge der geraden Vielfachen von 3

Diskussion: Übung Mengenoperationen

- ▶ 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.
 - ▶ $M_1 \cup M_2$
 - ▶ Die Menge der ungeraden Zahlen und der Vielfachen von 3
 - ▶ $M_1 \cup M_2 = \{x \mid \exists i \in \mathbb{N} : x = 3i \text{ oder } x = 2i + 1\} = \{6i + k \mid i \in \mathbb{N}, k \in \{0, 1, 3, 5\}\}$
 - ▶ $M_1 \cap M_2$
 - ▶ Die Menge der ungeraden Vielfachen von 3
 - ▶ $M_1 \cap M_2 = \{6i + 3 \mid i \in \mathbb{N}\}$
 - ▶ $M_1 \setminus M_2$
 - ▶ Die Menge der geraden Vielfachen von 3
 - ▶ $M_1 \setminus M_2 = \{6i \mid i \in \mathbb{N}\}$
 - ▶ $M_1 \setminus \overline{M_2}$

Diskussion: Übung Mengenoperationen

- ▶ 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.
 - ▶ $M_1 \cup M_2$
 - ▶ Die Menge der ungeraden Zahlen und der Vielfachen von 3
 - ▶ $M_1 \cup M_2 = \{x \mid \exists i \in \mathbb{N} : x = 3i \text{ oder } x = 2i + 1\} = \{6i + k \mid i \in \mathbb{N}, k \in \{0, 1, 3, 5\}\}$
 - ▶ $M_1 \cap M_2$
 - ▶ Die Menge der ungeraden Vielfachen von 3
 - ▶ $M_1 \cap M_2 = \{6i + 3 \mid i \in \mathbb{N}\}$
 - ▶ $M_1 \setminus M_2$
 - ▶ Die Menge der geraden Vielfachen von 3
 - ▶ $M_1 \setminus M_2 = \{6i \mid i \in \mathbb{N}\}$
 - ▶ $M_1 \setminus \overline{M_2}$

Diskussion: Übung Mengenoperationen

- ▶ 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.
 - ▶ $M_1 \cup M_2$
 - ▶ Die Menge der ungeraden Zahlen und der Vielfachen von 3
 - ▶ $M_1 \cup M_2 = \{x \mid \exists i \in \mathbb{N} : x = 3i \text{ oder } x = 2i + 1\} = \{6i + k \mid i \in \mathbb{N}, k \in \{0, 1, 3, 5\}\}$
 - ▶ $M_1 \cap M_2$
 - ▶ Die Menge der ungeraden Vielfachen von 3
 - ▶ $M_1 \cap M_2 = \{6i + 3 \mid i \in \mathbb{N}\}$
 - ▶ $M_1 \setminus M_2$
 - ▶ Die Menge der geraden Vielfachen von 3
 - ▶ $M_1 \setminus M_2 = \{6i \mid i \in \mathbb{N}\}$
 - ▶ $M_1 \setminus \overline{M_2}$
 - ▶ Siehe $M_1 \cap M_2$

Diskussion: Übung Mengenoperationen

- ▶ 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.
 - ▶ $M_1 \cup M_2$
 - ▶ Die Menge der ungeraden Zahlen und der Vielfachen von 3
 - ▶ $M_1 \cup M_2 = \{x \mid \exists i \in \mathbb{N} : x = 3i \text{ oder } x = 2i + 1\} = \{6i + k \mid i \in \mathbb{N}, k \in \{0, 1, 3, 5\}\}$
 - ▶ $M_1 \cap M_2$
 - ▶ Die Menge der ungeraden Vielfachen von 3
 - ▶ $M_1 \cap M_2 = \{6i + 3 \mid i \in \mathbb{N}\}$
 - ▶ $M_1 \setminus M_2$
 - ▶ Die Menge der geraden Vielfachen von 3
 - ▶ $M_1 \setminus M_2 = \{6i \mid i \in \mathbb{N}\}$
 - ▶ $M_1 \setminus \overline{M_2}$
 - ▶ Siehe $M_1 \cap M_2$
 - ▶ $M_1 \triangle M_2$

Diskussion: Übung Mengenoperationen

- ▶ 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.
 - ▶ $M_1 \cup M_2$
 - ▶ Die Menge der ungeraden Zahlen und der Vielfachen von 3
 - ▶ $M_1 \cup M_2 = \{x \mid \exists i \in \mathbb{N} : x = 3i \text{ oder } x = 2i + 1\} = \{6i + k \mid i \in \mathbb{N}, k \in \{0, 1, 3, 5\}\}$
 - ▶ $M_1 \cap M_2$
 - ▶ Die Menge der ungeraden Vielfachen von 3
 - ▶ $M_1 \cap M_2 = \{6i + 3 \mid i \in \mathbb{N}\}$
 - ▶ $M_1 \setminus M_2$
 - ▶ Die Menge der geraden Vielfachen von 3
 - ▶ $M_1 \setminus M_2 = \{6i \mid i \in \mathbb{N}\}$
 - ▶ $M_1 \setminus \overline{M_2}$
 - ▶ Siehe $M_1 \cap M_2$
 - ▶ $M_1 \triangle M_2$
 - ▶ Die Menge der geraden Vielfachen von 3 und der ungeraden Zahlen, die nicht durch 3 teilbar sind

Diskussion: Übung Mengenoperationen

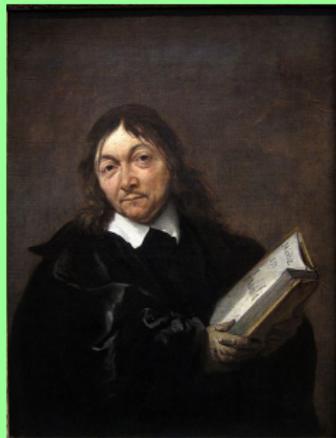
- ▶ Sei $T = \mathbb{N}$, $M_1 = \{3i \mid i \in \mathbb{N}\}$, $M_2 = \{2i + 1, i \in \mathbb{N}\}$. Berechnen Sie die folgenden Mengen. Geben Sie jeweils eine mathematische und eine umgangssprachliche Charakterisierung des Ergebnisses an.
 - ▶ $M_1 \cup M_2$
 - ▶ Die Menge der ungeraden Zahlen und der Vielfachen von 3
 - ▶ $M_1 \cup M_2 = \{x \mid \exists i \in \mathbb{N} : x = 3i \text{ oder } x = 2i + 1\} = \{6i + k \mid i \in \mathbb{N}, k \in \{0, 1, 3, 5\}\}$
 - ▶ $M_1 \cap M_2$
 - ▶ Die Menge der ungeraden Vielfachen von 3
 - ▶ $M_1 \cap M_2 = \{6i + 3 \mid i \in \mathbb{N}\}$
 - ▶ $M_1 \setminus M_2$
 - ▶ Die Menge der geraden Vielfachen von 3
 - ▶ $M_1 \setminus M_2 = \{6i \mid i \in \mathbb{N}\}$
 - ▶ $M_1 \setminus \overline{M_2}$
 - ▶ Siehe $M_1 \cap M_2$
 - ▶ $M_1 \triangle M_2$
 - ▶ Die Menge der geraden Vielfachen von 3 und der ungeraden Zahlen, die nicht durch 3 teilbar sind
 - ▶ $M_1 \triangle M_2 = \{6i \mid i \in \mathbb{N}\} \cup \{6i + k \mid i \in \mathbb{N}, k \in \{1, 5\}\} = \{6i + k \mid i \in \mathbb{N}, k \in \{0, 1, 5\}\}$

Kartesisches Produkt

Definition: Das **kartesische Produkt**

$M_1 \times M_2$ zweier Mengen M_1 und M_2 ist die Menge $\{(x, y) | x \in M_1, y \in M_2\}$.

- ▶ $M_1 \times M_2$ ist eine Menge von Paaren oder 2-Tupeln
- ▶ Verallgemeinerung: $M_1 \times M_2 \dots \times M_n = \{(x_1, x_2, \dots, x_n) | x_i \in M_i\}$ ist eine Menge von n -Tupeln
- ▶ Beispiel: $M_1 = \{1, 2, 3\}$, $M_2 = \{a, b\}$
 - ▶ $M_1 \times M_2 = \{(1, a), (2, a), (3, a), (1, b), (2, b), (3, b)\}$



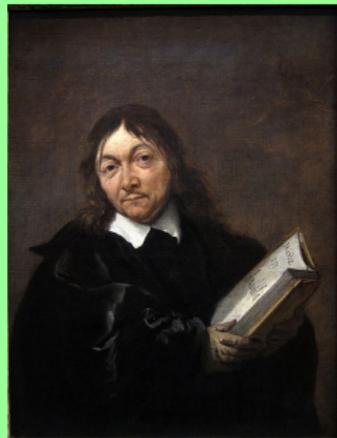
„Cogito, ergo sum“
René Descartes, *Discours de la méthode pour bien conduire sa raison, et chercher la vérité dans les sciences*, 1637

Kartesisches Produkt

Definition: Das **kartesische Produkt**

$M_1 \times M_2$ zweier Mengen M_1 und M_2 ist die Menge $\{(x, y) | x \in M_1, y \in M_2\}$.

- ▶ $M_1 \times M_2$ ist eine Menge von Paaren oder 2-Tupeln
- ▶ Verallgemeinerung: $M_1 \times M_2 \dots \times M_n = \{(x_1, x_2, \dots, x_n) | x_i \in M_i\}$ ist eine Menge von n -Tupeln
- ▶ Beispiel: $M_1 = \{1, 2, 3\}$, $M_2 = \{a, b\}$
 - ▶ $M_1 \times M_2 = \{(1, a), (2, a), (3, a), (1, b), (2, b), (3, b)\}$
 - ▶ $M_2 \times M_1 = ?$



„Cogito, ergo sum“
René Descartes, *Discours de la méthode pour bien conduire sa raison, et chercher la vérité dans les sciences*, 1637

Kartesisches Produkt

Definition: 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: $M_1 \times M_2 \dots \times M_n = \{(x_1, x_2, \dots, x_n) \mid x_i \in M_i\}$ ist eine Menge von n -Tupeln
- ▶ Beispiel: $M_1 = \{1, 2, 3\}$, $M_2 = \{a, b\}$
 - ▶ $M_1 \times M_2 = \{(1, a), (2, a), (3, a), (1, b), (2, b), (3, b)\}$
 - ▶ $M_2 \times M_1 = ?$
 - ▶ $M_1 \times M_1 = ?$



„Cogito, ergo sum“
René Descartes, *Discours de la méthode pour bien conduire sa raison, et chercher la vérité dans les sciences*, 1637

Definition: Die **Potenzmenge** 2^M einer Menge M ist die Menge aller Teilmengen von M , also $2^M = \{M' \mid M' \subseteq M\}$.

- ▶ Wichtig: $M \in 2^M$ und $\emptyset \in 2^M$
- ▶ Alternative Schreibweise: $\mathfrak{P}(M)$
- ▶ Beispiel: $M_1 = \{1, 2, 3\}$
 - ▶ $2^{M_1} = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$

Definition: Die **Potenzmenge** 2^M einer Menge M ist die Menge aller Teilmengen von M , also $2^M = \{M' \mid M' \subseteq M\}$.

- ▶ Wichtig: $M \in 2^M$ und $\emptyset \in 2^M$
- ▶ Alternative Schreibweise: $\mathfrak{P}(M)$
- ▶ Beispiel: $M_1 = \{1, 2, 3\}$
 - ▶ $2^{M_1} = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$
- ▶ $M_2 = \{a, b\}$
 - ▶ $2^{M_2} = ?$

Übung: Karthesisches Produkt und Potenzmenge

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

Berechnen Sie:

- ▶ $M_1 \times M_1$
- ▶ $M_1 \times M_2$
- ▶ $M_2 \times M_1$
- ▶ 2^{M_2}
- ▶ Warum lasse ich Sie nicht 2^{M_1} berechnen?

Lösung: $M_1 \times M_1$

$$M_1 \times M_1 =$$

Lösung: $M_1 \times M_1$

$$\begin{aligned} M_1 \times M_1 = & \\ & \{(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (1, 8), \\ & (2, 1), (2, 2), (2, 3), (2, 4), (2, 5), (2, 6), (2, 7), (2, 8), \\ & (3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (3, 6), (3, 7), (3, 8), \\ & (4, 1), (4, 2), (4, 3), (4, 4), (4, 5), (4, 6), (4, 7), (4, 8), \\ & (5, 1), (5, 2), (5, 3), (5, 4), (5, 5), (5, 6), (5, 7), (5, 8), \\ & (6, 1), (6, 2), (6, 3), (6, 4), (6, 5), (6, 6), (6, 7), (6, 8), \\ & (7, 1), (7, 2), (7, 3), (7, 4), (7, 5), (7, 6), (7, 7), (7, 8), \\ & (8, 1), (8, 2), (8, 3), (8, 4), (8, 5), (8, 6), (8, 7), (8, 8)\} \end{aligned}$$

Lösung: $M_1 \times M_2$

$$M_1 \times M_2 =$$

Lösung: $M_1 \times M_2$

$$\begin{aligned} M_1 \times M_2 = & \\ & \{(1, 2), (1, 4), (1, 6), (1, 8), (1, 10), (1, 12), \\ & (2, 2), (2, 4), (2, 6), (2, 8), (2, 10), (2, 12), \\ & (3, 2), (3, 4), (3, 6), (3, 8), (3, 10), (3, 12), \\ & (4, 2), (4, 4), (4, 6), (4, 8), (4, 10), (4, 12), \\ & (5, 2), (5, 4), (5, 6), (5, 8), (5, 10), (5, 12), \\ & (6, 2), (6, 4), (6, 6), (6, 8), (6, 10), (6, 12), \\ & (7, 2), (7, 4), (7, 6), (7, 8), (7, 10), (7, 12), \\ & (8, 2), (8, 4), (8, 6), (8, 8), (8, 10), (8, 12)\} \end{aligned}$$

Lösung: $M_2 \times M_1$

$$M_2 \times M_1 =$$

Lösung: $M_2 \times M_1$

$$M_2 \times M_1 =$$

$\{(2, 1), (2, 2), (2, 3), (2, 4), (2, 5), (2, 6), (2, 7), (2, 8),$
 $(4, 1), (4, 2), (4, 3), (4, 4), (4, 5), (4, 6), (4, 7), (4, 8),$
 $(6, 1), (6, 2), (6, 3), (6, 4), (6, 5), (6, 6), (6, 7), (6, 8),$
 $(8, 1), (8, 2), (8, 3), (8, 4), (8, 5), (8, 6), (8, 7), (8, 8),$
 $(10, 1), (10, 2), (10, 3), (10, 4), (10, 5), (10, 6), (10, 7), (10, 8),$
 $(12, 1), (12, 2), (12, 3), (12, 4), (12, 5), (12, 6), (12, 7), (12, 8)\}$

Lösung: Potenzmenge

Definition: Die **Potenzmenge** 2^M einer Menge M ist die Menge aller Teilmengen von M , also $2^M = \{M' \mid M' \subseteq M\}$.

- ▶ Sei $M_1 = \{1, 2, 3, 4, 5, 6, 7, 8\}$, $M_2 = \{2, 4, 6, 8, 10, 12\}$
 - ▶ Berechnen Sie 2^{M_2}
 - ▶ Warum lasse ich Sie nicht 2^{M_1} berechnen?
- ▶ $2^{M_2} =$

Lösung: Potenzmenge

Definition: Die **Potenzmenge** 2^M einer Menge M ist die Menge aller Teilmengen von M , also $2^M = \{M' \mid M' \subseteq M\}$.

- ▶ Sei $M_1 = \{1, 2, 3, 4, 5, 6, 7, 8\}$, $M_2 = \{2, 4, 6, 8, 10, 12\}$
 - ▶ Berechnen Sie 2^{M_2}
 - ▶ Warum lasse ich Sie nicht 2^{M_1} berechnen?
- ▶ $2^{M_2} = \{\{\}, \{10\}, \{12\}, \{10, 12\}, \{8\}, \{8, 10\}, \{8, 12\}, \{8, 10, 12\}, \{6\}, \{6, 10\}, \{6, 12\}, \{6, 10, 12\}, \{6, 8\}, \{6, 8, 10\}, \{6, 8, 12\}, \{6, 8, 10, 12\}, \{4\}, \{4, 10\}, \{4, 12\}, \{4, 10, 12\}, \{4, 8\}, \{4, 8, 10\}, \{4, 8, 12\}, \{4, 8, 10, 12\}, \{4, 6\}, \{4, 6, 10\}, \{4, 6, 12\}, \{4, 6, 10, 12\}, \{4, 6, 8\}, \{4, 6, 8, 10\}, \{4, 6, 8, 12\}, \{4, 6, 8, 10, 12\}, \{2\}, \{2, 10\}, \{2, 12\}, \{2, 10, 12\}, \{2, 8\}, \{2, 8, 10\}, \{2, 8, 12\}, \{2, 8, 10, 12\}, \{2, 6\}, \{2, 6, 10\}, \{2, 6, 12\}, \{2, 6, 10, 12\}, \{2, 6, 8\}, \{2, 6, 8, 10\}, \{2, 6, 8, 12\}, \{2, 6, 8, 10, 12\}, \{2, 4\}, \{2, 4, 10\}, \{2, 4, 12\}, \{2, 4, 10, 12\}, \{2, 4, 8\}, \{2, 4, 8, 10\}, \{2, 4, 8, 12\}, \{2, 4, 8, 10, 12\}, \{2, 4, 6\}, \{2, 4, 6, 10\}, \{2, 4, 6, 12\}, \{2, 4, 6, 10, 12\}, \{2, 4, 6, 8\}, \{2, 4, 6, 8, 10\}, \{2, 4, 6, 8, 12\}, \{2, 4, 6, 8, 10, 12\}\}$

Übung: M^3

- ▶ Sei $M = \{a, b, c\}$. Berechnen Sie $M^3 (= M \times M \times M)$.
- ▶ $M^3 =$

Übung: M^3

- ▶ Sei $M = \{a, b, c\}$. Berechnen Sie $M^3 (= M \times M \times M)$.
- ▶ $M^3 = \{(a, a, a), (a, a, b), (a, a, c), (a, b, a), (a, b, b), (a, b, c), (a, c, a), (a, c, b), (a, c, c), (b, a, a), (b, a, b), (b, a, c), (b, b, a), (b, b, b), (b, b, c), (b, c, a), (b, c, b), (b, c, c), (c, a, a), (c, a, b), (c, a, c), (c, b, a), (c, b, b), (c, b, c), (c, c, a), (c, c, b), (c, c, c)\}$

- ▶ Das Gebiet der **Algebra** beschäftigt sich mit den Eigenschaften von **Rechenoperationen**
- ▶ Eine **Algebraische Struktur** (oder nur **Algebra**) besteht aus:
 - ▶ Einer Menge (der Trägermenge)
 - ▶ Einer Menge von Operatoren auf dieser Menge
- ▶ Bekannte algebraische Strukturen:
 - ▶ $(\mathbb{Z}, +)$ ist eine **Gruppe**
 - ▶ $(\mathbb{Z}, +, *)$ ist ein **Ring**
 - ▶ $(\{0, s(0), s(s(0)), \dots\}, p, a)$ ist eine **Termalgebra**

- ▶ Das Gebiet der **Algebra** beschäftigt sich mit den Eigenschaften von **Rechenoperationen**
- ▶ Eine **Algebraische Struktur** (oder nur **Algebra**) besteht aus:
 - ▶ Einer Menge (der Trägermenge)
 - ▶ Einer Menge von Operatoren auf dieser Menge
- ▶ Bekannte algebraische Strukturen:
 - ▶ $(\mathbb{Z}, +)$ ist eine **Gruppe**
 - ▶ $(\mathbb{Z}, +, *)$ ist ein **Ring**
 - ▶ $(\{0, s(0), s(s(0)), \dots\}, p, a)$ ist eine **Termalgebra**
- ▶ **Mengenalgebra:**
 - ▶ Die Trägermenge der Algebra ist die Menge der Mengen über einem gemeinsamen Universum
 - ▶ Die Operatoren sind $\cup, \cap, \bar{}, \dots$

Algebraische Regeln (1)

$M_1, M_2, M_3 \subseteq T$ seien beliebige Teilmengen der gemeinsamen Trägermenge T . Es gelten:

▶ Kommutativgesetze

▶ $M_1 \cup M_2 = M_2 \cup M_1$

▶ $M_1 \cap M_2 = M_2 \cap M_1$

▶ Neutrale Elemente

▶ $M_1 \cup \emptyset = M_1$

▶ $M_1 \cap T = M_1$

▶ Absorption

▶ $M_1 \cup T = T$

▶ $M_1 \cap \emptyset = \emptyset$

▶ Assoziativgesetze

▶ $M_1 \cup (M_2 \cup M_3) = (M_1 \cup M_2) \cup M_3$

▶ $M_1 \cap (M_2 \cap M_3) = (M_1 \cap M_2) \cap M_3$

Algebraische Regeln (2)

$M_1, M_2, M_3 \subseteq T$ seien beliebige Teilmengen der gemeinsamen Trägermenge T .

▶ Distributivgesetze

▶ $M_1 \cup (M_2 \cap M_3) = (M_1 \cup M_2) \cap (M_1 \cup M_3)$

▶ $M_1 \cap (M_2 \cup M_3) = (M_1 \cap M_2) \cup (M_1 \cap M_3)$

▶ Inverse Elemente

▶ $M_1 \cup \overline{M_1} = T$

▶ $M_1 \cap \overline{M_1} = \emptyset$

▶ Idempotenz

▶ $M_1 \cup M_1 = M_1$

▶ $M_1 \cap M_1 = M_1$

▶ Gesetze von De-Morgan

▶ $\overline{M_1 \cup M_2} = \overline{M_1} \cap \overline{M_2}$

▶ $\overline{M_1 \cap M_2} = \overline{M_1} \cup \overline{M_2}$

▶ Doppelte Komplementbildung

▶ $\overline{\overline{M_1}} = M_1$



Augustus De Morgan
(1806-1871)

Relationen

Allgemeine Relationen

Definition: 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 $R \subseteq M_1 \times M_2 \times \dots \times M_n$ (oder äquivalent $R \in 2^{M_1 \times M_2 \times \dots \times M_n}$).

Allgemeine Relationen

Definition: 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 $R \subseteq M_1 \times M_2 \times \dots \times M_n$ (oder äquivalent $R \in 2^{M_1 \times M_2 \times \dots \times M_n}$).

► Beispiel:

- $M_1 = \{\text{Müller, Mayer, Schulze, Doe, Roe}\}$ (z.B. Personen)
- $M_2 = \{\text{Logik, Lineare Algebra, BWL, Digitaltechnik, PM}\}$ (z.B. Kurse)
- $\text{Belegt} = \{(\text{Müller, Logik}), (\text{Müller, BWL}), (\text{Müller, Digitaltechnik}), (\text{Mayer, BWL}), (\text{Mayer, PM}), (\text{Schulze, Lineare Algebra}), (\text{Schulze, Digitaltechnik}), (\text{Doe, PM})\}$

Allgemeine Relationen

Definition: 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 $R \subseteq M_1 \times M_2 \times \dots \times M_n$ (oder äquivalent $R \in 2^{M_1 \times M_2 \times \dots \times M_n}$).

► Beispiel:

- $M_1 = \{\text{Müller, Mayer, Schulze, Doe, Roe}\}$ (z.B. Personen)
- $M_2 = \{\text{Logik, Lineare Algebra, BWL, Digitaltechnik, PM}\}$ (z.B. Kurse)
- $\text{Belegt} = \{(\text{Müller, Logik}), (\text{Müller, BWL}), (\text{Müller, Digitaltechnik}), (\text{Mayer, BWL}), (\text{Mayer, PM}), (\text{Schulze, Lineare Algebra}), (\text{Schulze, Digitaltechnik}), (\text{Doe, PM})\}$
- Welche Kurse hat Mayer belegt?

Allgemeine Relationen

Definition: 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 $R \subseteq M_1 \times M_2 \times \dots \times M_n$ (oder äquivalent $R \in 2^{M_1 \times M_2 \times \dots \times M_n}$).

► Beispiel:

- $M_1 = \{\text{Müller, Mayer, Schulze, Doe, Roe}\}$ (z.B. Personen)
- $M_2 = \{\text{Logik, Lineare Algebra, BWL, Digitaltechnik, PM}\}$ (z.B. Kurse)
- $\text{Belegt} = \{(\text{Müller, Logik}), (\text{Müller, BWL}), (\text{Müller, Digitaltechnik}), (\text{Mayer, BWL}), (\text{Mayer, PM}), (\text{Schulze, Lineare Algebra}), (\text{Schulze, Digitaltechnik}), (\text{Doe, PM})\}$
- Welche Kurse hat Mayer belegt?
- Welche Kurse hat Roe belegt?

Allgemeine Relationen

Definition: 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 $R \subseteq M_1 \times M_2 \times \dots \times M_n$ (oder äquivalent $R \in 2^{M_1 \times M_2 \times \dots \times M_n}$).

- ▶ Beispiel:
 - ▶ $M_1 = \{\text{Müller, Mayer, Schulze, Doe, Roe}\}$ (z.B. Personen)
 - ▶ $M_2 = \{\text{Logik, Lineare Algebra, BWL, Digitaltechnik, PM}\}$ (z.B. Kurse)
 - ▶ $\text{Belegt} = \{(\text{Müller, Logik}), (\text{Müller, BWL}), (\text{Müller, Digitaltechnik}), (\text{Mayer, BWL}), (\text{Mayer, PM}), (\text{Schulze, Lineare Algebra}), (\text{Schulze, Digitaltechnik}), (\text{Doe, PM})\}$
 - ▶ Welche Kurse hat Mayer belegt?
 - ▶ Welche Kurse hat Roe belegt?
- ▶ Wir schreiben oft $R(x, y)$ statt $(x, y) \in R$
 - ▶ Im Beispiel also z.B. $\text{Belegt}(\text{Schulze, Digitaltechnik})$

- ▶ 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: Sei R eine Relation über M_1, M_2, \dots, M_n .

- ▶ R heißt **homogen**, falls $M_i = M_j$ für alle $i, j \in \{1, \dots, n\}$.
- ▶ R heißt **binär**, falls $n = 2$.
- ▶ R heißt **homogene binäre Relation**, fall R homogen und binär ist.

Definition: Sei R eine Relation über M_1, M_2, \dots, M_n .

- ▶ R heißt **homogen**, falls $M_i = M_j$ für alle $i, j \in \{1, \dots, n\}$.
 - ▶ R heißt **binär**, falls $n = 2$.
 - ▶ R heißt **homogene binäre Relation**, fall R homogen und binär ist.
- ▶ Wenn R homogen ist, so nennen wir R auch eine Relation über M

Definition: Sei R eine Relation über M_1, M_2, \dots, M_n .

- ▶ R heißt **homogen**, falls $M_i = M_j$ für alle $i, j \in \{1, \dots, n\}$.
 - ▶ R heißt **binär**, falls $n = 2$.
 - ▶ R heißt **homogene binäre Relation**, fall R homogen und binär ist.
-
- ▶ Wenn R homogen ist, so nennen wir R auch eine Relation über M
 - ▶ Im Fall von binären Relationen schreiben wir oft xRy statt $R(x, y)$ (z.B. $1 < 2$ statt $<(1, 2)$ oder $(1, 2) \in <$)

Definition: Sei R eine Relation über M_1, M_2, \dots, M_n .

- ▶ R heißt **homogen**, falls $M_i = M_j$ für alle $i, j \in \{1, \dots, n\}$.
 - ▶ R heißt **binär**, falls $n = 2$.
 - ▶ R heißt **homogene binäre Relation**, fall R homogen und binär ist.
-
- ▶ Wenn R homogen ist, so nennen wir R auch eine Relation über M
 - ▶ Im Fall von binären Relationen schreiben wir oft xRy statt $R(x, y)$ (z.B. $1 < 2$ statt $< (1, 2)$ oder $(1, 2) \in <$)
 - ▶ Im folgenden nehmen wir bis auf weiteres an, dass Relationen homogen und binär sind, soweit nichts anderes spezifiziert ist

- ▶ Beispiele für homogene binäre Relationen:
 - ▶ $=$ über \mathbb{N}
 - ▶ $= = \{(0, 0), (1, 1), (2, 2), \dots\}$
 - ▶ $<$ über \mathbb{Z}
 - ▶ $< = \{(i, i + j) \mid i \in \mathbb{Z}, j \in \mathbb{N}^+\}$
 - ▶ \neq über $\{w \mid w \text{ ist ein deutscher Name}\}$
 - ▶ $\{(Müller, Mayer), (Müller, Schulze), (Mayer, Schulze), (Mayer, Müller), (Schulze, Mayer), (Schulze, Müller)\} \subseteq \neq$
 - ▶ \subseteq über 2^M für eine Menge M
 - ▶ Z.B. $(\{a, b\}, \{a, b, c\}) \in \subseteq$

Eigenschaften von Relationen (1)

Definition: Sei R eine binäre Relation über $A \times B$.

- ▶ Gilt für alle $\forall a \in A \exists b \in B$ mit $R(a, b)$, so heißt R **linkstotal**
 - ▶ Gilt für alle $\forall a \in A, \forall b, c \in B : R(a, b)$ und $R(a, c)$ impliziert $b = c$, so heisst R **rechtseindeutig**
-
- ▶ Linkstotal: Jedes Element aus A steht mit mindestens einem Element aus B in Relation
 - ▶ Rechtseindeutig: Jedes Element aus A steht mit höchstens einem Element aus B in Relation.

Eigenschaften von Relationen (2)

Definition: Sei R eine homogene binäre Relation über A .

- ▶ **Gilt** $\forall a \in A : R(a, a)$, so heißt R **reflexiv**
- ▶ **Gilt** $\forall a, b \in A : R(a, b)$ impliziert $R(b, a)$, so heißt R **symmetrisch**
- ▶ **Gilt** $\forall a, b, c \in A : R(a, b)$ und $R(b, c)$ implizieren $R(a, c)$, so heißt R **transitiv**
- ▶ Ist R reflexiv, symmetrisch und transitiv, so ist R eine **Äquivalenzrelation**

- ▶ Untersuchen Sie für die folgenden Relationen, ob sie linkstotal, rechtseindeutig, reflexiv, symmetrisch, transitiv sind. Geben Sie jeweils eine Begründung oder ein Gegenbeispiel an.
 - ▶ $> \subseteq \mathbb{N}^2$
 - ▶ $\leq \subseteq \mathbb{N}^2$
 - ▶ $= \subseteq A \times A$ (die Gleichheitsrelation auf einer beliebigen nichtleeren Menge A)
- ▶ Zeigen oder widerlegen Sie:
 - ▶ Jede Äquivalenzrelation ist linkstotal
 - ▶ Jede Äquivalenzrelation ist rechtseindeutig

Darstellung von Relationen: Mengendarstellung

Wir können (endliche) Relationen auf verschiedene Arten darstellen (und im Computer repräsentieren).

- ▶ Bekannt: Mengendarstellung

- ▶ Liste alle Tupel auf, die in Relation stehen

- ▶ Beispiel: $M = \{0, 1, 2, 3\}$.

- $R = \{(0, 0), (0, 1), (1, 2), (2, 3), (3, 3), (3, 1)\}$

- ▶ Vorteile:

- ▶ Kompakt

- ▶ Einfach zu implementieren

- ▶ Nachteil:

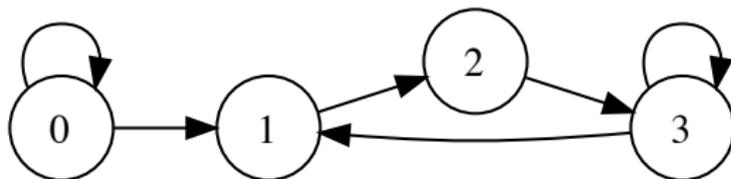
- ▶ Nicht anschaulich

- ▶ Nicht übersichtlich

- ▶ Prüfen, ob zwei Elemente in Relation steht, dauert lange (Liste durchsuchen)

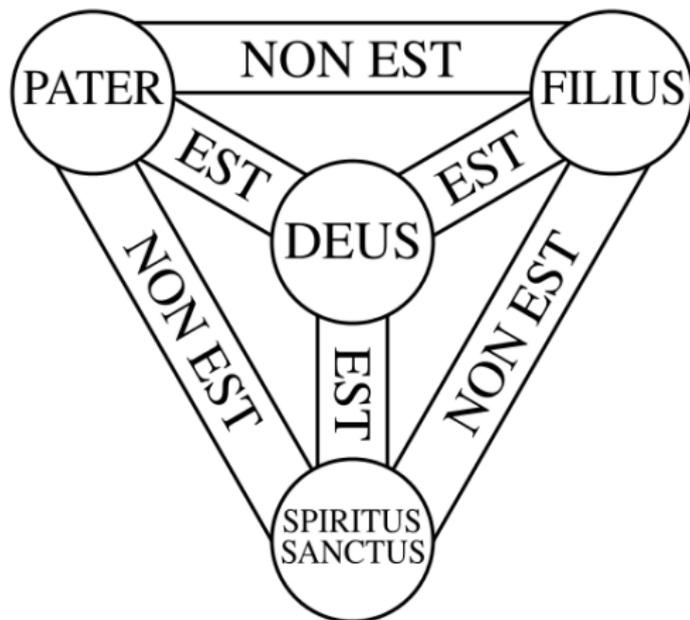
Darstellung von (endlichen) Relationen: Graphdarstellung

- ▶ Graphdarstellung
 - ▶ Elemente sind **Knoten**
 - ▶ Zwei Elemente x, y sind mit einer **Kante** verbunden, wenn xRy gilt
 - ▶ Beispiel: $M = \{0, 1, 2, 3\}$.
 $R = \{(0, 0), (0, 1), (1, 2), (2, 3), (3, 3), (3, 1)\}$

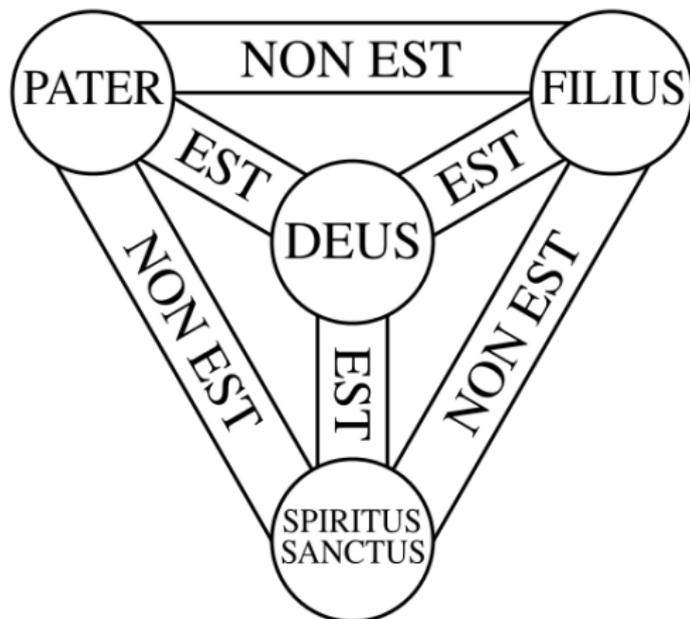


- ▶ Vorteile:
 - ▶ Übersichtlich (wenn der Graph nicht zu groß ist)
 - ▶ Anschaulich: Manche Eigenschaften können leicht erkannt werden
- ▶ Nachteile:
 - ▶ Nur anschaulich - wie repräsentieren wir den Graph im Rechner?
 - ▶ ... und beim Malen: Plazieren von Knoten und Kanten ist nicht trivial
 - ▶ Übersichtlichkeit geht bei komplexen Relationen verloren

Historisches Beispiel/Übung



Historisches Beispiel/Übung



- Beschreiben Sie die gezeigte(n) Relation(en)

Darstellung von (endlichen) Relationen: Tabellendarstellung

- ▶ Darstellung als Tabelle oder Matrix
 - ▶ Tabellenzeilen und Spalten sind mit Elementen beschriftet
 - ▶ An Stelle Zeile x , Spalte y steht eine 1, wenn xRy , sonst 0

	0	1	2	3
0	1	1	0	0
1	0	0	1	0
2	0	0	0	1
3	0	3	0	1

Darstellung von (endlichen) Relationen: Tabellendarstellung

- ▶ Darstellung als Tabelle oder Matrix

- ▶ Tabellenzeilen und Spalten sind mit Elementen beschriftet
- ▶ An Stelle Zeile x , Spalte y steht eine 1, wenn xRy , sonst 0

	0	1	2	3
0	1	1	0	0
1	0	0	1	0
2	0	0	0	1
3	0	3	0	1

... oder als Matrix: $\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 3 & 0 & 1 \end{pmatrix}$

- ▶ Vorteile:

- ▶ Sehr einfach im Rechner realisierbar
- ▶ Prüfen, ob xRy geht schnell („lookup“)
- ▶ Übersichtlicher als Mengen
- ▶ Manche Eigenschaften können leicht erkannt werden

- ▶ Nachteile:

- ▶ Viel Speicherbedarf (immer n^2 Einträge)
- ▶ Ich verwechsele immer Zeilen und Spalten ;-)

Definition: Seien R eine Relation. Die **inverse Relation** (zu R) ist $R^{-1} = \{(y, x) \mid (x, y) \in R\}$.

- ▶ Für symmetrische Relationen gilt $R^{-1} = R$
- ▶ Beispiele:
 - ▶ $> \subseteq \mathbb{N} \times \mathbb{N}$ (die normale „größer“-Relation) ist die inverse Relation zu $<$, also formal: $<^{-1} = >$
 - ▶ Für die Gleichheitsrelation gilt: $=^{-1} = =$ (und das ist kein Tippfehler - lies: „Die inverse Relation der Gleichheitsrelation ist wieder die Gleichheitsrelation“)

Verknüpfung von Relationen

Definition: Seien R_1, R_2 zwei Relationen. Das **Relationsprodukt** $R_1 \circ R_2$ ist die Relation $\{(x, y) \mid \exists z : (x, z) \in R_1 \text{ und } (z, y) \in R_2\}$.

- ▶ Beachte: Es gilt nicht immer $R_1 \subseteq R_1 \circ R_2$ oder $R_2 \subseteq R_1 \circ R_2$

Verknüpfung von Relationen

Definition: Seien R_1, R_2 zwei Relationen. Das **Relationsprodukt** $R_1 \circ R_2$ ist die Relation $\{(x, y) \mid \exists z : (x, z) \in R_1 \text{ und } (z, y) \in R_2\}$.

- ▶ Beachte: Es gilt nicht immer $R_1 \subseteq R_1 \circ R_2$ oder $R_2 \subseteq R_1 \circ R_2$

Definition: Sei R eine Relation über M . Wir definieren:

- ▶ $R^0 = \{(x, x) \mid x \in M\}$ (die **Gleichheitsrelation** oder **Identität**)
- ▶ $R^n = R \circ R^{n-1}$ für $n \in \mathbb{N}^+$

Übung: Relationen

Sei $M = \{a, b, c, d\}$, $R = \{(a, b), (b, c), (c, d)\}$,
 $S = \{(a, a), (b, b), (c, c)\}$. Berechnen Sie die folgenden Relationen und stellen Sie sie als Matrix und Graph da:

- ▶ R^{-1}
- ▶ R^0
- ▶ R^1
- ▶ R^2
- ▶ R^3
- ▶ R^4
- ▶ $S \circ S$
- ▶ $R \circ S$

Hüllenbildung

Definition: Seien R eine Relation. Dann gilt:

- ▶ Die **reflexive Hülle** $R \cup R^0$ von R ist die kleinste Relation, die R enthält und reflexiv ist.
- ▶ Die **symmetrische Hülle** $R \cup R^{-1}$ von R ist die kleinste Relation, die R enthält und symmetrisch ist.
- ▶ Die **transitive Hülle** R^+ von R ist die kleinste Relation, die R enthält und transitiv ist.
- ▶ Die **reflexive und transitive Hülle** R^* von R ist die kleinste Relation, die R enthält und reflexiv und transitiv ist.
- ▶ Die **reflexive, symmetrische und transitive Hülle** $(R \cup R^{-1})^*$ von R ist die kleinste Äquivalenzrelation, die R enthält.

„Kleinste“ bezieht sich auf die Teilmengenrelation (\subset)

- ▶ $\{a, b\}$ ist in diesem Sinne kleiner als $\{a, b, c\}$, aber nicht kleiner als $\{b, c, d\}$

Sei $M = \{a, b, c, d\}$, $R = \{(a, b), (b, c), (c, d)\}$ (wie oben). Berechnen Sie zu R

- ▶ Die reflexive Hülle
- ▶ Die symmetrische Hülle
- ▶ Die transitive Hülle
- ▶ Die reflexive transitive Hülle
- ▶ Die reflexive, symmetrische und transitive Hülle

und stellen Sie sie als Tabelle/Matrix da.

Funktionen

Definition: Seien M, N Mengen.

- ▶ Eine (totale) Funktion $f : M \rightarrow N$ ist eine Relation $f \subseteq (M \times N)$, die linkstotal und rechtseindeutig ist.
- ▶ Eine partielle Funktion $f : M \rightarrow N$ ist eine Relation $f \subseteq (M \times N)$, die rechtseindeutig ist.

▶ Anmerkungen:

- ▶ Eine Funktion (auch: **Abbildung**) ordnet (jedem) Element aus M höchstens ein Element aus N zu
 - ▶ Mathematische Funktionen sind **total**
 - ▶ Informatische Funktionen sind mal so, mal
- ▶ M heißt **Definitionsmenge** von f
- ▶ N heißt **Zielmenge** von f
- ▶ Oft wird eine konkrete Funktion durch eine Zuordnungsvorschrift definiert:
 - ▶ $f : \mathbb{N} \rightarrow \mathbb{N}, x \mapsto x^2$
 - ▶ $g : \mathbb{Z} \rightarrow \mathbb{N}, x \mapsto |x|$
 - ▶ Wir schreiben konkret: $f(x) = y$ statt $(x, y) \in f$ oder xfy

Definition: Sei M, N ein Mengen und $f : M \rightarrow N$ eine Funktion. Sei $M_0 \subseteq M_1$ und $N_0 \subseteq N_1$

- ▶ $f(M_0) = \{y \mid \exists x \in M_0 : f(x) = y\}$ heißt das **Bild** von M_0 unter f .
- ▶ $\{x \mid \exists y \in N_0 : f(x) = y\}$ heißt das **Urbild** von N_0 ,

Eigenschaften von Funktionen

Sei $f : M \rightarrow N$ eine Funktion.

- ▶ f heißt **surjektiv**, wenn $\forall y \in N \exists x \in M : f(x) = y$
 - ▶ f heißt **injektiv**, wenn $\forall y \in N : f(x) = y$ und $f(z) = y \leadsto x = z$
 - ▶ f heißt **bijektiv** (oder „1-zu-1“), wenn f injektiv und surjektiv ist
-
- ▶ Wenn f surjektiv ist, so gilt $f(M) = N$
 - ▶ Wenn f injektiv ist, so ist f^{-1} rechtseindeutig (also eine (partielle) Funktion)

- ▶ Betrachten Sie die folgenden Funktionen:
 - ▶ $f_1 : \mathbb{Z} \rightarrow \mathbb{N}, x \mapsto |x|$
 - ▶ $f_2 : \mathbb{N} \rightarrow \mathbb{N}, x \mapsto |x|$
 - ▶ $f_3 : \mathbb{N} \rightarrow \mathbb{N}, x \mapsto 2x$
 - ▶ $f_4 : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}, (x, y) \mapsto x + y$
- ▶ Welche der Funktionen sind injektiv, surjektiv, bijektiv?
- ▶ Für f_1, f_2, f_3 : Was ist jeweils das Bild und das Urbild von $\{2, 4, 6, 8\}$?
- ▶ Für f_4 : Was ist das Urbild von $\{6, 8\}$?

- ▶ Die Mächtigkeit oder **Kardinalität** $|M|$ einer Menge M ist ein Maß für die Anzahl der Elemente in M
- ▶ Zwei Mengen M, N sind gleichmächtig, wenn eine bijektive Abbildung $f : M \rightarrow N$ existiert
- ▶ Für endliche Mengen ist $|M|$ die Anzahl der Elemente in M
- ▶ Eine unendliche Menge heißt **abzählbar**, wenn Sie die selbe Kardinalität wie \mathbb{N} hat

Übung: Kardinalität

- ▶ Zeigen Sie: \mathbb{Z} ist abzählbar
- ▶ Für endliche Mengen M gilt: $|2^M| = 2^{|M|}$

Zurück C

Übung: Relationen für Fortgeschrittene

- ▶ Betrachten Sie die Menge $M = \{a, b, c\}$.
 - ▶ Wie viele (binäre homogene) Relationen über M gibt es?
 - ▶ Wie viele dieser Relationen sind
 - ▶ Linkstotal
 - ▶ Rechtseindeutig
 - ▶ Reflexiv
 - ▶ Symmetrisch
 - ▶ Transitiv (das könnte schwieriger sein ;-)
 - ▶ Funktionen (einschließlich partieller Funktionen)
 - ▶ Totale Funktionen?
- ▶ Betrachten Sie folgende Relation über \mathbb{N} : xRy gdw. $x = y + 2$
 - ▶ Was ist die transitive Hülle von R ?
 - ▶ Was ist die reflexive, symmetrische, transitive Hülle von R ?
 - ▶ Betrachten Sie $R' = R \cup \{(0, 1)\}$. Was ist die transitive Hülle von R' ?
- ▶ Zeigen oder widerlegen (per Gegenbeispiel) Sie:
 - ▶ Jede homogene binäre symmetrische und transitive Relation ist eine Äquivalenzrelation
 - ▶ Jede linkstotale homogene binäre symmetrische und transitive Relation ist eine Äquivalenzrelation

Funktionales Programmieren mit Scheme

A language that doesn't affect the way you think about programming, is not worth knowing.

Alan Perlis (1982)

Any sufficiently complicated C or Fortran program contains an ad hoc, informally-specified, bug-ridden, slow implementation of half of Common Lisp.

Philip Greenspun (ca. 1993)

Lisp is worth learning for the profound enlightenment experience you will have when you finally get it; that experience will make you a better programmer for the rest of your days, even if you never actually use Lisp itself a lot.

Eric S. Raymond (2001)

- ▶ LISP: **L**ist **P**rocessing
 - ▶ 1958 von John McCarthy entworfen
 - ▶ Realisiert Church's λ -Kalkül
 - ▶ Implementierung durch Steve Russell
- ▶ Wichtige Dialekte:
 - ▶ Scheme (seit 1975, aktueller Standard R⁷RS, 2013)
 - ▶ Common Lisp (1984, ANSI 1994)
- ▶ Eigenschaften von Lisp
 - ▶ Funktional
 - ▶ Interaktiv (read-eval-print)
 - ▶ Einfache, konsistente Syntax (*s-expressions*)
 - ▶ ... für Daten und Programme
 - ▶ Dynamisch getypt
- ▶ Eigenschaften von Scheme
 - ▶ Minimalistisch
 - ▶ Iteration ((fast) nur) durch Rekursion



Alonzo
Church
(1903–1995)



Steve Russel
(1937–)

Unpersonal Computers



„The type 704 Electronic Data-Processing Machine is a large-scale, high-speed electronic calculator controlled by an internally stored program of the single address type.“
IBM 704 Manual of operation

Lisp in the Real World

- ▶ KI-Systeme und Reasoner
 - ▶ S-Setheo
 - ▶ Gandalf
 - ▶ ACL2
 - ▶ Viele Expertensysteme
- ▶ Scripting
 - ▶ Emacs (ELisp)
 - ▶ AutoCAD (AutoLISP)
 - ▶ GIMP (SIOD/TinyScheme)
 - ▶ LilyPond/gdb/GnuCash (Guile)
- ▶ Sonstiges
 - ▶ (Yahoo Stores)
 - ▶ Real-Time Börsenhandel

Übung: Hello World

```
;; Definiere Funktion hello, die "Hello World  
;; ausgibt  
(define (hello)  
  (display "Hello World")  
  (newline)  
)  
  
;; Aufruf der Funktion  
(hello)
```

► Führen Sie das Programm aus

- Eintippen oder Download von <http://www.lehre.dhbw-stuttgart.de/~ssschulz/lgli2014.html>
- Von der Kommandozeile (z.B.): `> guile-2.0 hello.scm`
- Aus dem Scheme-Interpreter: `> (load "hello.scm")`

Ein funktionaleres Beispiel

- ▶ Die Fakultät einer natürlichen Zahl n ist das Produkt der Zahlen von 1 bis n : $fak(n) = \prod_{i=1}^n i$
 - ▶ $fak(3) = 6, fak(5) = 120, \dots$
- ▶ Rekursiv:
 - ▶ $fak(0) = 1$
 - ▶ $fak(n) = n fak(n - 1)$ (falls $n > 0$)
- ▶ In Scheme:

```
;; Factorial
(define (fak x)
  (if (= x 0)
      1
      (* x (fak (- x 1)))))
)
```

Syntax von Scheme

- ▶ Scheme-Programme bestehen aus **symbolischen Ausdrücken** (*s-expressions*)
- ▶ Definition *s-expression* (etwas vereinfacht):
 - ▶ Atome (Zahlen, Strings, Identifier, ...) sind *s-expressions*
 - ▶ Wenn e_1, e_2, \dots, e_n *s-expressions* sind, dann auch $(e_1 e_2 \dots e_n)$ (eine Liste von *s-expressions* ist eine *s-expression*)
- ▶ Beachte: Verschachtelte Listen sind möglich, und der Normalfall!
- ▶ Beispiele
 - ▶ "a" (ein String)
 - ▶ + (ein Identifier mit vordefinierter Bedeutung)
 - ▶ if (ein Identifier mit vordefinierter Bedeutung)
 - ▶ fak (ein Identifier ohne vordefinierte Bedeutung)
 - ▶ (+ 1 2) (ein Ausdruck in Prefix-Notation)
 - ▶ (+ 3 (* 5 2) (- 2 3)) (ein verschachtelter Ausdruck)

Syntax von Scheme

- ▶ Scheme-Programme bestehen aus **symbolischen Ausdrücken** (*s-expressions*)
- ▶ Definition *s-expression* (etwas vereinfacht):
 - ▶ Atome (Zahlen, Strings, Identifier, ...) sind *s-expressions*
 - ▶ Wenn e_1, e_2, \dots, e_n *s-expressions* sind, dann auch $(e_1 e_2 \dots e_n)$ (eine Liste von *s-expressions* ist eine *s-expression*)
- ▶ Beachte: Verschachtelte Listen sind möglich, und der Normalfall!
- ▶ Beispiele
 - ▶ "a" (ein String)
 - ▶ + (ein Identifier mit vordefinierter Bedeutung)
 - ▶ if (ein Identifier mit vordefinierter Bedeutung)
 - ▶ fak (ein Identifier ohne vordefinierte Bedeutung)
 - ▶ (+ 1 2) (ein Ausdruck in Prefix-Notation)
 - ▶ (+ 3 (* 5 2) (- 2 3)) (ein verschachtelter Ausdruck)
 - ▶ (define (fak x)(if (= x 0) 1 (* x (fak (- x 1)))))

Read-Eval-Print

LISP programmers know the value of everything and the cost of nothing.

Alan Perlis (1982)

LISP programmers know the value of everything and the cost of nothing.

Alan Perlis (1982)

- ▶ Der Scheme-Interpreter ist eine *read-eval-print*-Schleife
 - ▶ Der Interpreter liest s-expressions vom Nutzer („read“)
 - ▶ Eintippen, oder *Copy&Paste* aus dem Editor
 - ▶ Der Interpreter wertet sie aus („eval“)
 - ▶ Dabei fallen eventuell [Seiteneffekte](#) an, z.B. die Definition einer Variablen oder eine Ausgabe
 - ▶ Der Interpreter schreibt das Ergebnis zurück („print“)
- ▶ Wir schreiben im folgenden > vor Nutzereingaben, ==> vor Rückgabewerte des Interpreters:
> (+ 5 10)
==> 15
>(list 10 11 12)
==> (10 11 12)

Berechnen durch Ausrechnen

- ▶ Scheme-Programme werden durch Auswerten von **symbolischen Ausdrücken** (*s-expressions*) ausgeführt
 - ▶ Auswerten von Atomen:
 - ▶ Konstanten (Strings, Zeichen, Zahlen, ...) haben ihren natürlichen Wert
 - ▶ Identifier haben nur dann einen Wert, wenn Sie definiert sind

```
> 10
```

```
==> 10
```

```
> "Hallo"
```

```
==> "Hallo"
```

```
> hallo
```

```
;;; <unknown-location>: warning: possibly unbound variable
```

Berechnen durch Ausrechnen

- ▶ Scheme-Programme werden durch Auswerten von **symbolischen Ausdrücken** (*s-expressions*) ausgeführt
 - ▶ Auswerten von Atomen:
 - ▶ Konstanten (Strings, Zeichen, Zahlen, ...) haben ihren natürlichen Wert
 - ▶ Identifier haben nur dann einen Wert, wenn Sie definiert sind
- ```
> 10
==> 10
> "Hallo"
==> "Hallo"
> hallo
;;; <unknown-location>: warning: possibly unbound variable
```
- ▶ Auswerten von (normalen) Listen:
    - ▶ Zuerst werden (in beliebiger Reihenfolge) alle Listenelemente ausgewertet
    - ▶ Dann wird das erste Ergebnis als Funktion betrachtet und diese mit den anderen Elementen als Argument aufgerufen

# Beispiel

```
> +
==> $16 = #<procedure + (:optional _ _ . _)>
> 17
==> 17
> (* 3 7)
==> 21
> (+ 17 (* 3 7))
==> 38
```

# Besonderheiten

► Problem:

```
(if (= x 0) 1 (/ 10 x))
```

# Besonderheiten

- ▶ Problem:

```
(if (= x 0) 1 (/ 10 x))
```

- ▶ Bestimmte Ausdrücke werden anders behandelt („special forms“)

- ▶ Auswertung erfolgt nach speziellen Regeln

- ▶ Es werden nicht notwendigerweise alle Argumente ausgewertet

- ▶ Beispiel: `(if tst expr1 expr2)`

- ▶ *tst* wird auf jeden Fall ausgewertet

- ▶ Ist der Wert von *tst* nicht `#f`, so wird (nur) *expr*<sub>1</sub> ausgewertet, das Ergebnis ist der Wert des gesamten `if`-Ausdrucks

- ▶ Sonst wird (nur) *expr*<sub>2</sub> ausgewertet und als Ergebnis zurückgegeben

- ▶ Wichtiges Beispiel: `(quote expr)`

- ▶ `quote` gibt sein Argument unausgewertet zurück

- ▶ `> (1 2 3)` ==> Fehler (1 ist ja keine Funktion)

- ▶ `> (quote (1 2 3))` ==> (1 2 3)

- ▶ Kurzform: `'` (Hochkomma)

- ▶ `> '(1 2 3)` ==> (1 2 3)

## Definitionen (neue Namen/neue Werte)

`define` führt einen Namen global ein, reserviert (falls nötig) Speicher für ihn, und gibt ihm (optional) einen Wert.

- ▶ `(define a obj)`
  - ▶ Erschaffe den Namen `a` und binde den Wert `obj` an ihn
  - ▶ `> (define a 12)`
  - ▶ `> a ==> 12`
- ▶ `(define (f args) exprs)`
  - ▶ Definiere eine Funktion mit Namen `f`, den angegebenen Argumenten, und der Rechenvorschrift `exprs`
  - ▶ `> (define (plus3 x) (+ x 3))`
  - ▶ `> (plus3 10) ==> 13`
- ▶ Der Rückgabewert einer `define`-Anweisung ist undefiniert
- ▶ `defines` stehen typischerweise auf der obersten Ebene eines Programms

# Datentypen

- ▶ Scheme ist stark, aber dynamisch getypt
  - ▶ Jedes Objekt hat einen klaren Typ
  - ▶ Namen können Objekte verschiedenen Typs referenzieren
  - ▶ Manche Funktionen erwartet bestimmte Typen (z.B. + erwartet Zahlen), sonst: Fehler
  - ▶ Andere Funktionen sind generisch (z.B. equal?)
- ▶ Wichtige Datentypen
  - ▶ Boolsche Werte #t, #f
  - ▶ Zahlen (Ganze Zahlen, Bruchzahlen, Reals, Komplex)
  - ▶ Strings
  - ▶ Einzelne Zeichen (#\a ist das einzelne a)
  - ▶ Vektoren (*arrays*, *Felder*)
  - ▶ Prozeduren (oder Funktionen - ja, das ist ein Datentyp!)
  - ▶ Listen (eigentlich: cons-Paare)
  - ▶ Symbole (z.B. hallo, +, vector->list)

- ▶ `(= number1 ... numbern)`
  - ▶ `#t`, falls alle Werte gleich sind, `#f` sonst
  - ▶ `> (= 1 1) ==> #t`
- ▶ `(equal? obj1 ... objn)`
  - ▶ `#t`, falls die Objekte „gleich genug“ sind
  - ▶ `> (equal? '(1 2 3) '(1 2 3)) ==> #t`
- ▶ `+, -, *, /`: Normale Rechenoperationen (mit beliebig vielen Argumenten)
  - ▶ `> (+ 1 2 3) ==> 6`
  - ▶ `> (- 1 2 3) ==> -4`
  - ▶ `> (* 2 3 5) ==> 30`
  - ▶ `> (/ 1 2) ==> 1/2`

# Übung: Fibonacci

- ▶ Die Fibonacci-Zahlen sind definiert wie folgt:
  - ▶  $fib(0) = 0$
  - ▶  $fib(1) = 1$
  - ▶  $fib(n) = fib(n - 1) + fib(n - 2)$  für  $n > 1$
- ▶ Schreiben Sie eine Scheme-Funktion, die die Fibonacci-Zahlen berechnet
- ▶ Was sind die Werte von  $fib(5)$ ,  $fib(10)$ ,  $fib(20)$ ,  $fib(30)$ ,  $fib(40)$ ?



Leonardo Bonacci  
(c. 1170 – c. 1250)

Zurück B

# Lisp Jedi

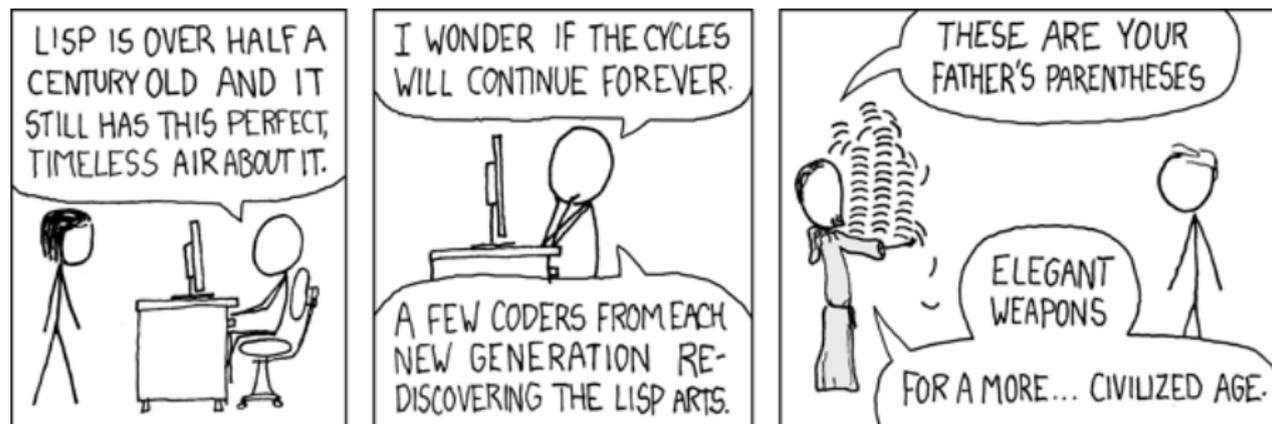


Image credit: <http://xkcd.com/297/>

# Listenverarbeitung

- ▶ '()
  - ▶ Die leere Liste
- ▶ (cons *obj list*) („constructor“)
  - ▶ Hänge *obj* als erstes Element in *list* ein und gib das Ergebnis zurück
  - ▶ > (cons 1 '(2 3 4)) ==> (1 2 3 4)
- ▶ (append *list*<sub>1</sub> *list*<sub>2</sub>)
  - ▶ Hänge *list*<sub>1</sub> und *list*<sub>2</sub> zusammen und gib das Ergebnis zurück
  - ▶ > (append '(1 2 3) '(4 5)) ==> (1 2 3 4 5)
- ▶ (car *list*) (Alternative: (first *list*))
  - ▶ Gib das erste Element von *list* zurück, ist *list* leer: Fehler
  - ▶ > (car '(1 2 3)) ==> 1
- ▶ (cdr *list*) (Alternative: (rest *list*))
  - ▶ Gib *list* ohne das erste Element zurück, ist *list* leer: Fehler
  - ▶ > (cdr '(1 2 3)) ==> (2 3)
- ▶ (null? *list*)
  - ▶ Gib #t zurück, wenn *list* leer ist
- ▶ (list *obj*<sub>1</sub> ... *obj*<sub>n</sub>)
  - ▶ Gib die Liste (*obj*<sub>1</sub> ... *obj*<sub>n</sub>) zurück



Contents of **A**ddress Part of **R**egister  
Contents of **D**ecrement Part of **R**egister

# Beispiel

- ▶ Berechne die Länge einer Liste
  - ▶ Die leere Liste hat Länge 0
  - ▶ Jede andere Liste hat Länge  $1 + \text{Länge der Liste ohne das erste Element}$

```
(define (len l)
 (if (null? l)
 0
 (+ 1 (len (cdr l)))
)
)
```

# Übung: Revert

- ▶ Schreiben Sie eine Funktion, die Listen umdreht:
  - ▶ `> (revert '()) ==> '()`
  - ▶ `> (revert '(1 2 3)) ==> '(3 2 1)`
  - ▶ `> (revert '(1 2 1 2)) ==> '(2 1 2 1)`
- ▶ Überlegen Sie dazu zuerst eine rekursive Definition der Operation!
- ▶ Bonusaufgabe: Schreiben Sie zwei Funktion `split` und `mix`
  - ▶ `split` macht aus einer Liste zwei, indem es abwechselnd Elemente verteilt:  
`(split '(1 2 3 4 5 6)) ==> ((1 3 5) (2 4 6))`
  - ▶ `mix` macht aus zwei Listen eine, indem es abwechselnd Elemente einfügt:  
`(mix '(1 2 3) '(6 5 4)) ==> (1 6 2 5 3 4)`
  - ▶ Überlegen Sie sinnvolles Verhalten, wenn die Listen unpassende Längen haben

# Tips zur Programmierung

- ▶ Editieren Sie Ihre Scheme-Programme in einem Texteditor
  - ▶ Die empfohlene Endung ist „.scm“
- ▶ Formatieren Sie die Programme lesbar
  - ▶ Emacs rückt mit [TAB] automatisch brauchbar ein
  - ▶ Ansonsten:
    - ▶ Alle direkten Teilausdrücke sollten gleich weit eingerückt sein
    - ▶ Unterausdrücke sollten weiter eingerückt sein, als der Gesamtausdruck
- ▶ Kopieren Sie einzelne Funktionen mit Copy&Paste
  - ▶ Unter X11 reicht meistens Markieren zum Kopieren, mittlerer Mausklick zum Pasten
  - ▶ Längere Programmfragmente: (load "myprog.scm")
- ▶ Starten Sie guile mittels rlwrap
  - ▶ Testen Sie dann direkt im Interpreter



# Übung: Mengenlehre in Scheme

- ▶ Repräsentieren Sie im folgenden Mengen als Listen
- ▶ Erstellen Sie Scheme-Funktionen für die folgenden Mengen-Operationen:
  - ▶ Einfügen:
    - ▶ `(insert 4 '(1 2 3)) ==> (1 2 3 4)` (Reihenfolge egal)
    - ▶ `(insert 2 '(1 2 3)) ==> (1 2 3)`
  - ▶ Vereinigung:
    - ▶ `(union '(1 2 3) '(3 4 5)) ==> (1 2 3 4 5)`
  - ▶ Schnittmenge:
    - ▶ `(intersection '(1 2 3) '(3 4 5)) ==> (3)`
  - ▶ Kartesisches Produkt:
    - ▶ `(kart '(1 2 3) '(a b c)) ==> ((1 a) (2 a) (3 a) (1 b) (2 b) (3 b) (1 c) (2 c) (3 c))`
  - ▶ Potenzmenge:
    - ▶ `(powerset '(1 2 3)) ==> (() (3) (2) (2 3) (1) (1 3) (1 2) (1 2 3))`
- ▶ Tipp: Hilfsfunktionen machen die Aufgabe einfacher!
- ▶ Bonus: Implementieren Sie eine Funktion, die die Verkettung von zwei Relationen realisieren!

# Funktionale Programmierung?

- ▶ Ein Programm besteht aus Funktionen
  - ▶ Die Ausführung entspricht der Berechnung eines Funktionswertes
  - ▶ Andere Effekte (I/O, Änderungen von Objekten, neue Definitionen, ...) heißen **Seiteneffekte**
- ▶ Idealerweise hat die Ausführung keine Seiteneffekte
  - ▶ Statt ein Objekt zu ändern, gib ein neues Objekt mit den gewünschten Eigenschaften zurück
    - ▶ Aber: Aus Effizienzgründen manchmal aufgeweicht
  - ▶ Ein-/Ausgabe sind in Scheme immer Seiteneffekte
- ▶ Funktionen sind Werte
  - ▶ Funktionen können als Parameter übergeben werden
  - ▶ Funktionen können dynamisch erzeugt werden

# Funktionsaufrufe

```
(define (sumsquare x y)
 (+ (* x x) y))
```

```
> (sumsquare 3 6)
=>> 15
```

Was passiert beim Aufruf (sumsquare 3 6)?

- ▶ Für die **formalen Parameter** x und y werden neue, temporäre Variablen angelegt
- ▶ Die **konkreten Parameter** 3 und 6 werden in diesen gespeichert
- ▶ Der **Rumpf** der Funktion wird in der so erweiterten Umgebung ausgewertet
- ▶ Der Wert des letzten (hier: einzigen) Ausdrucks des Rumpfes wird zurückgegeben

# Speichemodell und Variablen (1)

- ▶ Objekte („Werte“) liegen (konzeptionell) im **Speicher**
  - ▶ Lebensdauer von Objekten ist theoretisch unbegrenzt
- ▶ **Variablen** sind Namen, die Orte im Speicher bezeichnen
  - ▶ Sprich: „Variable X ist an den Speicherort Y **gebunden**“
  - ▶ Der **Wert** der Variable ist der an diesem Ort gespeicherte Wert (falls es einen gibt)
    - ▶ Sprich: „Variable X ist an den Wert Y gebunden“ (ja, das ist potentiell verwirrend)
  - ▶ Der Wert kann sich ändern, ohne dass sich der Name oder der Speicherort ändern (Seiteneffekt!)
- ▶ Lebensdauer von Variablen:
  - ▶ Unbegrenzt oder
  - ▶ Während der Programmausführung in einem bestimmten **Sichtbarkeitsbereich**
- ▶ Wenn ein neue Variable entsteht, wird für diese Speicher reserviert und als benutzt markiert



# Auswertung und Umgebung

- ▶ Umgebung: Alle „im Moment“ sichtbaren **Variablen** mit ihren assoziierten Werten

- ▶ Initiale Umgebung: Beim Start von Scheme

- ▶ Neue Variablen:

- ▶ Dauerhaft durch `define` auf Top-Level

- ▶ Sichtbarkeit des Namens ab Ende des `define`-Ausdrucks

- ▶ Temporäre Namen:

- ▶ Z.B. Parameter in Funktionen

```
(define (add4 x) <--- Bei der _Ausfuehrung_ ist
 x ab hier definiert
```

```
 (+ x 4)
) <--- ...und bis hier
```

- ▶ Später mehr dazu

# Auswertung und Umgebung

- ▶ Umgebung: Alle „im Moment“ sichtbaren **Variablen** mit ihren assoziierten Werten

- ▶ Initiale Umgebung: Beim Start von Scheme

- ▶ Neue Variablen:

- ▶ Dauerhaft durch `define` auf Top-Level

- ▶ Sichtbarkeit des Namens ab Ende des `define`-Ausdrucks

- ▶ Temporäre Namen:

- ▶ Z.B. Parameter in Funktionen

```
(define (add4 x) <--- Bei der _Ausfuehrung_ ist
 x ab hier definiert
```

```
 (+ x 4)
) <--- ...und bis hier
```

- ▶ Später mehr dazu

Eine Sicht: Entwicklung in Scheme/Lisp reichert die initiale Umgebung mit Funktionen an, bis die zu lösende Aufgabe trivial wird!

# Sequenzen

- ▶ An manchen Stellen erlaubt Scheme **Sequenzen** von Ausdrücken
  - ▶ Sequenzen werden der Reihe nach ausgewertet
  - ▶ Wert der Sequenz ist der Wert des letzten Ausdrucks
- ▶ Sequenzen sind z.B. im Rumpf einer Funktion erlaubt
- ▶ Beispiel:

```
(define (mach-was x)
 (+ 10 x)
 (display "Ich mache was")
 (newline)
 (* x x))
```

```
> (mach-was 10)
Ich mache was
=> 100
```

## Special Form: begin

- ▶ `begin` ermöglicht Sequenzen überall
  - ▶ Argument eines `begin` blocks ist eine Sequenz
  - ▶ Wert ist der Wert der Sequenz (also des letzten Ausdrucks)
- ▶ Beispiel

```
> (+ 10
 (begin (display "Hallo")
 (newline)
 (+ 10 10)
 (* 10 10)))
```

## Special Form: begin

- ▶ `begin` ermöglicht Sequenzen überall
  - ▶ Argument eines `begin` blocks ist eine Sequenz
  - ▶ Wert ist der Wert der Sequenz (also des letzten Ausdrucks)
- ▶ Beispiel

```
> (+ 10
 (begin (display "Hallo")
 (newline)
 (+ 10 10)
 (* 10 10)))
```

⇒ 110

## Special Form: begin

- ▶ `begin` ermöglicht Sequenzen überall
  - ▶ Argument eines `begin` blocks ist eine Sequenz
  - ▶ Wert ist der Wert der Sequenz (also des letzten Ausdrucks)
- ▶ Beispiel

```
> (+ 10
 (begin (display "Hallo")
 (newline)
 (+ 10 10)
 (* 10 10)))
```

⇒ 110

`begin` und Sequenzen sind i.a. nur für I/O und andere Seiteneffekte notwendig. Sie durchbrechen das funktionale Paradigma!

# Special Form: cond

- ▶ `cond` ermöglicht die Auswahl aus mehreren Alternativen
- ▶ Ein `cond`-Ausdruck besteht aus dem Schlüsselwort `cond` und einer Reihe von **Klauseln**
  - ▶ Jede Klausel ist eine Sequenz von Ausdrücken. Der erste Ausdruck der Sequenz ist der **Test** der Klausel
  - ▶ Bei der letzten Klausel darf der Test auch `else` sein (dann muss mindestens ein weiterer Ausdruck folgen)
- ▶ Semantik:
  - ▶ Die Tests werden der Reihe nach evaluiert
  - ▶ Ist der Wert eines Testes ungleich `#f`, so wird die Sequenz evaluiert
    - ▶ Wert des `cond`-Ausdrucks ist der Wert der Sequenz (also der Wert ihres letzten Ausdrucks)
    - ▶ Alle weiteren Klauseln werden ignoriert
    - ▶ Der Test `else` wird wie die Konstante `#t` behandelt (ist also immer wahr)

## cond Beispiele

```
> (define x 10)
> (cond ((= x 9) "Neun")
 ((= x 10) "Zehn")
 ((= x 11) "Elf")
 (else "Sonstwas"))
```

## cond Beispiele

```
> (define x 10)
> (cond ((= x 9) "Neun")
 ((= x 10) "Zehn")
 ((= x 11) "Elf")
 (else "Sonstwas"))
```

⇒ "Zehn"

## cond Beispiele

```
> (define x 10)
> (cond ((= x 9) "Neun")
 ((= x 10) "Zehn")
 ((= x 11) "Elf")
 (else "Sonstwas"))
```

⇒ "Zehn"

```
> (cond ((= x 9) (display "Neun")
 (newline)
 9)
 (else (display "Sonstwas")
 (newline)
 (+ x 5))))
```

## cond Beispiele

```
> (define x 10)
> (cond ((= x 9) "Neun")
 ((= x 10) "Zehn")
 ((= x 11) "Elf")
 (else "Sonstwas"))
```

⇒ "Zehn"

```
> (cond ((= x 9) (display "Neun")
 (newline)
 9)
 (else (display "Sonstwas")
 (newline)
 (+ x 5))))
```

Sonstwas

⇒ 15

## Special Form: and

- ▶ Syntax: `(and expr1 ...)`
  - ▶ Wertet die Ausdrücke der Reihe nach aus
  - ▶ Ist der Wert eines Ausdruck `#f`, so gib `#f` zurück - die weiteren Ausdrücke werden nicht ausgewertet!
  - ▶ Sonst gib den Wert des letzten Ausdrucks zurück
- ▶ Beispiele:
  - > `(and (> 2 3) (+ 3 4))`

## Special Form: and

- ▶ Syntax: (and *expr*<sub>1</sub> ...)
  - ▶ Wertet die Ausdrücke der Reihe nach aus
  - ▶ Ist der Wert eines Ausdruck #f, so gib #f zurück - die weiteren Ausdrücke werden nicht ausgewertet!
  - ▶ Sonst gib den Wert des letzten Ausdrucks zurück
- ▶ Beispiele:

```
> (and (> 2 3) (+ 3 4))
```

```
⇒ #f
```

```
> (and (< 2 3) (+ 3 4))
```

## Special Form: and

- ▶ Syntax: (and *expr*<sub>1</sub> ...)
  - ▶ Wertet die Ausdrücke der Reihe nach aus
  - ▶ Ist der Wert eines Ausdruck #f, so gib #f zurück - die weiteren Ausdrücke werden nicht ausgewertet!
  - ▶ Sonst gib den Wert des letzten Ausdrucks zurück
- ▶ Beispiele:

> (and (> 2 3) (+ 3 4))

⇒ #f

> (and (< 2 3) (+ 3 4))

⇒ 7

## Special Form: or

- ▶ Syntax: (or *expr*<sub>1</sub> ...)
  - ▶ Wertet die Ausdrücke der Reihe nach aus
  - ▶ Ist der Wert eines Ausdruck ungleich #f, so gib diesen Wert zurück - die weiteren Ausdrücke werden nicht ausgewertet!
  - ▶ Sonst gib #f zurück
- ▶ Beispiele
  - > (define x 0)
  - > (or (= x 0) (/ 100 x))

## Special Form: or

- ▶ Syntax: (or *expr*<sub>1</sub> ...)
  - ▶ Wertet die Ausdrücke der Reihe nach aus
  - ▶ Ist der Wert eines Ausdruck ungleich #f, so gib diesen Wert zurück - die weiteren Ausdrücke werden nicht ausgewertet!
  - ▶ Sonst gib #f zurück
- ▶ Beispiele

```
> (define x 0)
> (or (= x 0) (/ 100 x))
```

==> #t

```
> (define x 4)
> (or (= x 0) (/ 100 x))
```

## Special Form: or

- ▶ Syntax: (or *expr*<sub>1</sub> ...)
  - ▶ Wertet die Ausdrücke der Reihe nach aus
  - ▶ Ist der Wert eines Ausdruck ungleich #f, so gib diesen Wert zurück - die weiteren Ausdrücke werden nicht ausgewertet!
  - ▶ Sonst gib #f zurück
- ▶ Beispiele

```
> (define x 0)
> (or (= x 0) (/ 100 x))
```

```
==> #t
```

```
> (define x 4)
> (or (= x 0) (/ 100 x))
```

```
==> 25
```

# Definierte Funktion: not

- ▶ not ist in der initialen Umgebung als eine normale Funktion definiert
- ▶ Syntax: (not *expr*)
- ▶ Semantik:
  - ▶ Ist der Wert von *expr* #f, so gib #t zurück
  - ▶ Sonst gib #f zurück
- ▶ Beispiele

> (not 1)

==> #f

> (not (> 3 4))

==> #t

> (not "")

==> #f

> (not +)

==> #f

## Special Form: let

- ▶ let führt temporäre Variablen für Zwischenergebnisse ein
- ▶ Syntax: Das Schlüsselwort let wird gefolgt von einer Liste von Bindungen und einem Rumpf
  - ▶ Eine Bindung besteht aus einer Variablen und einem Ausdruck (dem Initialisierer)
  - ▶ Der Rumpf ist eine Sequenz von Scheme-Ausdrücken
- ▶ Semantik von let:
  - ▶ Die Initialisierer werden in beliebiger Reihenfolge ausgewertet, die Ergebnisse gespeichert
  - ▶ Die Variablen werden angelegt und an die Speicherstellen gebunden, die die Ergebnisse der Initialisierer enthalten
    - ▶ Gültigkeitsbereich der Variablen: Rumpf des let-Ausdrucks
  - ▶ Der Rumpf wird in der um die neuen Variablen erweiterten Umgebung ausgewertet
    - ▶ Wert: Wert der Sequenz, also des letzten Ausdrucks

## Beispiel für let

```
> (let ((a 10)
 (b 20)
 (c 30))
 (+ a b c))
⇒ 60
```

## Beispiel für let

```
> (let ((a 10)
 (b 20)
 (c 30))
 (+ a b c))
```

⇒ 60

```
> (let ((a +)
 (b *))
 (a (b 10 20) (b 5 10)))
```

## Beispiel für let

```
> (let ((a 10)
 (b 20)
 (c 30))
 (+ a b c))
```

⇒ 60

```
> (let ((a +)
 (b *))
 (a (b 10 20) (b 5 10)))
```

⇒ 250

## Special Form: `let*`

- ▶ `let*` führt ebenfalls temporäre Variablen für Zwischenergebnisse ein
- ▶ Syntax: Wie bei `let`
- ▶ Semantik von `let*`:
  - ▶ Ähnlich wie `let`, aber die Bindungen werden der Reihe nach ausgewertet
  - ▶ Jede spätere Bindung kann auf die vorher stehenden Variablen zugreifen
- ▶ Beispiel:

```
(let* ((a 10)
 (b (+ a 20)))
 (* a b))
```

## Special Form: let\*

- ▶ `let*` führt ebenfalls temporäre Variablen für Zwischenergebnisse ein
- ▶ Syntax: Wie bei `let`
- ▶ Semantik von `let*`:
  - ▶ Ähnlich wie `let`, aber die Bindungen werden der Reihe nach ausgewertet
  - ▶ Jede spätere Bindung kann auf die vorher stehenden Variablen zugreifen
- ▶ Beispiel:

```
(let* ((a 10)
 (b (+ a 20)))
 (* a b))
```

⇒ 300

# Übung: Alter Wein in Neuen Schläuchen

- ▶ Lösen sie die Fibonacci-Zahlen-Aufgabe eleganter
- ▶ Erinnerung:
  - ▶  $fib(0) = 0$
  - ▶  $fib(1) = 1$
  - ▶  $fib(n) = fib(n - 1) + fib(n - 2)$  für  $n > 1$ :q!
- ▶ Finden Sie für möglichst viele der Mengenfunktionen elegantere oder effizientere Implementierung.
  - ▶ Bearbeiten Sie dabei mindestens die Potenzmengenbildung.

# Sortieren durch Einfügen

- ▶ Ziel: Sortieren einer Liste von Zahlen
- ▶ Also:
  - ▶ Eingabe: Eine Liste von Zahlen
  - ▶ Z.B. '(2 4 1 4 6 0 12 3 17)
  - ▶ Ausgabe: Eine Liste, die die selben Zahlen in aufsteigender Reihenfolge enthält
  - ▶ Im Beispiel: '(0 1 2 3 4 4 6 12 17)
- ▶ Idee: Baue eine neue Liste, in die die Elemente aus der alten Liste sortiert eingefügt werden

| <b>Abzuarbeiten</b>    | <b>(Zwischen-)ergebnis</b> |
|------------------------|----------------------------|
| '(2 4 1 4 6 0 12 3 17) | '()                        |
| '(4 1 4 6 0 12 3 17)   | '(2)                       |
| '(1 4 6 0 12 3 17)     | '(2 4)                     |
| '(4 6 0 12 3 17)       | '(1 2 4)                   |
| '(6 0 12 3 17)         | '(1 2 4 4)                 |
| ...                    | ...                        |
| '()                    | '(0 1 2 3 4 4 6 12 17)     |

# InsertSort in Scheme

- ▶ Ziel: Sortieren einer Liste von Zahlen
- ▶ Verfahren: Füge alle Elemente der Reihe nach sortiert in eine neue Liste ein
- ▶ Funktion 1: Sortiertes Einfügen eines Elements
  - ▶ `(insert k lst)`
    - ▶ Fall 1: `lst` ist leer
    - ▶ Fall 2a: `k` ist kleiner als `(car lst)`
    - ▶ Fall 2b: `k` ist nicht kleiner als `(car lst)`
- ▶ Funktion 2: Sortiertes Einfügen aller Elemente
  - ▶ Fall 1: Liste ist leer
  - ▶ Fall 2: Sortiere `(cdr lst)`, füge `(car lst)` ein

Anmerkung: Durch die übliche Rekursion wird die sortierte Liste in Scheme von hinten nach vorne aufgebaut - erst wird der Rest sortiert, dann das erste Element einsortiert!

THE CLASSIC WORK  
NEWLY UPDATED AND REVISED

## The Art of Computer Programming

VOLUME 3  
Sorting and Searching  
Second Edition

DONALD E. KNUTH

# Übung: InsertSort

- ▶ Erstellen Sie eine Funktion (`insert k lst`)
  - ▶ Eingaben: Eine Zahl `k` und ...
  - ▶ eine bereits sortierte Liste von Zahlen `lst`
  - ▶ Wert: Die Liste, die entsteht, wenn man `k` an der richtigen Stelle in `lst` einfügt
- ▶ Erstellen Sie eine Funktion (`isort lst`)
  - ▶ (`isort lst`) soll `lst` in aufsteigender Reihenfolge zurückgeben
  - ▶ Verwenden Sie Sortieren durch Einfügen als Algorithmus
- ▶ Beispiellisten können (in Guile) mit folgender Funktion generiert werden (auch auf <http://wwwlehre.dhbw-stuttgart.de/~sschulz/lgli2014.html>):

```
(define (randlist n)
 (if (= n 0)
 '()
 (cons (random 10000) (randlist (- n 1)))))
```

# Sortieren allgemeiner

- ▶ Problem: Unser `isort` sortiert
  1. Nur Zahlen
  2. Nur aufsteigend
- ▶ Lösung?

# Sortieren allgemeiner

- ▶ Problem: Unser `isort` sortiert
  1. Nur Zahlen
  2. Nur aufsteigend
- ▶ Lösung?
  - ▶ Scheme ist **funktional**
  - ▶ Wir können die „Kleiner-Funktion“ als Parameter übergeben!

```
> (isort '(2 5 3 1))
```

```
⇒ (1 2 3 5)
```

```
> (isort_g '(2 5 3 1) <)
```

```
⇒ (1 2 3 5)
```

```
> (isort_g '(2 5 3 1) >)
```

```
⇒ (5 3 2 1)
```

```
> (isort_g '("Hallo" "Tschuess" "Ende") string >?)
```

```
⇒ ("Tschuess" "Hallo" "Ende")
```

## Übung: Generisches Sortieren

- ▶ Wandeln Sie die Funktionen `insert` und `isort` so ab, dass Sie als zweites bzw. drittes Argument eine Vergleichsfunktion akzeptieren
- ▶ Bonus: Sortieren Sie eine Liste von Zahlen lexikographisch (also  $1 < 11 < 111 < 1111 < \dots < 2 < 22 < 222 \dots$ )
  - ▶ Die Funktion `number->string` könnte hilfreich sein!

- ▶ Alternatives Sortierverfahren: *Mergesort* („Sortieren durch Vereinen/Zusammenfügen“)
- ▶ Idee: Sortiere Teillisten und füge diese dann sortiert zusammen
  - ▶ Schritt 1: Teile die Liste rekursiv in jeweils zwei etwa gleichgroße Teile, bis die Listen jeweils die Länge 0 oder 1 haben
    - ▶ Listen der Länge 0 oder 1 sind immer sortiert!
  - ▶ Schritt 2: Füge die sortierten Listen sortiert zusammen
    - ▶ Vergleiche die ersten Elemente der beiden Listen
    - ▶ Nimm das kleinere als erste Element des Ergebnisses
    - ▶ Hänge dahinter das Ergebnis des Zusammenfügens der Restlisten

## Beispiel: Zusammenfügen

- ▶ Beispiel:

| Liste 1    | Liste 2 | Ergebnisliste |
|------------|---------|---------------|
| (1 3 8 10) | (2 3 7) | ( )           |

## Beispiel: Zusammenfügen

► Beispiel:

| Liste 1    | Liste 2 | Ergebnisliste |
|------------|---------|---------------|
| (1 3 8 10) | (2 3 7) | ( )           |
| (3 8 10)   | (2 3 7) | (1)           |

## Beispiel: Zusammenfügen

► Beispiel:

| Liste 1    | Liste 2 | Ergebnisliste |
|------------|---------|---------------|
| (1 3 8 10) | (2 3 7) | ()            |
| (3 8 10)   | (2 3 7) | (1)           |
| (3 8 10)   | (3 7)   | (1 2)         |

## Beispiel: Zusammenfügen

► Beispiel:

| Liste 1    | Liste 2 | Ergebnisliste |
|------------|---------|---------------|
| (1 3 8 10) | (2 3 7) | ()            |
| (3 8 10)   | (2 3 7) | (1)           |
| (3 8 10)   | (3 7)   | (1 2)         |
| (8 10)     | (3 7)   | (1 2 3)       |

## Beispiel: Zusammenfügen

► Beispiel:

| Liste 1    | Liste 2 | Ergebnisliste |
|------------|---------|---------------|
| (1 3 8 10) | (2 3 7) | ()            |
| (3 8 10)   | (2 3 7) | (1)           |
| (3 8 10)   | (3 7)   | (1 2)         |
| (8 10)     | (3 7)   | (1 2 3)       |
| (8 10)     | (7)     | (1 2 3 3)     |

## Beispiel: Zusammenfügen

► Beispiel:

| Liste 1    | Liste 2 | Ergebnisliste |
|------------|---------|---------------|
| (1 3 8 10) | (2 3 7) | ()            |
| (3 8 10)   | (2 3 7) | (1)           |
| (3 8 10)   | (3 7)   | (1 2)         |
| (8 10)     | (3 7)   | (1 2 3)       |
| (8 10)     | (7)     | (1 2 3 3)     |
| (8 10)     | ()      | (1 2 3 3 7)   |

# Beispiel: Zusammenfügen

► Beispiel:

| Liste 1    | Liste 2 | Ergebnisliste    |
|------------|---------|------------------|
| (1 3 8 10) | (2 3 7) | ()               |
| (3 8 10)   | (2 3 7) | (1)              |
| (3 8 10)   | (3 7)   | (1 2)            |
| (8 10)     | (3 7)   | (1 2 3)          |
| (8 10)     | (7)     | (1 2 3 3)        |
| (8 10)     | ()      | (1 2 3 3 7)      |
| ()         | ()      | (1 2 3 3 7 8 10) |

## Beispiel: Zusammenfügen

► Beispiel:

| Liste 1    | Liste 2 | Ergebnisliste    |
|------------|---------|------------------|
| (1 3 8 10) | (2 3 7) | ()               |
| (3 8 10)   | (2 3 7) | (1)              |
| (3 8 10)   | (3 7)   | (1 2)            |
| (8 10)     | (3 7)   | (1 2 3)          |
| (8 10)     | (7)     | (1 2 3 3)        |
| (8 10)     | ()      | (1 2 3 3 7)      |
| ()         | ()      | (1 2 3 3 7 8 10) |

### Spezialfälle:

- Liste 1 ist leer
- Liste 2 ist leer
- Liste 1 und Liste 2 haben das gleiche erste Element!

# Mergesort rekursiv

- ▶ Rekursiver Algorithmus:
  - ▶ Input: Zu sortierende Liste  $lst$
  - ▶ Fall 1:  $lst$  ist leer oder hat genau ein Element
    - ▶ Dann ist  $lst$  sortiert  $\implies$  gib  $lst$  zurück
  - ▶ Fall 2:  $lst$  hat mindestens zwei Elemente
    - ▶ Dann: Teile  $lst$  in zwei (kleinere) Teillisten
    - ▶ Sortiere diese rekursiv mit Mergesort
    - ▶ Füge die Ergebnislisten zusammen
- ▶ Beispiel:  
(2 3 1 7 5)                      Zu sortierende Liste

# Mergesort rekursiv

- ▶ Rekursiver Algorithmus:
  - ▶ Input: Zu sortierende Liste  $lst$
  - ▶ Fall 1:  $lst$  ist leer oder hat genau ein Element
    - ▶ Dann ist  $lst$  sortiert  $\implies$  gib  $lst$  zurück
  - ▶ Fall 2:  $lst$  hat mindestens zwei Elemente
    - ▶ Dann: Teile  $lst$  in zwei (kleinere) Teillisten
    - ▶ Sortiere diese rekursiv mit Mergesort
    - ▶ Füge die Ergebnislisten zusammen
- ▶ Beispiel:
  - (2 3 1 7 5)                      Zu sortierende Liste
  - ((2 1 5) (3 7))                 1. Split

# Mergesort rekursiv

- ▶ Rekursiver Algorithmus:
  - ▶ Input: Zu sortierende Liste  $lst$
  - ▶ Fall 1:  $lst$  ist leer oder hat genau ein Element
    - ▶ Dann ist  $lst$  sortiert  $\implies$  gib  $lst$  zurück
  - ▶ Fall 2:  $lst$  hat mindestens zwei Elemente
    - ▶ Dann: Teile  $lst$  in zwei (kleinere) Teillisten
    - ▶ Sortiere diese rekursiv mit Mergesort
    - ▶ Füge die Ergebnislisten zusammen
- ▶ Beispiel:

|                           |                      |
|---------------------------|----------------------|
| (2 3 1 7 5)               | Zu sortierende Liste |
| ((2 1 5) (3 7))           | 1. Split             |
| ((((2 5) (1)) ((3) (7)))) | 2. und 3. Split      |

# Mergesort rekursiv

- ▶ Rekursiver Algorithmus:
  - ▶ Input: Zu sortierende Liste 1st
  - ▶ Fall 1: 1st ist leer oder hat genau ein Element
    - ▶ Dann ist 1st sortiert  $\implies$  gib 1st zurück
  - ▶ Fall 2: 1st hat mindestens zwei Elemente
    - ▶ Dann: Teile 1st in zwei (kleinere) Teillisten
    - ▶ Sortiere diese rekursiv mit Mergesort
    - ▶ Füge die Ergebnislisten zusammen

- ▶ Beispiel:

|                               |                      |
|-------------------------------|----------------------|
| (2 3 1 7 5)                   | Zu sortierende Liste |
| ((2 1 5) (3 7))               | 1. Split             |
| ((((2 5) (1)) ((3) (7))))     | 2. und 3. Split      |
| (((((2) (5)) (1)) ((3) (7)))) | 4. Split             |

---

# Mergesort rekursiv

- ▶ Rekursiver Algorithmus:
  - ▶ Input: Zu sortierende Liste  $l_{st}$
  - ▶ Fall 1:  $l_{st}$  ist leer oder hat genau ein Element
    - ▶ Dann ist  $l_{st}$  sortiert  $\implies$  gib  $l_{st}$  zurück
  - ▶ Fall 2:  $l_{st}$  hat mindestens zwei Elemente
    - ▶ Dann: Teile  $l_{st}$  in zwei (kleinere) Teillisten
    - ▶ Sortiere diese rekursiv mit Mergesort
    - ▶ Füge die Ergebnislisten zusammen

- ▶ Beispiel:

(2 3 1 7 5)

Zu sortierende Liste

((2 1 5) (3 7))

1. Split

(((2 5) (1)) ((3) (7)))

2. und 3. Split

((((2) (5)) (1)) ((3) (7)))

4. Split

---

((((2 5) (1)) (3 7)))

1. und 2. Zusammenfügen

# Mergesort rekursiv

- ▶ Rekursiver Algorithmus:
  - ▶ Input: Zu sortierende Liste 1st
  - ▶ Fall 1: 1st ist leer oder hat genau ein Element
    - ▶ Dann ist 1st sortiert  $\implies$  gib 1st zurück
  - ▶ Fall 2: 1st hat mindestens zwei Elemente
    - ▶ Dann: Teile 1st in zwei (kleinere) Teillisten
    - ▶ Sortiere diese rekursiv mit Mergesort
    - ▶ Füge die Ergebnislisten zusammen

- ▶ Beispiel:

|                               |                         |
|-------------------------------|-------------------------|
| (2 3 1 7 5)                   | Zu sortierende Liste    |
| ((2 1 5) (3 7))               | 1. Split                |
| ((((2 5) (1)) ((3) (7))))     | 2. und 3. Split         |
| (((((2) (5)) (1)) ((3) (7)))) | 4. Split                |
| <hr/>                         |                         |
| ((((2 5) (1)) (3 7)))         | 1. und 2. Zusammenfügen |
| ((1 2 5) (3 7))               | 3. Zusammenfügen        |

# Mergesort rekursiv

- ▶ Rekursiver Algorithmus:
  - ▶ Input: Zu sortierende Liste 1st
  - ▶ Fall 1: 1st ist leer oder hat genau ein Element
    - ▶ Dann ist 1st sortiert  $\implies$  gib 1st zurück
  - ▶ Fall 2: 1st hat mindestens zwei Elemente
    - ▶ Dann: Teile 1st in zwei (kleinere) Teillisten
    - ▶ Sortiere diese rekursiv mit Mergesort
    - ▶ Füge die Ergebnislisten zusammen

- ▶ Beispiel:

|                               |                         |
|-------------------------------|-------------------------|
| (2 3 1 7 5)                   | Zu sortierende Liste    |
| ((2 1 5) (3 7))               | 1. Split                |
| (( (2 5) (1) ) ((3) (7)))     | 2. und 3. Split         |
| (( ((2) (5)) (1) ) ((3) (7))) | 4. Split                |
| <hr/>                         |                         |
| (( (2 5) (1) ) (3 7))         | 1. und 2. Zusammenfügen |
| ((1 2 5) (3 7))               | 3. Zusammenfügen        |
| (1 2 3 5 7)                   | 4. und Fertig!          |

# Übung: Mergesort

- ▶ Implementieren Sie *Mergesort*
  - ▶ ... für Listen von Zahlen mit fester Ordnung
  - ▶ Generisch (mit Vergleichsfunktion als Parameter)
- ▶ Tipps:
  - ▶ Implementieren Sie zunächst eine Funktion `split`, die eine Liste bekommt, und diese in zwei Teillisten aufteilt. Rückgabewert ist eine Liste mit den beiden Teilen!
  - ▶ Implementieren Sie eine Funktion `merge`, die zwei sortierte Listen zu einer sortierten List zusammenfügt.

Zurück B

Zurück C

- ▶ Idee: Problemlösung durch Fallunterscheidung
  - ▶ Fall 1: „Einfach“ - das Problem kann direkt gelöst werden
  - ▶ Fall 2: „Komplex“
    - ▶ Zerlege das Problem in einfachere Unterprobleme (oder identifiziere ein einzelnes Unterproblem)
    - ▶ Löse diese Unterprobleme, in der Regel rekursiv (d.h. durch Anwendung des selben Verfahrens)
    - ▶ Kombiniere die Lösungen der Unterprobleme zu einer Gesamtlösung (falls notwendig)

- ▶ Idee: Problemlösung durch Fallunterscheidung
  - ▶ Fall 1: „Einfach“ - das Problem kann direkt gelöst werden
  - ▶ Fall 2: „Komplex“
    - ▶ Zerlege das Problem in einfachere Unterprobleme (oder identifiziere ein einzelnes Unterproblem)
    - ▶ Löse diese Unterprobleme, in der Regel rekursiv (d.h. durch Anwendung des selben Verfahrens)
    - ▶ Kombiniere die Lösungen der Unterprobleme zu einer Gesamtlösung (falls notwendig)
- ▶ Beispiel: (is-element? element lst)
  - ▶ Einfacher Fall: lst ist leer (dann immer #f)
  - ▶ Komplexer Fall: lst ist nicht leer
  - ▶ Zerlegung:
    - ▶ Ist element das erste Element von lst
    - ▶ Ist element im Rest von lst
  - ▶ Kombination: Logisches oder der Teillösungen

## Rekursion – is-element (1)

```
(define (is-element? x set)
 (if (null? set)
 #f
 (if (equal? x (car set))
 #t
 (is-element? x (cdr set)))))
```

- ▶ Beispiel: (is-element? element lst)
  - ▶ Einfacher Fall: lst ist leer (dann immer #f)
  - ▶ Komplexer Fall: lst ist nicht leer
  - ▶ Zerlegung:
    - ▶ Ist element das erste Element von lst
    - ▶ Ist element im Rest von lst
  - ▶ Kombination: Logisches **oder** der Teillösungen

## Rekursion – is-element (2)

```
(define (is-element? x set)
 (if (null? set)
 #f
 (or (equal? x (car set))
 (is-element? x (cdr set)))))
```

## Rekursion – is-element (2)

```
(define (is-element? x set)
 (if (null? set)
 #f
 (or (equal? x (car set))
 (is-element? x (cdr set)))))
```

- ▶ Standard-Muster: Bearbeite alle Elemente einer Liste
  - ▶ Bearbeite erstes Element (equal? x (car set))
  - ▶ Bearbeite Rest: (is-element? x (cdr set))
- ▶ Beachte: Der Name – hier is-element? – wird in der ersten Zeile angelegt
  - ▶ Er kann also in der 5. Zeile referenziert werden
  - ▶ Er hat einen Wert erst, wenn das define abgeschlossen ist
  - ▶ Aber dieser Wert wird erst gebraucht, wenn die Funktion mit **konkreten Parametern** aufgerufen wird!

# Übung: Listenvervielfachung

- ▶ Schreiben Sie eine Funktion (`scalar-mult liste faktor`)
  - ▶ `liste` ist eine Liste von Zahlen
  - ▶ `faktor` ist eine Zahl
  - ▶ Ergebnis ist eine Liste, in der die mit `faktor` multiplizierten Werte aus `liste` stehen
  - ▶ Beispiele:

```
> (scalar-mult '(1 2 3) 5)
```

```
==> (5 10 15)
```

```
> (scalar-mult '(7 11 13) 0.5)
```

```
==> (3.5 5.5 6.5)
```

```
> (scalar-mult '() 42)
```

```
==> ()
```

# Input und Output

- ▶ Standard-Scheme unterstützt Eingabe und Ausgabe von Zeichen über (virtuelle) Geräte
  - ▶ Geräte werden durch **Ports** abstrahiert
  - ▶ Jeder Port ist **entweder** ...
    - ▶ Eingabe-Port oder
    - ▶ Ausgabe-Port
  - ▶ Ports sind ein eigener Datentyp in Scheme
    - ▶ `(port? obj)` ist wahr, gdw. `obj` ein Port ist
- ▶ Ports können mit Dateien oder Terminals assoziiert sein
  - ▶ Wird für eine Operation kein besonderer Port angegeben, so werden folgende Ports verwendet:
    - ▶ Eingabe: `(current-input-port)`
    - ▶ Ausgabe: `(current-output-port)`
  - ▶ Beide sind per Default an das Eingabe-Terminal gebunden

# Ein- und Ausgabebefehle (1)

- ▶ Die wichtigsten Scheme-Befehle für ein- und Ausgabe sind:
  - ▶ `(read port)` (das Argument ist optional)
    - ▶ Liest ein Scheme-Objekt in normaler Scheme-Syntax von dem angegebenen Port
  - ▶ `(write obj port)` (das 2. Argument ist optional)
    - ▶ Schreibe ein Scheme-Objekt in normaler Scheme-Syntax auf den angegebenen Port
- ▶ Mit `write` geschriebene Objekte können mit `read` wieder gelesen werden
  - ▶ So ist es sehr einfach, interne Datenstrukturen zu speichern und zu laden!
  - ▶ Stichworte: *Persistierung/Serialisierung* (vergleiche z.B. Java Hibernate)

## Ein- und Ausgabebefehle (2)

- ▶ Weitere Befehle:
  - ▶ `(display obj port)` (das 2. Argument ist optional)
    - ▶ Schreibt eine „mensch-lesbare“ Form von *obj*
    - ▶ Strings ohne Anführungsstriche
    - ▶ Zeichen als einfache Zeichen
  - ▶ `(newline port)`
    - ▶ Schreibt einen Zeilenumbruch
  - ▶ `(read-char port)` und `(write-char c port)`
    - ▶ Liest bzw. schreibt ein einzelnes Zeichen
  - ▶ `(peek-char port)`
    - ▶ Versucht, ein Zeichen zu lesen
    - ▶ Erfolg: Gibt das Zeichen zurück, *ohne es von der Eingabe zu entfernen*
    - ▶ End-of-file: Gibt ein *eof-object* zurück. (`eof-object? obj`) ist nur für solche Objekte wahr.

- ▶ Ports können mit Dateien verknüpft erschaffen werden:
  - ▶ Input: (`open-input-file name`) öffnet die Datei mit dem angegebenen Namen und gibt einen Port zurück, der aus dieser Datei liest
    - ▶ Schließen der Datei: (`close-input-port port`)
  - ▶ Output: (`open-output-file name`)
    - ▶ Schließen der Datei: (`close-output-port port`)

## Beispiel

```
> (define (print-file inp-port)
 (if (not (eof-object? (peek-char inp-port)))
 (begin
 (write-char (read-char inp-port))
 (print-file inp-port))
 (close-input-port inp-port)))
```

```
> (let ((inprt (open-input-file "test.txt")))
 (print-file inprt))
```

Testfile

Was passiert?

Ende!

```
>
```

# Guile/R<sup>6</sup>RS-Erweiterung: Kommandozeilen und Programmende

- ▶ Beim (nicht-interaktiven) Start eines Scheme-Programmes können auf der Kommandozeile Argumente mitgegeben werden
  - ▶ Beispiel: `guile-2.0 myprog.scm Eins zwei drei`
  - ▶ Die Funktion `(command-line)` liefert diese Argumente als Liste von Strings:  

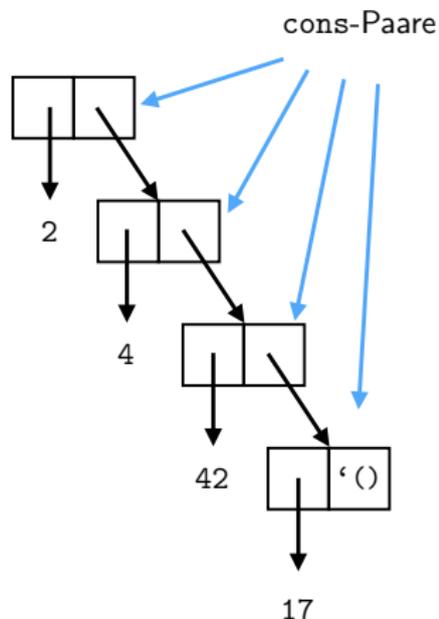
```
> (command-line)
=> (" myprog.scm" " Eins" " zwei" " drei")
```
  - ▶ Bei Racket: `(current-command-line-arguments)`
- ▶ Beenden des Programs: `(exit obj)`
  - ▶ `obj` ist optional
  - ▶ Ohne `obj`: Normales Ende (Erfolg)
  - ▶ Ist `obj #f`, dann abnormes Ende (Fehlerfall)
  - ▶ Ansonsten: Implementierungsdefiniert (aber in erster Näherung: Kleine numerische Werte werden als OS-`exit()`-Werte interpretiert)

# Übung: Komplexität von Sortierverfahren

- ▶ Beschaffen Sie sich eine Datei mit 4 Listen von zufällig generierten natürlichen Zahlen mit 1000, 2000, 4000 und 8000 Elementen (in Scheme-Syntax)
  - ▶ Unter `http://www.lehre.dhbw-stuttgart.de/~sschulz/lgli2014.html` können Sie die Datei `sortlists.txt` herunterladen
  - ▶ Alternativ können Sie äquivalente Listen selbst erzeugen (siehe `randlist.scm` auf der Webseite - Vorsicht, der Code ist Guile-spezifisch)
- ▶ Schreiben Sie ein Scheme-Programm, das diese Listen einliest und vergleichen Sie die Laufzeit von *Sortieren durch Einfügen* und *Mergesort* auf diesen Listen
  - ▶ Was beobachten Sie?
  - ▶ Können Sie diese Beobachtungen erklären?

# Die Wahrheit über Listen

- ▶ Nichtleere Listen bestehen aus cons-Paaren
- ▶ cons-Paaren haben zwei Felder:
  - ▶ car: Zeigt auf ein Element
  - ▶ cdr: Zeigt auf den Rest der Liste
- ▶ Listen sind rekursiv definiert:
  - ▶ Die leere Liste '()' ist eine Liste
  - ▶ Ein cons-Paar, dessen zweites Element eine Liste ist, ist auch eine Liste
- ▶ Die Funktion (cons a b) ...
  - ▶ Beschafft ein neues cons-Paar
  - ▶ Schreibt (einen Zeiger auf) a in dessen car
  - ▶ Schreibt (einen Zeiger auf) b in dessen cdr



'(2 4 42 17)' im Speicher

## Übung: cons-Paare

- ▶ Was passiert, wenn Sie die folgenden Ausdrücke auswerten?
  - ▶ `(cons 1 2)`
  - ▶ `(cons '() '())`
  - ▶ `(cons 1 '())`
  - ▶ `(cons 1 (cons 2 (cons 3 '())))`
  - ▶ `'(1 . 2 )` (die Leerzeichen um den Punkt sind wichtig!)
  - ▶ `'(1 . (2 . (3 . ())))`
  - ▶ `'(1 . (2 . (3 . 4)))`
- ▶ Können Sie die Ergebnisse erklären?

# Funktionen auf Listen und Paaren

## ► Typen

- `(pair? obj)` ist wahr, wenn `obj` ein cons-Paar ist
- `(list? obj)` ist wahr, wenn `obj` eine Liste ist
- `(null? obj)` ist wahr, wenn `obj` die leere Liste ist

## ► `car` und `cdr` verallgemeinert

- Scheme unterstützt Abkürzungen für den Zugriff auf Teile oder Elemente der Liste
- `(caar x)` ist äquivalent to `(car (car x))`
- `(cadr x)` ist äquivalent to `(car (cdr x))` (das zweite Element von `x`)
- `(caddr x)` ist äquivalent to `(cdr (cdr (cdr x)))` (`x` ohne die ersten 3 Elemente)
- Diese Funktionen sind bis Tiefe 4 im Standard vorgesehen

# Teillisten und Elemente

- ▶ Beliebiger Zugriff:
  - ▶ `(list-ref lst k)` gibt das *k*te Element von *lst* zurück (wenn es existiert, sonst Fehler)
  - ▶ `(list-tail lst k)` gibt die Liste ohne die ersten *k* Elemente zurück
- ▶ `(member obj lst)` sucht *obj* in List
  - ▶ Wird *obj* gefunden, so wird die längste Teil-Liste, die mit *obj* beginnt, zurückgegeben
  - ▶ Sonst: `#f`
  - ▶ Gleichheit wird über `equal?` getestet
  - ▶ Beispiele:
    - ▶ `(member 1 '(2 3 1 4 1)) ==> (1 4 1)`
    - ▶ `(member 8 '(2 3 1 4 1)) ==> #f`

# Alists - Schlüssel/Wert Paare

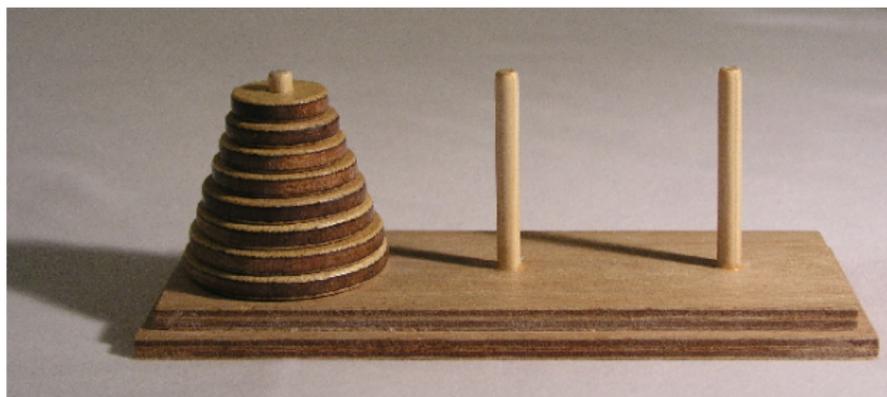
- ▶ *Association lists* sind Listen von Paaren
  - ▶ Idee: 1. Element ist der Schlüssel, 2. Element ist der Wert
  - ▶ Beachte: Jede nicht-leere Liste ist ein Paar!
- ▶ Die Funktion (`assoc obj alist`) sucht in Alists
  - ▶ Gibt es in *alist* ein Paar, dessen *car* gleich *obj* ist, so wird dieses Paar zurückgegeben
  - ▶ Sonst: `#f`
- ▶ Beispiel:

```
> (define db '(("Schulz" "Informatiker" "arm")
 ("Mayer" "Politiker" "bestechlich")
 ("Duck" "Ente" "reich")))
> (assoc "Mayer" db)
=> ("Mayer" "Politiker" "bestechlich")
> (assoc "Stephan" db)
=> #f
> (assoc "Ente" db)
=> #f
```



# Puzzle: Türme von Hanoi

- ▶ Klassisches Denk-/Geschicklichkeitsspiel
  - ▶ Ziel: Einen **Turm** von einer Position auf eine andere umziehen
  - ▶ Ein Turm besteht aus Scheiben verschiedener Größe
  - ▶ Es kann immer nur eine Scheibe bewegt werden
  - ▶ Es darf nie eine größere Scheibe auf einer kleineren liegen
  - ▶ Es gibt nur 3 mögliche Ablagestellen/Turmbauplätze



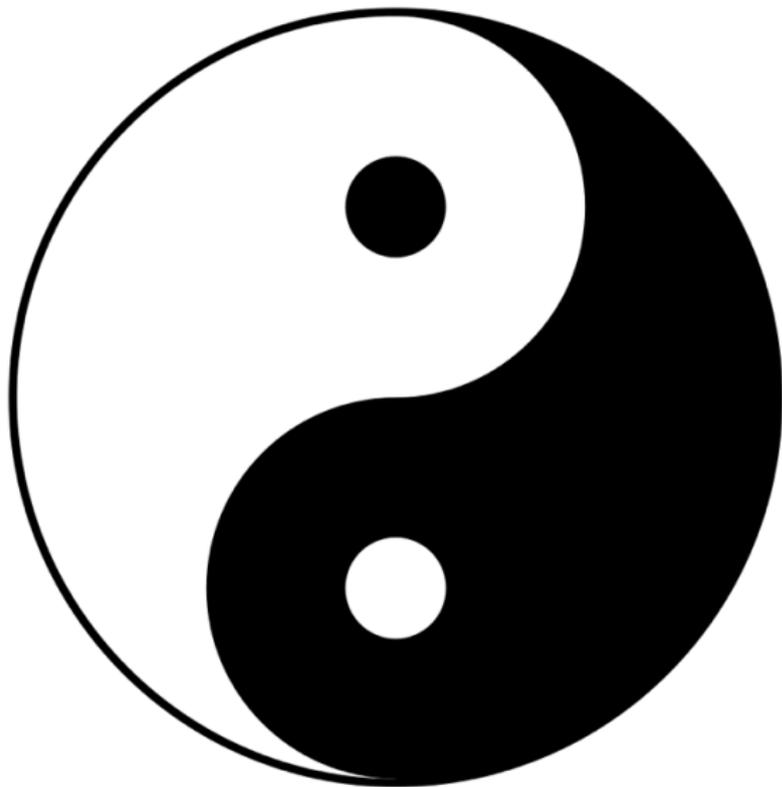
- ▶ Spielbar z.B. unter <http://www.dynamicdrive.com/dynamicindex12/towerhanoi.htm>

# Übung: Puzzle: Türme von Hanoi

- ▶ Erstellen Sie ein Scheme-Programm, das die *Türme von Hanoi* für beliebige Turmgrößen spielt
  - ▶ Was ist ein geeignetes rekursives Vorgehen?
  - ▶ Wie repräsentieren Sie den Spielstand?
  - ▶ Welche elementaren Operationen brauchen Sie
  - ▶ Wie und wann geben Sie die Züge und den Spielstand aus?
- ▶ Bonus: Versehen Sie ihr Programm mit einer „schönen“ Ausgabe des Spielstandes

- Gruppen von ca. 3 Studierenden
- Entwurfsideen schriftlich (informell) festhalten

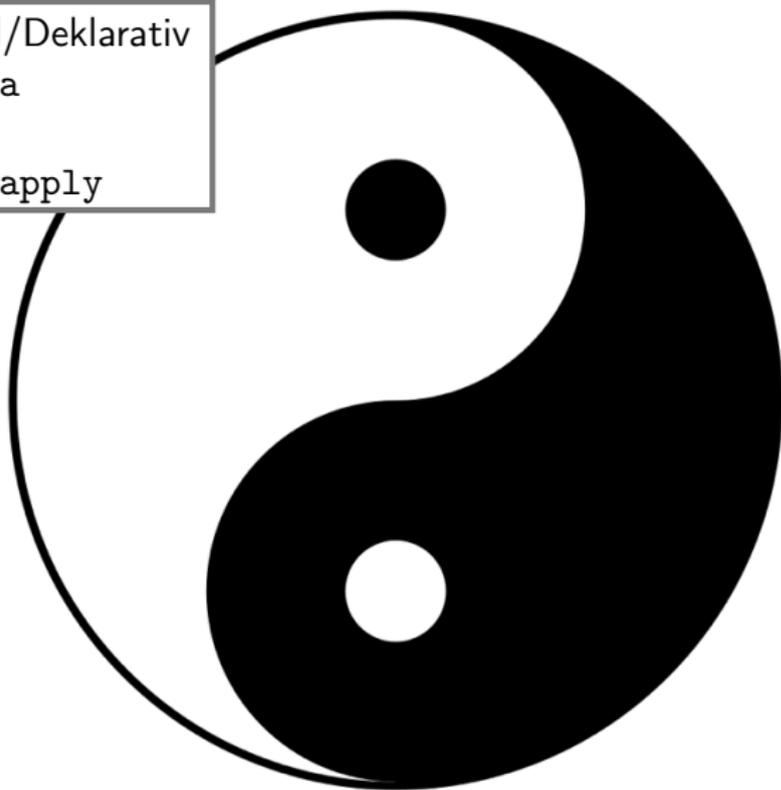
```
[Shell] guile-2.0 towers.scm 3
A: (1 2 3)
B: ()
C: ()
Moving disk 1 from A to B
A: (2 3)
B: (1)
C: ()
Moving disk 2 from A to C
A: (3)
B: (1)
C: (2)
Moving disk 1 from B to C
A: (3)
B: ()
C: (1 2)
Moving disk 3 from A to B
A: ()
B: (3)
C: (1 2)
Moving disk 1 from C to A
A: (1)
B: (3)
C: (2)
Moving disk 2 from C to B
A: (1)
B: (2 3)
C: ()
Moving disk 1 from A to B
A: ()
B: (1 2 3)
C: ()
```



# Die Helle und die Dunkle Seite der Macht

Functional/Deklarativ

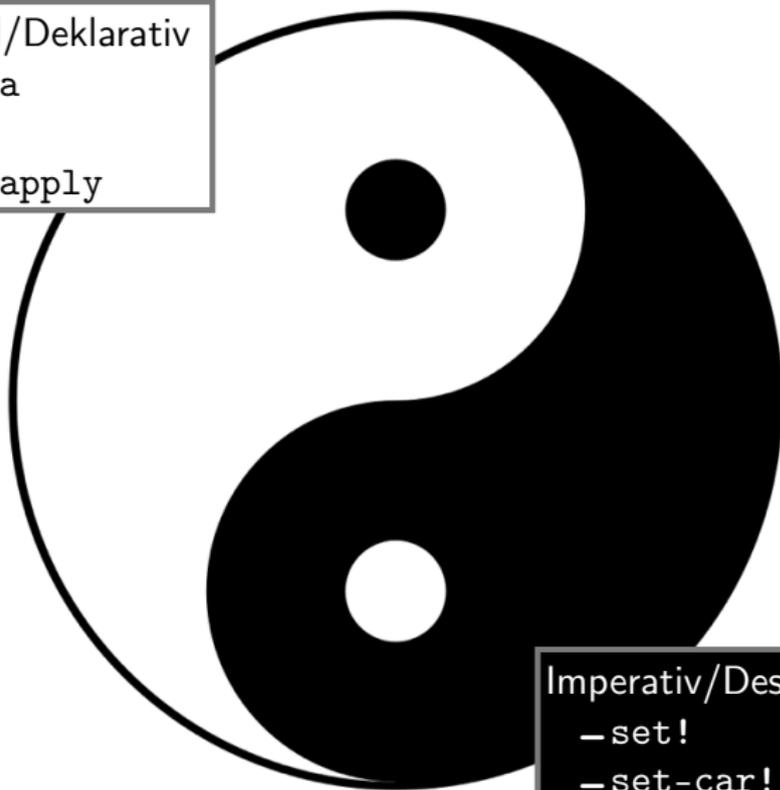
- lambda
- map
- eval/apply



# Die Helle und die Dunkle Seite der Macht

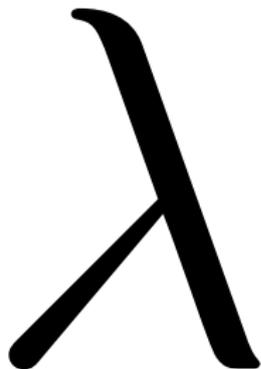
Functional/Deklarativ

- lambda
- map
- eval/apply



Imperativ/Destruktiv

- set!
- set-car!/set-cdr!



- ▶ Bekannt: Funktionsdefinition mit `define`
  - ▶ Z.B. `(define (square x)(* x x))`
- ▶ Zwei separate Operationen:
  - ▶ Kreiere **Funktion**, die `x` quadriert
  - ▶ Lege **Variable** `square` an und binde die Variable an die Funktion
  - ▶ Mit `define` können wir Operation 2 auch ohne Operation 1 ausführen: `(define newsquare square)`
- ▶ Frage: Können wir auch Funktionen erzeugen, ohne ihnen einen Namen zu geben?

# Anonyme Funktionen



- ▶ Bekannt: Funktionsdefinition mit `define`
  - ▶ Z.B. `(define (square x)(* x x))`
- ▶ Zwei separate Operationen:
  - ▶ Kreiere **Funktion**, die `x` quadriert
  - ▶ Lege **Variable** `square` an und binde die Variable an die Funktion
  - ▶ Mit `define` können wir Operation 2 auch ohne Operation 1 ausführen: `(define newsquare square)`
- ▶ Frage: Können wir auch Funktionen erzeugen, ohne ihnen einen Namen zu geben?

Antwort: Ja, mit `lambda`!

# lambda

- ▶ lambda-Ausdrücke haben als Wert eine neue Prozedur/Funktion
  - ▶ Argumente: Liste von formalen Parametern und Sequenz von Ausdrücken
  - ▶ Die neue Funktion akzeptiert die angegebenen Parameter
  - ▶ Sie wertet die Sequenz in der um die formalen Parameter mit ihren aktuellen Werten erweiterten Umgebung aus
  - ▶ Wert der Funktion ist der Wert der Sequenz
- ▶ Beispiel:

```
> ((lambda (x y)(+ (* 2 x) y)) 3 5)
```

# lambda

- ▶ lambda-Ausdrücke haben als Wert eine neue Prozedur/Funktion
  - ▶ Argumente: Liste von formalen Parametern und Sequenz von Ausdrücken
  - ▶ Die neue Funktion akzeptiert die angegebenen Parameter
  - ▶ Sie wertet die Sequenz in der um die formalen Parameter mit ihren aktuellen Werten erweiterten Umgebung aus
  - ▶ Wert der Funktion ist der Wert der Sequenz
- ▶ Beispiel:

```
> ((lambda (x y)(+ (* 2 x) y)) 3 5)
```

```
==> 11
```

```
> ((lambda () (/ 6.283 2)))
```

# lambda

- ▶ lambda-Ausdrücke haben als Wert eine neue Prozedur/Funktion
  - ▶ Argumente: Liste von formalen Parametern und Sequenz von Ausdrücken
  - ▶ Die neue Funktion akzeptiert die angegebenen Parameter
  - ▶ Sie wertet die Sequenz in der um die formalen Parameter mit ihren aktuellen Werten erweiterten Umgebung aus
  - ▶ Wert der Funktion ist der Wert der Sequenz
- ▶ Beispiel:

```
> ((lambda (x y)(+ (* 2 x) y)) 3 5)
```

```
==> 11
```

```
> ((lambda () (/ 6.283 2)))
```

```
==> 3.1415
```

# Warum lambda?

- ▶ lambda ist fundamental!
  - ▶ `(define (fun args) body)` ist nur eine Abkürzung für `(define fun (lambda (args) body))`
- ▶ lambdas können im lokalen Kontext erzeugt werden
  - ▶ Keine Verschmutzung des globalen Namensraums
  - ▶ Zugriff auf lokale Variable (!)
- ▶ lambda kann let und (mit geistigem Strecken) define ersetzen:
  - ▶ `(let ((a init1) (b init2)) (mach-was a b))` ist äquivalent zu
  - ▶ `((lambda (a b) (mach-was a b)) init1 init2)`

# Von Mengen zu funktionierenden Funktionen

- ▶ Rückblick Mengenlehre:
  - ▶ Funktionen sind rechtseindeutige Relationen
  - ▶ Relationen sind Mengen von Paaren
- ▶ Rückblick Hausaufgabe:
  - ▶ Mengen sind duplikatfreie Listen
  - ▶ Paare sind zweielementige Listen
- ▶ Mit `lambda` können wir aus der Mengenversion eine anwendbare Funktion machen:

```
> (define (set->fun relation)
 (lambda (x) (let ((res (assoc x relation)))
 (and res (cadr res)))))
```

```
> (define rel '((1 2) (2 4) (3 6)))
```

```
> (define fun (set->fun rel))
```

```
> (fun 3)
```

```
==> 6
```

```
> (fun 10)
```

```
==> #f
```

## Für faule: map

- ▶ Version 1: `map` wendet eine Funktion auf alle Elemente einer Liste an
  - ▶ Argumente: Funktion mit einem Argument, Liste von Elementen
  - ▶ Ergebnis: Liste von Ergebnissen
- ▶ Beispiel:  
> **(map (lambda (x)(\* 3 x)) '(1 2 3 4))**

## Für faule: map

- ▶ Version 1: map wendet eine Funktion auf alle Elemente einer Liste an

- ▶ Argumente: Funktion mit einem Argument, Liste von Elementen
- ▶ Ergebnis: Liste von Ergebnissen

- ▶ Beispiel:

```
> (map (lambda (x)(* 3 x)) '(1 2 3 4))
```

```
⇒ (3 6 9 12)
```

```
> (map (lambda (x)(if (> x 10) 1 0)) '(5 12 14 3 31))
```

## Für faule: map

- ▶ Version 1: map wendet eine Funktion auf alle Elemente einer Liste an
  - ▶ Argumente: Funktion mit einem Argument, Liste von Elementen
  - ▶ Ergebnis: Liste von Ergebnissen
- ▶ Beispiel:

```
> (map (lambda (x)(* 3 x)) '(1 2 3 4))
```

```
⇒ (3 6 9 12)
```

```
> (map (lambda (x)(if (> x 10) 1 0)) '(5 12 14 3 31))
```

```
⇒ (0 1 1 0 1)
```

## map für Funktionen mit mehreren Argumenten

- ▶ Version 2: `map` wendet eine Funktion auf alle Tupel von korrespondierenden Elementen mehrerer Listen an
  - ▶ Argument: Funktion  $f$  mit  $n$  Argumenten
  - ▶  $n$  Listen von Elementen  $(l_1, \dots, l_n)$
  - ▶ Ergebnis: Liste von Ergebnissen
    - ▶ Das erste Element des Ergebnisses ist  $f((\text{car } l_1), \dots, (\text{car } l_2))$
- ▶ Beispiel:  

```
> (map (lambda (x y) (if (> x y) x y))
 '(2 4 6 3) '(1 5 8 4))
```

## map für Funktionen mit mehreren Argumenten

- ▶ Version 2: map wendet eine Funktion auf alle Tupel von korrespondierenden Elementen mehrerer Listen an
  - ▶ Argument: Funktion  $f$  mit  $n$  Argumenten
  - ▶  $n$  Listen von Elementen  $(l_1, \dots, l_n)$
  - ▶ Ergebnis: Liste von Ergebnissen
    - ▶ Das erste Element des Ergebnises ist  $f((\text{car} l_1), \dots, (\text{car} l_2))$

- ▶ Beispiel:

```
> (map (lambda (x y) (if (> x y) x y))
 '(2 4 6 3) '(1 5 8 4))
```

```
⇒ (2 5 8 4)
```

## Program oder Daten? apply

- ▶ `apply` wendet eine Funktion auf eine Liste von Argumenten an
  - ▶ Argument 1: Funktion, die  $n$  Argumente akzeptiert
  - ▶ Argument 2: Liste von  $n$  Argumenten
  - ▶ Ergebnis: Wert der Funktion, angewendet auf die Argumente
- ▶ Beispiel:  
> (**apply** + '(11 7 14))

## Program oder Daten? apply

- ▶ `apply` wendet eine Funktion auf eine Liste von Argumenten an
  - ▶ Argument 1: Funktion, die  $n$  Argumente akzeptiert
  - ▶ Argument 2: Liste von  $n$  Argumenten
  - ▶ Ergebnis: Wert der Funktion, angewendet auf die Argumente
- ▶ Beispiel:

```
> (apply + '(11 7 14))
```

```
==> 32
```

```
> (apply map (list (lambda (x)(+ x 3)) '(1 2 3)))
```

## Program oder Daten? apply

- ▶ `apply` wendet eine Funktion auf eine Liste von Argumenten an
  - ▶ Argument 1: Funktion, die  $n$  Argumente akzeptiert
  - ▶ Argument 2: Liste von  $n$  Argumenten
  - ▶ Ergebnis: Wert der Funktion, angewendet auf die Argumente
- ▶ Beispiel:

```
> (apply + '(11 7 14))
```

```
⇒ 32
```

```
> (apply map (list (lambda (x)(+ x 3)) '(1 2 3)))
```

```
⇒ (4 5 6)
```

## Program oder Daten? apply

- ▶ `apply` wendet eine Funktion auf eine Liste von Argumenten an
  - ▶ Argument 1: Funktion, die  $n$  Argumente akzeptiert
  - ▶ Argument 2: Liste von  $n$  Argumenten
  - ▶ Ergebnis: Wert der Funktion, angewendet auf die Argumente
- ▶ Beispiel:

```
> (apply + '(11 7 14))
```

```
⇒ 32
```

```
> (apply map (list (lambda (x)(+ x 3)) '(1 2 3)))
```

```
⇒ (4 5 6)
```

Achtung: `apply` ruft die anzuwendende Funktion **einmal** mit **allen** Listenelementen auf, `map` für jedes Element einzeln!

# Scheme in einem Befeh: eval

- ▶ `eval` nimmt einen Scheme-Ausdruck und eine Umgebung, und wertet den Ausdruck in der Umgebung aus
  - ▶ Vom Standard garantierte interessante Umgebungen:
    - ▶ `(scheme-report-environment 5)` - wie in R5RS definiert
    - ▶ `(interaction-environment)` - wie im interaktiven Modus
    - ▶ `(null-environment 5)` - ohne definierte Variablen!
- ▶ Beispiele:

```
> (eval '(+ 3 4) (scheme-report-environment 5))
=> 7
```

```
> (eval '(+ 3 4) (null-environment 5))
ERROR: In procedure memoize-variable-access!:
ERROR: Unbound variable: +
```

# Scheme in einem Befeh: eval

- ▶ `eval` nimmt einen Scheme-Ausdruck und eine Umgebung, und wertet den Ausdruck in der Umgebung aus
  - ▶ Vom Standard garantierte interessante Umgebungen:
    - ▶ `(scheme-report-environment 5)` - wie in R5RS definiert
    - ▶ `(interaction-environment)` - wie im interaktiven Modus
    - ▶ `(null-environment 5)` - ohne definierte Variablen!
- ▶ Beispiele:

```
> (eval '(+ 3 4) (scheme-report-environment 5))
=> 7
```

```
> (eval '(+ 3 4) (null-environment 5))
ERROR: In procedure memoize-variable-access!:
ERROR: Unbound variable: +
```



## Übung: map und apply

- ▶ Schreiben Sie eine Funktion, die das Skalarprodukt von zwei  $n$ -elementigen Vektoren (repräsentiert als Listen) berechnet
  - ▶ Das Skalarprodukt von  $(a_1, a_2, \dots, a_n)$  und  $(b_1, b_2, \dots, b_n)$  ist definiert als  $\sum_{1 \leq i \leq n} a_i b_i$
  - ▶ Beispiel: (skalarprodukt '(1 2 3) '(2 4 6)) ==> 28
- ▶ Schreiben Sie eine Funktion, die geschachtelte Listen verflacht.
  - ▶ Beispiel: (flatten '(1 2 (3 4 (5 6) 7 8) 9)) ==> (1 2 3 4 5 6 7 8 9)



## Destruktive Zuweisung: set!

- ▶ `set!` weist einer existierenden Variable einen neuen Wert zu

```
> (define a 10)
```

```
> a
```

```
==> 10
```

```
> (set! a "Hallo")
```

```
> a
```

```
==> "Hallo"
```

```
> (set! a '(+ 2 11))
```

```
> a
```

```
==> (+ 2 11)
```

```
> (eval a (interaction-environment 5))
```

```
==> 13
```

## set-car!/set-cdr!

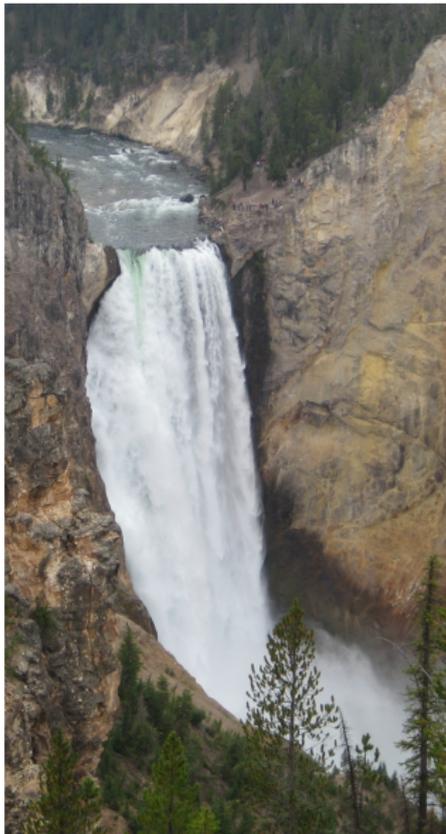
- ▶ set-car! verändert den ersten Wert eines existierenden cons-Paares
- ▶ set-cdr! verändert den zweiten Wert eines existierenden cons-Paares

```
> (define l '(1 2 3))
> (set-car! l "Hallo")
> l
=> ("Hallo" 2 3)
> (set-cdr! (cddr l) '(5 6 7))
> l
=> ("Hallo" 2 3 5 6 7)
> (set-cdr! (cddr l) l)
????
```

# Scheme Namenskonventionen

- ▶ Verändernde (destruktive) Funktionen enden in !
  - ▶ `set!`, `set-car!`, `vector-set!`, ...
  - ▶ Diese Funktionen haben typischerweise keinen Rückgabewert!
- ▶ Prädikate (liefern #t oder #f) enden in ?
  - ▶ `null?`, `pair?`, `equal?`...
- ▶ Konvertierungsfunktionen enthalten ->
  - ▶ `string->number`, `list->vector`

# Das Typsystem von Scheme



- ▶ Scheme ist eine strikt, aber **dynamisch** getypte Sprache:
  - ▶ Jedes Datenobjekt hat einen eindeutigen Basistyp
  - ▶ Dieser Typ geht direkt aus dem Objekt hervor, nicht aus seiner Speicherstelle („Variable“)
  - ▶ Variablen können an Objekte verschiedenen Typs gebunden sein
- ▶ Objekte können in Listen (und Vektoren) zu komplexeren Strukturen kombiniert werden

# Die Typen von Scheme

- ▶ Typprädikate (jedes Objekt hat genau einen dieser Typen):
  - boolean?        #t und #f
  - pair?            cons-Zellen (damit auch nicht-leere Listen)
  - symbol?        Normale Bezeichner, z.B. hallo, \*, symbol?. Achtung: Symbole müssen gequoted werden, wenn man das Symbol, nicht seinen Wert referenzieren will!
  
  - number?        Zahlen: 1, 3.1415, ...
  - char?           Einzelne Zeichen: #\a, #\b, #\7, ...
  - string?        "Hallo", "1", "1/2 oder Otto"
  - vector?        Aus Zeitmangel nicht erwähnt (nehmen Sie Listen)
  - port?          Siehe Vorlesung zu Input/Output
  - procedure?    Ausführbare Funktionen (per define oder lambda)
  - null?          Sonderfall: Die leere Liste '()

# Symbole als Werte

*Symbols are objects whose usefulness rests on the fact that two symbols are identical (in the sense of eqv?) if and only if their names are spelled the same way.*

*R5RS (6.3.3)*

- ▶ Symbole sind eine eigener Scheme-Datentyp
  - ▶ Sie können als Variablennamen dienen, sind aber auch selbst Werte
  - ▶ Ein Symbol wird durch **quoten** direkt verwendet (sonst in der Regel sein **Wert**)

- ▶ Beispiele:

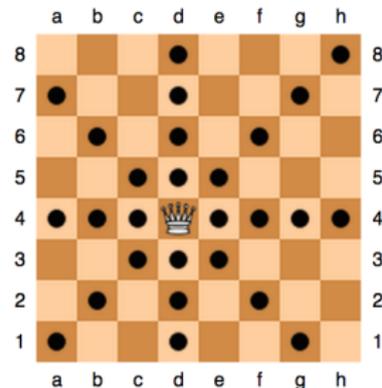
```
> (define a 'hallo)
> a
=> hallo
> (define l '(hallo dings bums))
> l
=> (hallo dings bums)
> (equal? (car l) a)
=> #t
```

# Mut zur Lücke

- ▶ Viel über Zahlen (R5RS 6.2)
  - ▶ Der Zahlenturm: *number*, *complex*, *real*, *rational*, *integer*
  - ▶ Viele Operationen
- ▶ Zeichen und Strings (R5RS 6.3.4 und 6.3.5)
  - ▶ "Hallo", *string-set!*, *string-ref*, *substring*, ...
- ▶ Vektoren (R5RS 6.3.6)
  - ▶ Vektoren sind (grob) wie Listen fester Länge (kein *append* oder *cons*) mit schnellerem Zugriff auf Elemente
- ▶ Variadische Funktionen (4.1.3)
  - ▶ Zusätzliche Argumente werden als Liste übergeben
- ▶ Macros (R5RS 5.3)
  - ▶ Erlauben die Definition von *Special Forms*
  - ▶ Wichtig, wenn man einen Scheme-Interpreter schreibt
  - ▶ Ansonsten sehr cool, aber nicht oft gebraucht

# Übung: Höfliche Damen

- ▶ Eine Dame (im Schach) bedroht alle Felder in ihrer Reihe, in ihrer Spalte, und auf ihren Diagonalen
- ▶ Erstellen Sie ein Programm, das 8 Damen auf einem Schachbrett so platziert, dass Sie sich nicht bedrohen.
- ▶ Bonus: Lösen Sie das Problem für beliebig große quadratische Schachbretter, also: Platzieren sie  $n$  Damen so auf einem  $n \times n$  großen Schachbrett so, dass sie sich nicht bedrohen



Zurück B

Zurück C

## Aussagenlogik

# Übung: Verbrechensaufklärung

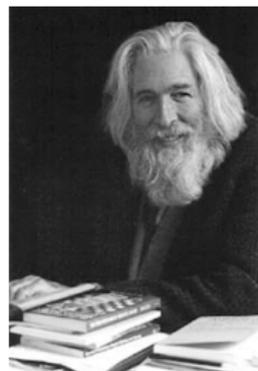
Ein Fall aus den Akten von Inspektor Craig: „Was fängst du mit diesen Fakten an?“ fragt Inspektor Craig den Sergeant McPherson.

1. Wenn A schuldig und B unschuldig ist, so ist C schuldig.
2. C arbeitet niemals allein.
3. A arbeitet niemals mit C.
4. Niemand außer A, B oder C war beteiligt, und mindestens einer von ihnen ist schuldig.

Der Sergeant kratzte sich den Kopf und sagte: „Nicht viel, tut mir leid, Sir. Können Sie nicht aus diesen Fakten schließen, wer unschuldig und wer schuldig ist?“ „Nein“, entgegnete Craig, „aber das Material reicht aus, um wenigstens einen von ihnen zu anzuklagen“.

## Wen und warum?

nach R. Smullyan: „Wie heißt dieses Buch?“



# Entscheidungshilfe

1. Wenn A schuldig und B unschuldig ist, so ist C schuldig.
2. C arbeitet niemals allein.
3. A arbeitet niemals mit C.
4. Niemand außer A, B oder C war beteiligt, und mindestens einer von ihnen ist schuldig.

| <b>A</b> | <b>B</b> | <b>C</b> | <b>1.</b> | <b>2.</b> | <b>3.</b> | <b>4.</b> | <b>1.-4.</b> |
|----------|----------|----------|-----------|-----------|-----------|-----------|--------------|
|          |          |          |           |           |           |           |              |
|          |          |          |           |           |           |           |              |
|          |          |          |           |           |           |           |              |
|          |          |          |           |           |           |           |              |
|          |          |          |           |           |           |           |              |
|          |          |          |           |           |           |           |              |
|          |          |          |           |           |           |           |              |
|          |          |          |           |           |           |           |              |

Syntax

- Was ist ein korrekter Satz?

Syntax

- Was ist ein korrekter Satz?

Semantik

- Wann ist ein Satz wahr oder falsch?

- Syntax - Was ist ein korrekter Satz?
- Semantik - Wann ist ein Satz wahr oder falsch?
- Deduktionsmechanismus - Wie kann ich neues Wissen herleiten? Wie kann ich die Gültigkeit einer Formel oder einer Ableitung gewährleisten?

## Atomare Aussagen

C ist schuldig

$C$

Die Straße ist nass

$strasseNass$

## Verknüpft mit logischen Operatoren

und

$\wedge$

oder

$\vee$

impliziert

$\rightarrow$

nicht

$\neg$

- ▶ Inspektor Craigs Ergebnisse
  1. Wenn A schuldig und B unschuldig ist, so ist C schuldig.
  2. C arbeitet niemals allein.
  3. A arbeitet niemals mit C.
  4. Niemand außer A, B oder C war beteiligt, und mindestens einer von ihnen ist schuldig.
- ▶ Atomare Aussagen

# Ermittlungsergebnisse formal

- ▶ Inspektor Craigs Ergebnisse

1. Wenn A schuldig und B unschuldig ist, so ist C schuldig.
2. C arbeitet niemals allein.
3. A arbeitet niemals mit C.
4. Niemand außer A, B oder C war beteiligt, und mindestens einer von ihnen ist schuldig.

- ▶ Atomare Aussagen

- ▶ A ist schuldig  
*A*

B ist schuldig  
*B*

C ist schuldig  
*C*

# Ermittlungsergebnisse formal

► Inspektor Craigs Ergebnisse

1. Wenn A schuldig und B unschuldig ist, so ist C schuldig.
2. C arbeitet niemals allein.
3. A arbeitet niemals mit C.
4. Niemand außer A, B oder C war beteiligt, und mindestens einer von ihnen ist schuldig.

► Atomare Aussagen

- A ist schuldig                      B ist schuldig                      C ist schuldig  
*A*                                              *B*                                              *C*

► Ermittlungsergebnisse formal

1.  $(A \wedge \neg B) \rightarrow C$

# Ermittlungsergebnisse formal

## ► Inspektor Craigs Ergebnisse

1. Wenn A schuldig und B unschuldig ist, so ist C schuldig.
2. C arbeitet niemals allein.
3. A arbeitet niemals mit C.
4. Niemand außer A, B oder C war beteiligt, und mindestens einer von ihnen ist schuldig.

## ► Atomare Aussagen

- A ist schuldig                      B ist schuldig                      C ist schuldig  
*A*                                              *B*                                              *C*

## ► Ermittlungsergebnisse formal

1.  $(A \wedge \neg B) \rightarrow C$
2.  $C \rightarrow (A \vee B)$

# Ermittlungsergebnisse formal

## ► Inspektor Craigs Ergebnisse

1. Wenn A schuldig und B unschuldig ist, so ist C schuldig.
2. C arbeitet niemals allein.
3. A arbeitet niemals mit C.
4. Niemand außer A, B oder C war beteiligt, und mindestens einer von ihnen ist schuldig.

## ► Atomare Aussagen

- A ist schuldig                      B ist schuldig                      C ist schuldig  
*A*                                              *B*                                              *C*

## ► Ermittlungsergebnisse formal

1.  $(A \wedge \neg B) \rightarrow C$
2.  $C \rightarrow (A \vee B)$  (mit 4. Teil 1)

# Ermittlungsergebnisse formal

## ► Inspektor Craigs Ergebnisse

1. Wenn A schuldig und B unschuldig ist, so ist C schuldig.
2. C arbeitet niemals allein.
3. A arbeitet niemals mit C.
4. Niemand außer A, B oder C war beteiligt, und mindestens einer von ihnen ist schuldig.

## ► Atomare Aussagen

- A ist schuldig                      B ist schuldig                      C ist schuldig
- A*                                              *B*                                              *C*

## ► Ermittlungsergebnisse formal

1.  $(A \wedge \neg B) \rightarrow C$
2.  $C \rightarrow (A \vee B)$  (mit 4. Teil 1)
3.  $A \rightarrow \neg C$

# Ermittlungsergebnisse formal

## ► Inspektor Craigs Ergebnisse

1. Wenn A schuldig und B unschuldig ist, so ist C schuldig.
2. C arbeitet niemals allein.
3. A arbeitet niemals mit C.
4. Niemand außer A, B oder C war beteiligt, und mindestens einer von ihnen ist schuldig.

## ► Atomare Aussagen

► A ist schuldig  
*A*

B ist schuldig  
*B*

C ist schuldig  
*C*

## ► Ermittlungsergebnisse formal

1.  $(A \wedge \neg B) \rightarrow C$
2.  $C \rightarrow (A \vee B)$  (mit 4. Teil 1)
3.  $A \rightarrow \neg C$
4.  $A \vee B \vee C$

**Definition:** Eine **aussagenlogische Signatur**  $\Sigma$  ist eine (nichtleere) abzählbare Menge von Symbolen, etwa

$$\Sigma = \{A_0, \dots, A_n\} \text{ oder } \Sigma = \{A_0, A_1, \dots\}$$

**Definition:** Eine **aussagenlogische Signatur**  $\Sigma$  ist eine (nichtleere) abzählbare Menge von Symbolen, etwa

$$\Sigma = \{A_0, \dots, A_n\} \text{ oder } \Sigma = \{A_0, A_1, \dots\}$$

## Bezeichnungen für Symbole in $\Sigma$

- ▶ atomare Aussagen
- ▶ Atome
- ▶ Aussagevariablen, aussagenlogische Variablen
- ▶ Propositionen

$\top$  Symbol für den Wahrheitswert „wahr“

$\perp$  Symbol für den Wahrheitswert „falsch“

# Syntax der Aussagenlogik: Logische Zeichen

- $\top$  Symbol für den Wahrheitswert „wahr“
- $\perp$  Symbol für den Wahrheitswert „falsch“
- $\neg$  Negationssymbol („nicht“)

# Syntax der Aussagenlogik: Logische Zeichen

$\top$  Symbol für den Wahrheitswert „wahr“

$\perp$  Symbol für den Wahrheitswert „falsch“

$\neg$  Negationssymbol („nicht“)

$\wedge$  Konjunktionssymbol („und“)

$\vee$  Disjunktionssymbol („oder“)

$\rightarrow$  Implikationssymbol („wenn ... dann“)

$\leftrightarrow$  Symbol für Äquivalenz („genau dann, wenn“)

# Syntax der Aussagenlogik: Logische Zeichen

$\top$  Symbol für den Wahrheitswert „wahr“

$\perp$  Symbol für den Wahrheitswert „falsch“

$\neg$  Negationssymbol („nicht“)

$\wedge$  Konjunktionssymbol („und“)

$\vee$  Disjunktionssymbol („oder“)

$\rightarrow$  Implikationssymbol („wenn ... dann“)

$\leftrightarrow$  Symbol für Äquivalenz („genau dann, wenn“)

$( )$  die beiden Klammern

**Definition:** Menge  $For0_\Sigma$  der Formeln über  $\Sigma$

Sei  $\Sigma$  eine Menge von Atomen.  $For0_\Sigma$  ist die kleinste Menge mit:

- ▶  $\top \in For0_\Sigma$
- ▶  $\perp \in For0_\Sigma$
- ▶  $\Sigma \subseteq For0_\Sigma$  (jedes Atom ist eine Formel)
- ▶ Wenn  $A, B \in For0_\Sigma$ , dann sind auch  
 $(\neg A)$ ,  $(A \wedge B)$ ,  $(A \vee B)$ ,  $(A \rightarrow B)$ ,  $(A \leftrightarrow B)$   
Elemente von  $For0_\Sigma$

- ▶ Beachte:  $A$  und  $B$  sind oben keine *aussagenlogischen Variablen*, sondern sie stehen für beliebige Formeln („*Meta-Variable*“)

# Übung: Syntax der Aussagenlogik

Sei  $\Sigma = \{A, B, C\}$ . Identifizieren Sie die korrekten aussagenlogischen Formeln.

a)  $A \rightarrow \perp$

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

c)  $(A \neg B)$

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

e)  $(\vee B \vee (C \wedge D))$

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

g)  $(A \rightarrow A)$

h)  $(A \wedge (\neg A))$

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

j)  $((\neg A) + B)$

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

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

# Präzedenz und Assoziativität

- ▶ Vereinbarung zum Minimieren von Klammern:
  - ▶ Das äußerste Klammernpaar kann weggelassen werden
  - ▶ Die Operatoren binden verschieden stark:

| Operator          | Präzedenz              |
|-------------------|------------------------|
| $\neg$            | 1 (stärkste Bindung)   |
| $\wedge$          | 2                      |
| $\vee$            | 3                      |
| $\rightarrow$     | 4                      |
| $\leftrightarrow$ | 5 (schwächste Bindung) |

$$A \wedge \neg B \rightarrow C \iff ((A \wedge (\neg B)) \rightarrow C)$$

- ▶ Zweistellige Operatoren gleicher Präzedenz sind linksassoziativ:

$$A \wedge B \wedge C \iff ((A \wedge B) \wedge C)$$

$$A \rightarrow B \rightarrow A \iff ((A \rightarrow B) \rightarrow A)$$

- ▶  $\neg$  ist rechtsassoziativ:  $\neg\neg\neg A \iff (\neg(\neg(\neg A)))$

- ▶ Klammern, die so überflüssig werden, dürfen weggelassen werden

# Übung: Präzedenzen und Klammern

Entfernen Sie so viele Klammern, wie möglich, ohne die Struktur der Formeln zu verändern

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

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

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

| Operator          | Präzedenz      |
|-------------------|----------------|
| $\neg$            | 1 (stärkste)   |
| $\wedge$          | 2              |
| $\vee$            | 3              |
| $\rightarrow$     | 4              |
| $\leftrightarrow$ | 5 (schwächste) |

AT LAST, SOME CLARITY! EVERY  
SENTENCE IS EITHER PURE,  
SWEET TRUTH OR A VILE,  
CONTEMPTIBLE LIE! ONE  
OR THE OTHER! NOTHING  
IN BETWEEN!



Sei  $\Sigma$  eine aussagenlogische Signatur

**Definition:** Aussagenlogische Interpretation

Eine **Interpretation** ist eine beliebige Abbildung  $I : \Sigma \rightarrow \{1, 0\}$

Sei  $\Sigma$  eine aussagenlogische Signatur

**Definition:** Aussagenlogische Interpretation

Eine **Interpretation** ist eine beliebige Abbildung  $I : \Sigma \rightarrow \{1, 0\}$

**Beispiel:**  $I = \{A \mapsto 1, B \mapsto 1, C \mapsto 0\}$

**Tabellarisch:**

| $A$ | $B$ | $C$ |
|-----|-----|-----|
| 1   | 1   | 0   |

Bei drei Atomen gibt es 8 mögliche Interpretationen

**Definition:** Auswertung von Formeln unter einer Interpretation

Eine Interpretation  $I$  wird fortgesetzt zu einer Auswertungsfunktion

$$\text{val}_I : \text{For}0_\Sigma \longrightarrow \{1, 0\}$$

durch:

$$\text{val}_I(\top) = 1$$

$$\text{val}_I(\perp) = 0$$

$$\text{val}_I(P) = I(P) \quad \text{für } P \in \Sigma$$

und:

$$\text{val}_I(\neg A) = \begin{cases} 0 & \text{falls } \text{val}_I(A) = 1 \\ 1 & \text{falls } \text{val}_I(A) = 0 \end{cases}$$

und:

$$\text{val}_I(A \wedge B) = \begin{cases} 1 & \text{falls } \text{val}_I(A) = 1 \text{ und } \text{val}_I(B) = 1 \\ 0 & \text{sonst} \end{cases}$$

$$\text{val}_I(A \vee B) = \begin{cases} 1 & \text{falls } \text{val}_I(A) = 1 \text{ oder } \text{val}_I(B) = 1 \\ 0 & \text{sonst} \end{cases}$$

und:

$$\text{val}_I(A \rightarrow B) = \begin{cases} 1 & \text{falls } \text{val}_I(A) = 0 \text{ oder } \text{val}_I(B) = 1 \\ 0 & \text{sonst} \end{cases}$$

und:

$$\text{val}_I(A \leftrightarrow B) = \begin{cases} 1 & \text{falls } \text{val}_I(A) = \text{val}_I(B) \\ 0 & \text{sonst} \end{cases}$$

# Wahrheitstafel für die logischen Operatoren

| $A$ | $B$ | $\neg A$ | $A \wedge B$ | $A \vee B$ | $A \rightarrow B$ | $A \leftrightarrow B$ |
|-----|-----|----------|--------------|------------|-------------------|-----------------------|
| 0   | 0   | 1        | 0            | 0          | 1                 | 1                     |
| 0   | 1   | 1        | 0            | 1          | 1                 | 0                     |
| 1   | 0   | 0        | 0            | 1          | 0                 | 0                     |
| 1   | 1   | 0        | 1            | 1          | 1                 | 1                     |

- ▶ Sprachregelung: Falls  $\text{val}_I(A) = 1/0$ :
  - ▶  $I$  macht  $A$  wahr/falsch
  - ▶  $A$  ist wahr/falsch unter  $I$
  - ▶  $A$  ist wahr/falsch in  $I$
- ▶ Statt  $\text{val}_I(A)$  schreiben wir auch einfach  $I(A)$

## Beispiel: Interpretationen

**Beispiel:** Die Interpretation  $I = \{A \mapsto 1, B \mapsto 1, C \mapsto 0\}$  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

## Ex falso, Quodlibet

- ▶ Aus einer falschen (widersprüchlichen) Annahme kann man alles folgern

## Ex falso, Quodlibet

- ▶ Aus einer falschen (widersprüchlichen) Annahme kann man alles folgern
  - ▶  $\perp \rightarrow A$  ist immer gültig
  - ▶ „Wenn der Staatshaushalt ausgeglichen ist, werden wir die Steuern senken“

## Modell einer Formel(menge)

Definition: Eine Interpretation  $I$  ist **Modell einer Formel**  $A \in \text{For}_0^\Sigma$ , falls

$$\text{val}_I(A) = 1$$

# Modell einer Formel(menge)

Definition: Eine Interpretation  $I$  ist **Modell einer Formel**  $A \in For0_\Sigma$ , falls

$$\text{val}_I(A) = 1$$

Definition: Eine Interpretation  $I$  ist **Modell einer Formelmenge**  $M \subseteq For0_\Sigma$ , falls

$$\text{val}_I(A) = 1 \quad \text{für alle } A \in M$$

## Modell einer Formel(menge)

Definition: Eine Interpretation  $I$  ist **Modell einer Formel**  $A \in For0_\Sigma$ , falls

$$\text{val}_I(A) = 1$$

Definition: Eine Interpretation  $I$  ist **Modell einer Formelmenge**  $M \subseteq For0_\Sigma$ , falls

$$\text{val}_I(A) = 1 \quad \text{für alle } A \in M$$

- Wir betrachten also eine Formelmenge hier implizit als Konjunktion („verundung“) ihrer einzelnen Formeln

## Erinnerung: Inspektor Craig

1.  $(A \wedge \neg B) \rightarrow C$
2.  $C \rightarrow (A \vee B)$
3.  $A \rightarrow \neg C$
4.  $A \vee B \vee C$

| <b>A</b> | <b>B</b> | <b>C</b> | <b>1.</b> | <b>2.</b> | <b>3.</b> | <b>4.</b> | <b>1.-4.</b> |         |
|----------|----------|----------|-----------|-----------|-----------|-----------|--------------|---------|
| 0        | 0        | 0        | 1         | 1         | 1         | 0         | 0            |         |
| 0        | 0        | 1        | 1         | 0         | 1         | 1         | 0            |         |
| 0        | 1        | 0        | 1         | 1         | 1         | 1         | 1            | Modell! |
| 0        | 1        | 1        | 1         | 1         | 1         | 1         | 1            | Modell! |
| 1        | 0        | 0        | 0         | 1         | 1         | 1         | 0            |         |
| 1        | 0        | 1        | 1         | 1         | 0         | 1         | 0            |         |
| 1        | 1        | 0        | 1         | 1         | 1         | 1         | 1            | Modell! |
| 1        | 1        | 1        | 1         | 1         | 0         | 1         | 0            |         |

## Übung: Interpretationen und Modelle

Finden Sie zwei Interpretationen für jede der folgenden Formeln. Dabei sollte eine ein Modell sein, die andere keine Modell.

a)  $A \wedge B \rightarrow C$

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

c)  $A \rightarrow B \leftrightarrow \neg B \rightarrow \neg A$

Zurück C

# Tautologie

Definition: Eine Formel  $F \in \text{For}_{0\Sigma}$  heißt **Tautologie** oder **allgemeingültig**, falls  $\text{val}_I(F) = 1$  für jede Interpretation  $I$ .

Schreibweise:  $\models F$

- ▶ Sprachregelung: Eine Tautologie ist **tautologisch**.
- ▶ Eine Tautologie ist unter allen Umständen wahr, unabhängig von der Belegung der Variablen
- ▶ Beispiele:
  - ▶  $\top$
  - ▶  $A \vee \neg A$
  - ▶  $A \rightarrow A$
  - ▶  $A \rightarrow (B \rightarrow A)$

# Übung: Formalisierung von Raubüberfällen

Mr. McGregor, ein Londoner Ladeninhaber, rief bei Scotland Yard an und teilte mit, dass sein Laden ausgeraubt worden sei. Drei Verdächtige, A, B und C, wurden zum Verhör geholt. Folgende Tatbestände wurden ermittelt:

1. Jeder der Männer A, B und C war am Tag des Geschehens in dem Laden gewesen, und kein anderer hatte den Laden an dem Tag betreten.
  2. Wenn A schuldig ist, so hat er genau einen Komplizen.
  3. Wenn B unschuldig ist, so ist auch C unschuldig.
  4. Wenn genau zwei schuldig sind, dann ist A einer von ihnen.
  5. Wenn C unschuldig ist, so ist auch B unschuldig.
- ▶ Beschreiben Sie die Ermittlungsergebnisse als logische Formeln.
  - ▶ Lösen sie das Verbrechen!

nach R. Smullyan: „Wie heißt dieses Buch?“

# Unerfüllbarkeit

Definition: (Unerfüllbarkeit)

- ▶ Eine Formel  $F \in For_{0\Sigma}$  heißt **unerfüllbar**, falls sie kein Modell hat, d.h. falls  $val_I(F) = 0$  für jede Interpretation  $I$ .
- ▶ Eine Formelmenge  $A \subseteq For_{0\Sigma}$  heißt **unerfüllbar**, wenn es für  $A$  kein Modell gibt, d.h. keine Interpretation, die alle Formeln in  $A$  zu 1 auswertet.
- ▶ Sprachregelung: Eine unerfüllbare Formel oder Formelmenge heißt auch **widersprüchlich** oder **inkonsistent**.
- ▶ Beispiele?
  - ▶  $\perp$

# Unerfüllbarkeit

Definition: (Unerfüllbarkeit)

- ▶ Eine Formel  $F \in For_{0\Sigma}$  heißt **unerfüllbar**, falls sie kein Modell hat, d.h. falls  $val_I(F) = 0$  für jede Interpretation  $I$ .
  - ▶ Eine Formelmenge  $A \subseteq For_{0\Sigma}$  heißt **unerfüllbar**, wenn es für  $A$  kein Modell gibt, d.h. keine Interpretation, die alle Formeln in  $A$  zu 1 auswertet.
- 
- ▶ Sprachregelung: Eine unerfüllbare Formel oder Formelmenge heißt auch **widersprüchlich** oder **inkonsistent**.
  - ▶ Beispiele?
    - ▶  $\perp$
    - ▶  $A \wedge \neg A$

# Unerfüllbarkeit

Definition: (Unerfüllbarkeit)

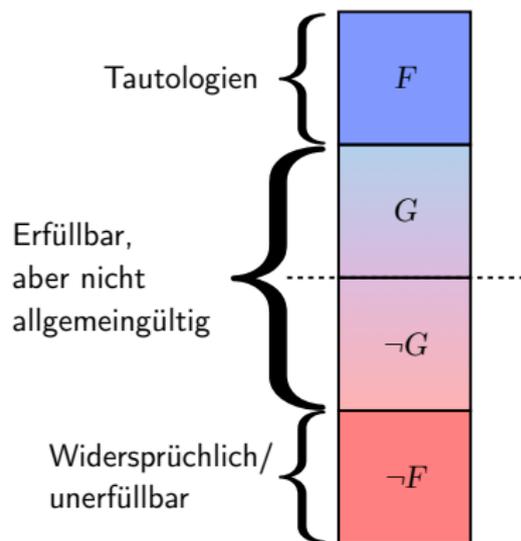
- ▶ Eine Formel  $F \in For_{0\Sigma}$  heißt **unerfüllbar**, falls sie kein Modell hat, d.h. falls  $val_I(F) = 0$  für jede Interpretation  $I$ .
  - ▶ Eine Formelmenge  $A \subseteq For_{0\Sigma}$  heißt **unerfüllbar**, wenn es für  $A$  kein Modell gibt, d.h. keine Interpretation, die alle Formeln in  $A$  zu 1 auswertet.
- 
- ▶ Sprachregelung: Eine unerfüllbare Formel oder Formelmenge heißt auch **widersprüchlich** oder **inkonsistent**.
  - ▶ Beispiele?
    - ▶  $\perp$
    - ▶  $A \wedge \neg A$
    - ▶  $(\neg A) \leftrightarrow A$

# Unerfüllbarkeit

Definition: (Unerfüllbarkeit)

- ▶ Eine Formel  $F \in For_{0\Sigma}$  heißt **unerfüllbar**, falls sie kein Modell hat, d.h. falls  $val_I(F) = 0$  für jede Interpretation  $I$ .
  - ▶ Eine Formelmenge  $A \subseteq For_{0\Sigma}$  heißt **unerfüllbar**, wenn es für  $A$  kein Modell gibt, d.h. keine Interpretation, die alle Formeln in  $A$  zu 1 auswertet.
- 
- ▶ Sprachregelung: Eine unerfüllbare Formel oder Formelmenge heißt auch **widersprüchlich** oder **inkonsistent**.
  - ▶ Beispiele?
    - ▶  $\perp$
    - ▶  $A \wedge \neg A$
    - ▶  $(\neg A) \leftrightarrow A$
    - ▶  $\{(A \vee B), (A \vee \neg B), (\neg A \vee B), (\neg A \vee \neg B)\}$

# (Un)erfüllbarkeit und Allgemeingültigkeit

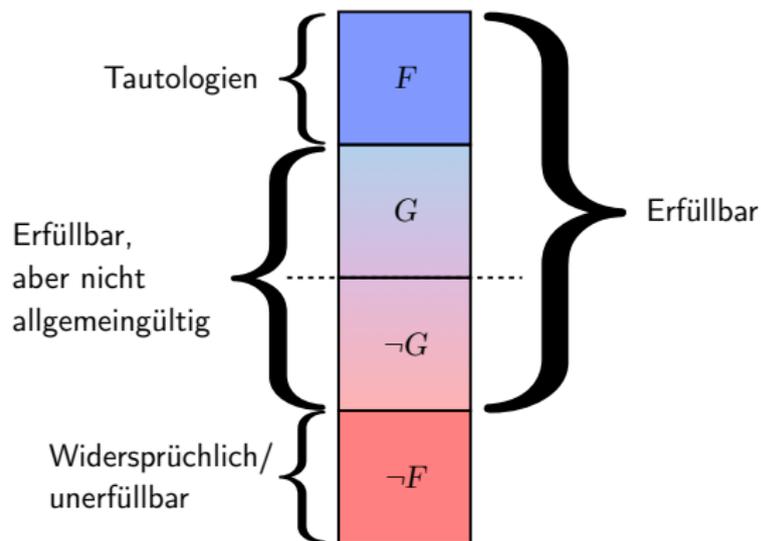


Formeln fallen in 3 Klassen:

- ▶ Allgemeingültig
- ▶ Unerfüllbar
- ▶ „Der Rest“ - erfüllbar, aber nicht allgemeingültig

- ▶ Die Negation einer allgemeingültigen Formel ist unerfüllbar
- ▶ Die Negation einer unerfüllbaren Formel ist allgemeingültig
- ▶ Die Negation einer „Rest“-Formel ist wieder im Rest!

# (Un)erfüllbarkeit und Allgemeingültigkeit



Formeln fallen in 3 Klassen:

- ▶ Allgemeingültig
- ▶ Unerfüllbar
- ▶ „Der Rest“ - erfüllbar, aber nicht allgemeingültig

- ▶ Die Negation einer allgemeingültigen Formel ist unerfüllbar
- ▶ Die Negation einer unerfüllbaren Formel ist allgemeingültig
- ▶ Die Negation einer „Rest“-Formel ist wieder im Rest!
- ▶ Speziell: die Negation einer erfüllbaren Formel kann erfüllbar sein!

**Satz (Dualität von Unerfüllbarkeit und Allgemeingültigkeit):** Sei  $F \in \text{For}_{0\Sigma}$  eine Formel. Dann gilt:  $F$  ist eine Tautologie gdw.  $(\neg F)$  unerfüllbar ist.

- ▶ Beweis („ $\implies$ “): Sei  $F$  eine Tautologie. Zu zeigen ist, dass  $I((\neg F)) = 0$  für alle Interpretationen  $I$ . Sei  $I$  also eine beliebige Interpretation.
  - ▶  $F$  allgemeingültig  $\rightsquigarrow I(F) = 1$
  - ▶  $\rightsquigarrow I((\neg F)) = 0$  (per Definition Evaluierungsfunktion)
  - ▶ Da  $I$  beliebig, gilt das Ergebnis für alle  $I$ .
- ▶ Beweis („ $\impliedby$ “): Sei  $(\neg F)$  unerfüllbar. Sei  $I$  also eine beliebige Interpretation.
  - ▶  $(\neg F)$  unerfüllbar  $\rightsquigarrow I((\neg F)) = 0$
  - ▶  $\rightsquigarrow I(F) = 1$  (per Definition Evaluierungsfunktion)
  - ▶ Da  $I$  beliebig, gilt das Ergebnis für alle  $I$ .
- ▶ Aus  $\implies$  und  $\impliedby$  folgt der Satz. q.e.d.

## Definition (logische Folgerung):

Eine Formel  $A$  folgt logisch aus Formelmenge  $KB$   
gdw.

alle Modelle von  $KB$  sind auch Modell von  $A$ .

Schreibweise:  $KB \models A$

- ▶ Der Folgerungsbegriff ist zentral in der Logik und ihren Anwendungen!
- ▶ Beispiele:
  - ▶ Folgt aus dem Verhandensein von bestimmten Mutationen eine Erkrankung?
  - ▶ Folgt aus der Spezifikation einer Schaltung das gewünscht Verhalten?
  - ▶ Folgt aus einer Reihe von Indizien die Schuld eines Angeklagten?
  - ▶ ...

# Die Wahrheitstafelmethode

- ▶ Wir wollen zeigen: Aus einer Formelmenge  $KB$  folgt eine Vermutung  $F$ .
- ▶ Wahrheitstafelmethode: Direkte Umsetzung der Definition von  $KB \models F$ 
  - ▶ Enumeriere alle Interpretationen in einer Tabelle
  - ▶ Für jede Interpretation:
    - ▶ Bestimme  $I(G)$  für alle  $G \in KB$
    - ▶ Bestimme  $I(F)$
    - ▶ Prüfe, ob jedes Modell von  $KB$  auch ein Modell von  $F$  ist

# Folgerungsproblem von Craig

- ▶ Inspektor Craigs Ergebnisse
  1. Wenn A schuldig und B unschuldig ist, so ist C schuldig.
  2. C arbeitet niemals allein.
  3. A arbeitet niemals mit C.
  4. Niemand außer A, B oder C war beteiligt, und mindestens einer von ihnen ist schuldig.
- ▶ Ermittlungsergebnisse formal
  1.  $(A \wedge \neg B) \rightarrow C$
  2.  $C \rightarrow (A \vee B)$
  3.  $A \rightarrow \neg C$
  4.  $A \vee B \vee C$
- ▶ Craigs erstes Problem: Sei  $KB = \{1., 2., 3., 4.\}$ . Gilt einer der folgenden Fälle?
  - ▶  $KB \models A$
  - ▶  $KB \models B$
  - ▶  $KB \models C$

# Schritt 1: Aufzählung aller möglichen Welten

## ► Ermittlungsergebnisse

1.  $(A \wedge \neg B) \rightarrow C$
2.  $C \rightarrow (A \vee B)$
3.  $A \rightarrow \neg C$
4.  $A \vee B \vee C$

## ► Vorgehen

- Enumeriere alle Interpretationen  $I$
- Berechne  $I(F)$  für alle  $F \in KB$
- Bestimme Modelle von  $KB$

| A | B | C | 1. | 2. | 3. | 4. | KB | Kommentar |
|---|---|---|----|----|----|----|----|-----------|
| 0 | 0 | 0 | 1  | 1  | 1  | 0  | 0  |           |
| 0 | 0 | 1 | 1  | 0  | 1  | 1  | 0  |           |
| 0 | 1 | 0 | 1  | 1  | 1  | 1  | 1  | Modell KB |
| 0 | 1 | 1 | 1  | 1  | 1  | 1  | 1  | Modell KB |
| 1 | 0 | 0 | 0  | 1  | 1  | 1  | 0  |           |
| 1 | 0 | 1 | 1  | 1  | 0  | 1  | 0  |           |
| 1 | 1 | 0 | 1  | 1  | 1  | 1  | 1  | Modell KB |
| 1 | 1 | 1 | 1  | 1  | 0  | 1  | 0  |           |

## Vermutung 1: A ist schuldig!

- ▶ Vermutung: A ist schuldig
- ▶ Prüfe, ob jedes Modell von  $KB$  auch ein Modell von  $A$  ist

| A | B | C | 1. | 2. | 3. | 4. | KB | Kommentar | A |
|---|---|---|----|----|----|----|----|-----------|---|
| 0 | 0 | 0 | 1  | 1  | 1  | 0  | 0  |           | 0 |
| 0 | 0 | 1 | 1  | 0  | 1  | 1  | 0  |           | 0 |
| 0 | 1 | 0 | 1  | 1  | 1  | 1  | 1  | Modell KB | 0 |
| 0 | 1 | 1 | 1  | 1  | 1  | 1  | 1  | Modell KB | 0 |
| 1 | 0 | 0 | 0  | 1  | 1  | 1  | 0  |           | 1 |
| 1 | 0 | 1 | 1  | 1  | 0  | 1  | 0  |           | 1 |
| 1 | 1 | 0 | 1  | 1  | 1  | 1  | 1  | Modell KB | 1 |
| 1 | 1 | 1 | 1  | 1  | 0  | 1  | 0  |           | 1 |

- ▶ Das ist nicht der Fall, es gilt also nicht  $KB \models A$
- ▶ Schreibweise auch  $KB \not\models A$ )

## Vermutung 2: $B$ ist schuldig!

- ▶ Vermutung:  $B$  ist schuldig
- ▶ Prüfe, ob jedes Modell von  $KB$  auch ein Modell von  $B$  ist

| <b>A</b> | <b>B</b> | <b>C</b> | <b>1.</b> | <b>2.</b> | <b>3.</b> | <b>4.</b> | <b>KB</b> | Kommentar | <b>B</b> |
|----------|----------|----------|-----------|-----------|-----------|-----------|-----------|-----------|----------|
| 0        | 0        | 0        | 1         | 1         | 1         | 0         | 0         |           | 0        |
| 0        | 0        | 1        | 1         | 0         | 1         | 1         | 0         |           | 0        |
| 0        | 1        | 0        | 1         | 1         | 1         | 1         | 1         | Modell KB | 1        |
| 0        | 1        | 1        | 1         | 1         | 1         | 1         | 1         | Modell KB | 1        |
| 1        | 0        | 0        | 0         | 1         | 1         | 1         | 0         |           | 0        |
| 1        | 0        | 1        | 1         | 1         | 0         | 1         | 0         |           | 0        |
| 1        | 1        | 0        | 1         | 1         | 1         | 1         | 1         | Modell KB | 1        |
| 1        | 1        | 1        | 1         | 1         | 0         | 1         | 0         |           | 1        |

- ▶ Das ist der Fall, es gilt also  $KB \models A$
- ▶ In allen möglichen Welten, in denen die Annahmen gelten, ist  $B$  schuldig!

## Komplexere Vermutung

- ▶ Vermutung:  $A$  oder  $B$  haben das Verbrechen begangen
- ▶ Prüfe, ob jedes Modell von  $KB$  auch ein Modell von  $A \vee B$  ist

| <b>A</b> | <b>B</b> | <b>C</b> | <b>1.</b> | <b>2.</b> | <b>3.</b> | <b>4.</b> | <b>KB</b> | Kommentar | $A \vee B$ |
|----------|----------|----------|-----------|-----------|-----------|-----------|-----------|-----------|------------|
| 0        | 0        | 0        | 1         | 1         | 1         | 0         | 0         |           | 0          |
| 0        | 0        | 1        | 1         | 0         | 1         | 1         | 0         |           | 0          |
| 0        | 1        | 0        | 1         | 1         | 1         | 1         | 1         | Modell KB | 1          |
| 0        | 1        | 1        | 1         | 1         | 1         | 1         | 1         | Modell KB | 1          |
| 1        | 0        | 0        | 0         | 1         | 1         | 1         | 0         |           | 1          |
| 1        | 0        | 1        | 1         | 1         | 0         | 1         | 0         |           | 1          |
| 1        | 1        | 0        | 1         | 1         | 1         | 1         | 1         | Modell KB | 1          |
| 1        | 1        | 1        | 1         | 1         | 0         | 1         | 0         |           | 1          |

- ▶ Das ist der Fall, es gilt also (logisch)  $KB \models (A \vee B)$

## Übung: Folgerung

1. Wenn Jane nicht krank ist und zum Meeting eingeladen wird, dann kommt sie zu dem Meeting.
2. Wenn der Boss Jane im Meeting haben will, lädt er sie ein.
3. Wenn der Boss Jane nicht im Meeting haben will, fliegt sie raus.
4. Jane war nicht im Meeting.
5. Jane war nicht krank.
6. **Vermutung:** Jane fliegt raus.

Formalisieren Sie das Problem und zeigen oder widerlegen Sie die Vermutung!

## Übung: Folgerung

1. Wenn Jane nicht krank ist und zum Meeting eingeladen wird, dann kommt sie zu dem Meeting.
  - ▶  $\neg K \wedge E \rightarrow M$
2. Wenn der Boss Jane im Meeting haben will, lädt er sie ein.
  - ▶  $B \rightarrow E$
3. Wenn der Boss Jane nicht im Meeting haben will, fliegt sie raus.
  - ▶  $\neg B \rightarrow F$
4. Jane war nicht im Meeting.
  - ▶  $\neg M$
5. Jane war nicht krank.
  - ▶  $\neg K$
6. **Vermutung:** Jane fliegt raus.
  - ▶  $F$

Formalisieren Sie das Problem und zeigen oder widerlegen Sie die Vermutung!

**Satz (Deduktionstheorem):**  $F_1, \dots, F_n \models G$  gdw.  
 $\models (F_1 \wedge \dots \wedge F_n) \rightarrow G$

**Satz (Deduktionstheorem):**  $F_1, \dots, F_n \models G$  gdw.  
 $\models (F_1 \wedge \dots \wedge F_n) \rightarrow G$

- ▶ Also: Eine Formel  $G$  folgt aus einer Formelmenge  $\{F_1, \dots, F_n\}$  genau dann, wenn die Formel  $(F_1 \wedge \dots \wedge F_n) \rightarrow G$  allgemeingültig ist ...

**Satz (Deduktionstheorem):**  $F_1, \dots, F_n \models G$  gdw.  
 $\models (F_1 \wedge \dots \wedge F_n) \rightarrow G$

- ▶ Also: Eine Formel  $G$  folgt aus einer Formelmenge  $\{F_1, \dots, F_n\}$  genau dann, wenn die Formel  $(F_1 \wedge \dots \wedge F_n) \rightarrow G$  allgemeingültig ist ...
- ▶ ... und also genau dann, wenn  $\neg((F_1 \wedge \dots \wedge F_n) \rightarrow G)$  unerfüllbar ist (nach Satz (Unerfüllbarkeit und Allgemeingültigkeit)).

Wir können das Problem der logischen Folgerung reduzieren auf einen Test auf Unerfüllbarkeit!

Wir können in der Aussagenlogik Unerfüllbarkeit von endlichen Formeln mit der Wahrheitstafelmethode entscheiden:

- ▶ Berechnen  $I(F)$  für alle Interpretationen  $I$
- ▶ Falls  $I(F) = 1$  für ein  $I$ , so ist  $F$  erfüllbar, sonst unerfüllbar.

Wir können in der Aussagenlogik Unerfüllbarkeit von endlichen Formeln mit der Wahrheitstafelmethode entscheiden:

- ▶ Berechnen  $I(F)$  für alle Interpretationen  $I$
- ▶ Falls  $I(F) = 1$  für ein  $I$ , so ist  $F$  erfüllbar, sonst unerfüllbar.

Sind wir mit der Aussagenlogik fertig?

# Übung: Aussagenlogik in Scheme

- ▶ Stellen Sie Überlegungen zu folgenden Fragen an:
  - ▶ Wie wollen Sie logische Formeln in Scheme repräsentieren?
  - ▶ Wie wollen Sie Interpretationen in Scheme repräsentieren?
- ▶ Schreiben Sie eine Funktion (`eval-prop formel interpretation`), die den Wert einer Formel unter einer Interpretation berechnet.

- ▶ Anwendungsbereiche
  - ▶ Hardware-Verifikation
  - ▶ Software-Verifikation
  - ▶ Kryptographie
  - ▶ Bio-Informatik
  - ▶ Diagnostik
  - ▶ Konfigurationsmanagement
- ▶ Größe echter Probleme:
  - ▶ Quelle: SAT-RACE 2010,  
<http://baldur.iti.uka.de/sat-race-2010/index.html>
  - ▶ **Kleinstes** CNF-Problem: 1694 aussagenlogische Variable
  - ▶ Größtes Problem: 10 950 109 aussagenlogische Variable
  - ▶ Beweiser im Wettbewerb haben  $\approx 80\%$  Erfolgsquote!

# Anwendungen von Aussagenlogik

- ▶ Anwendungsbereiche
  - ▶ Hardware-Verifikation
  - ▶ Software-Verifikation
  - ▶ Kryptographie
  - ▶ Bio-Informatik
  - ▶ Diagnostik
  - ▶ Konfigurationsmanagement
- ▶ Größe echter Probleme:
  - ▶ Quelle: SAT-RACE 2010,  
<http://baldur.iti.uka.de/sat-race-2010/index.html>
  - ▶ **Kleinstes** CNF-Problem: 1694 aussagenlogische Variable
  - ▶ Größtes Problem: 10 950 109 aussagenlogische Variable
  - ▶ Beweiser im Wettbewerb haben  $\approx 80\%$  Erfolgsquote!

Problem: Die Wahrheitstafelmethode muss immer **alle** Interpretationen betrachten – bei  $n$  verschiedenen Atomen sind das  $2^n$  Möglichkeiten!

- ▶ Betrachte z.B. Formel  $F = (a \wedge \neg b)$
- ▶ Unter welchen Umständen kann  $F$  unter  $I$  zu 1 ausgewertet werden?

- ▶ Betrachte z.B. Formel  $F = (a \wedge \neg b)$
- ▶ Unter welchen Umständen kann  $F$  unter  $I$  zu 1 ausgewertet werden?
  - ▶  $a$  muss zu 1 ausgewertet werden **und**
  - ▶  $\neg b$  muss zu 1 ausgewertet werden

- ▶ Betrachte z.B. Formel  $F = (a \wedge \neg b)$
- ▶ Unter welchen Umständen kann  $F$  unter  $I$  zu 1 ausgewertet werden?
  - ▶  $a$  muss zu 1 ausgewertet werden **und**
  - ▶  $\neg b$  muss zu 1 ausgewertet werden
- ▶ Betrachte z.B. Formel  $F = (a \rightarrow \neg b)$
- ▶ Unter welchen Umständen kann  $F$  unter  $I$  zu 1 ausgewertet werden?

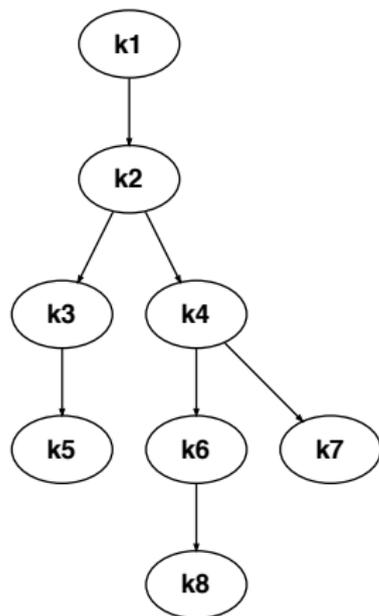
- ▶ Betrachte z.B. Formel  $F = (a \wedge \neg b)$
- ▶ Unter welchen Umständen kann  $F$  unter  $I$  zu 1 ausgewertet werden?
  - ▶  $a$  muss zu 1 ausgewertet werden **und**
  - ▶  $\neg b$  muss zu 1 ausgewertet werden
- ▶ Betrachte z.B. Formel  $F = (a \rightarrow \neg b)$
- ▶ Unter welchen Umständen kann  $F$  unter  $I$  zu 1 ausgewertet werden?
  - ▶  $\neg a$  muss zu 1 ausgewertet werden **oder**
  - ▶  $\neg b$  muss zu 1 ausgewertet werden

- ▶ Betrachte z.B. Formel  $F = (a \wedge \neg b)$
- ▶ Unter welchen Umständen kann  $F$  unter  $I$  zu 1 ausgewertet werden?
  - ▶  $a$  muss zu 1 ausgewertet werden **und**
  - ▶  $\neg b$  muss zu 1 ausgewertet werden
- ▶ Betrachte z.B. Formel  $F = (a \rightarrow \neg b)$
- ▶ Unter welchen Umständen kann  $F$  unter  $I$  zu 1 ausgewertet werden?
  - ▶  $\neg a$  muss zu 1 ausgewertet werden **oder**
  - ▶  $\neg b$  muss zu 1 ausgewertet werden

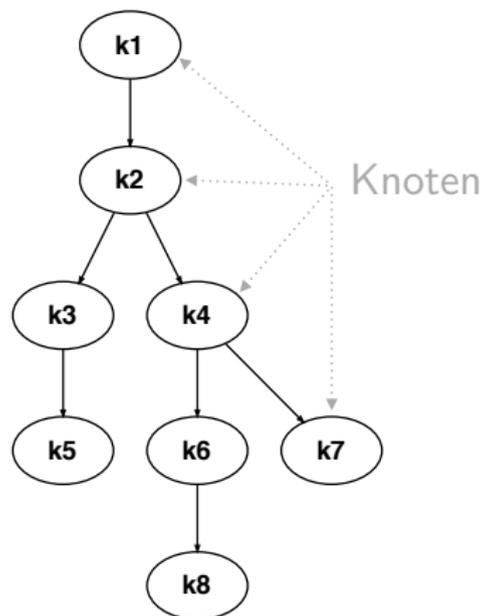
Idee: Zerlege eine Formel systematisch in Teilformeln, so dass die Erfüllbarkeit/Unerfüllbarkeit offensichtlich wird!

## Wesentliche Eigenschaften

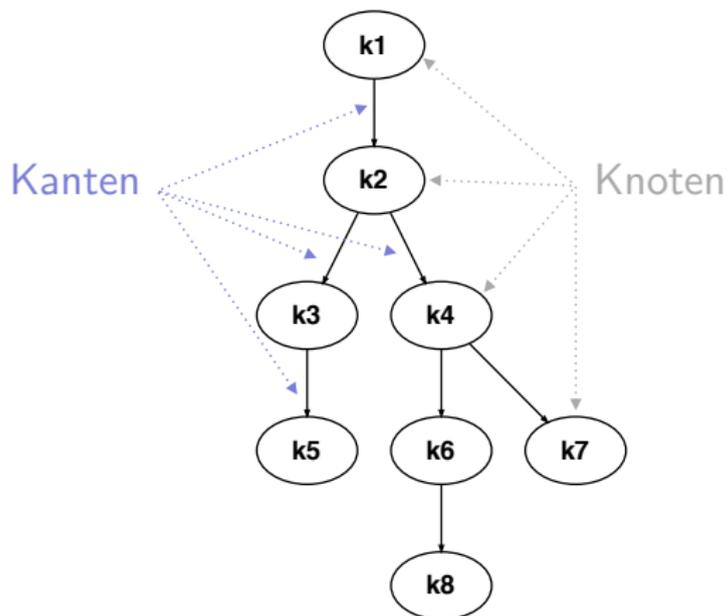
- ▶ Widerlegungskalkül: Versucht die **Unerfüllbarkeit** einer Formel (oder Formelmenge) zu zeigen
- ▶ Beweis durch vollständige **Fallunterscheidung**
- ▶ Sukzessive Zerlegung der Formel in ihre Bestandteile („Analyse“)



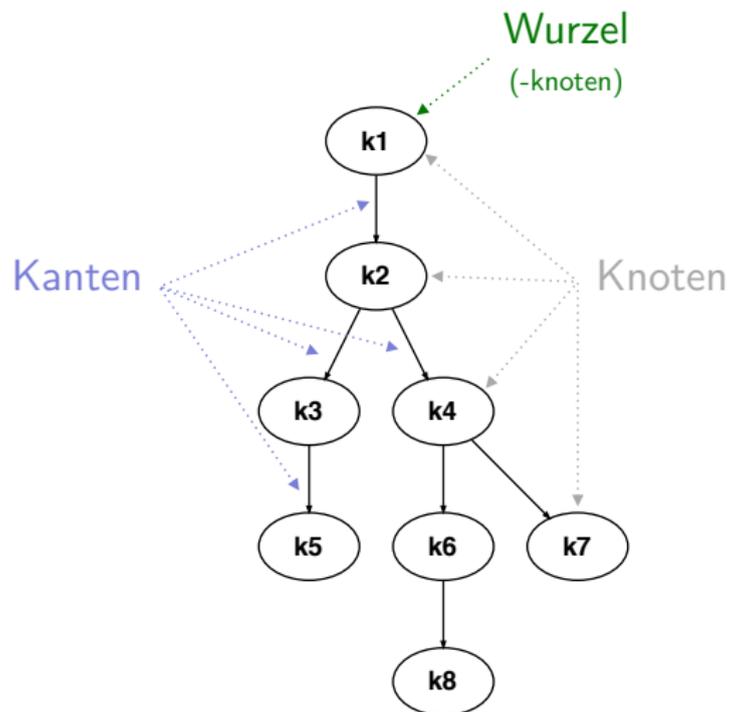
# Exkurs: Bäume



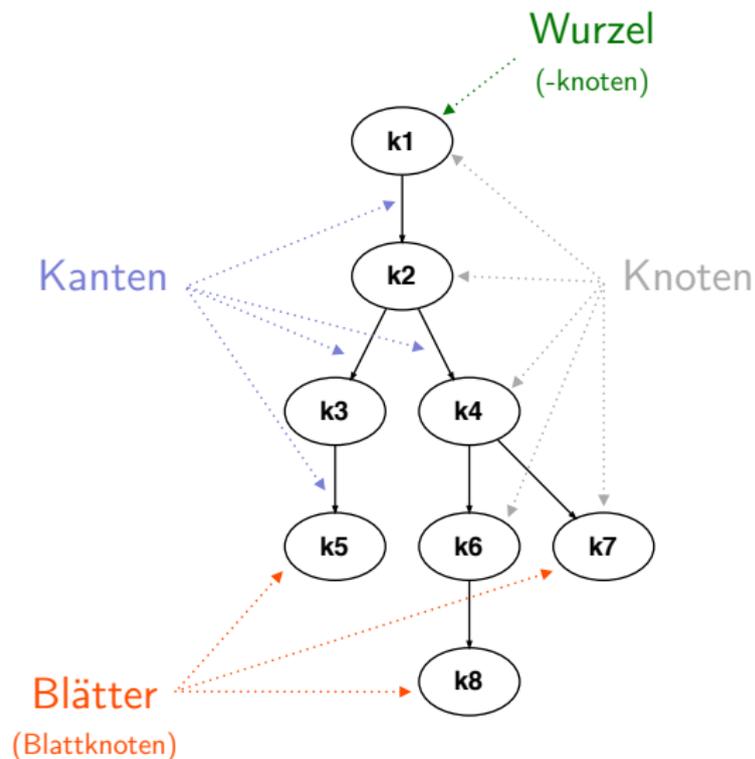
# Exkurs: Bäume

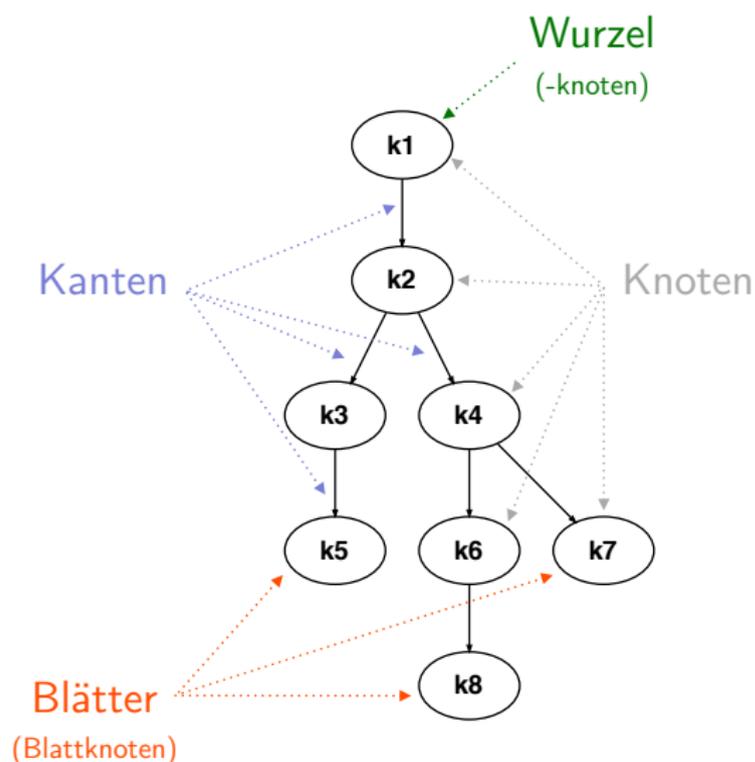


# Exkurs: Bäume



# Exkurs: Bäume





Formal:

- ▶  $T = (V, E)$
- ▶  $V = \{k1, k2, k3, k4, k5, k6, k7, k8\}$
- ▶  $E = \{(k1, k2), (k2, k3), (k2, k4), (k3, k5), (k4, k6), (k4, k7), (k6, k8)\}$
- ▶ k1 ist die **Wurzel**
- ▶ k1, k2, k3, k4, k6 sind **innere Knoten**
- ▶ k5, k8, k7 sind **Blattknoten**

## Definition: Baum

- ▶ Sei  $V$  eine beliebige Menge (die Knoten, **Vertices**) und  $E \subseteq V \times V$  (die Kanten, **Edges**) eine zweistellige Relation über  $E$ .  
Dann ist das Tupel  $(V, E)$  ein (ungeordneter) **Baum**, wenn folgende Eigenschaften gelten:
  - ▶ Es existiert ein ausgezeichnetes Element  $r \in V$  (die **Wurzel**), so dass kein Knoten  $x \in V$  mit  $(x, r) \in E$  existiert.  
„Die Wurzel hat keinen Vorgänger.“
  - ▶ Für alle  $k \in V \setminus \{r\}$  gilt: Es gibt genau ein  $x \in V$  mit  $(x, k) \in E$ .  
„Jeder Knoten außer der Wurzel hat einen eindeutigen Vorgänger.“

## Bäume (2)

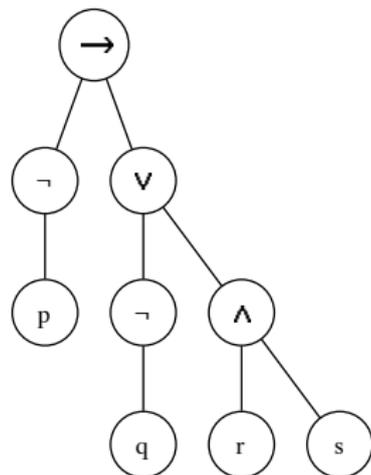
**Definition:** Pfad, Ast, Binärbaum. . .

Sei  $T = (V, E)$  ein Baum. Wir definieren:

- ▶ Ist  $(a, b) \in E$ , so heißt  $a$  Vorgänger von  $b$  und  $b$  Nachfolger von  $a$ .
- ▶ Ein **Blatt** in  $T$  ist ein Knoten ohne Nachfolger.
- ▶ Ein **innerer Knoten** ist ein Knoten, der kein Blatt ist.
- ▶ Ein **Pfad** in  $T$  ist eine Sequenz von Knoten  $p = \langle k_0, k_1, \dots, k_n \rangle$  ( $m \in \mathbb{N}$ ) mit der Eigenschaft, dass  $(k_i, k_{i+1}) \in E$  für alle  $i$  zwischen 0 und  $n - 1$ .
- ▶ Ein **Ast** oder **maximaler Pfad** ist ein Pfad, bei dem der erste Knoten die Wurzel und der letzte Knoten ein Blatt ist.
- ▶ Ein Baum, bei dem jeder Knoten maximal zwei Nachfolger hat, heißt **Binärbaum** oder **dyadischer Baum**.

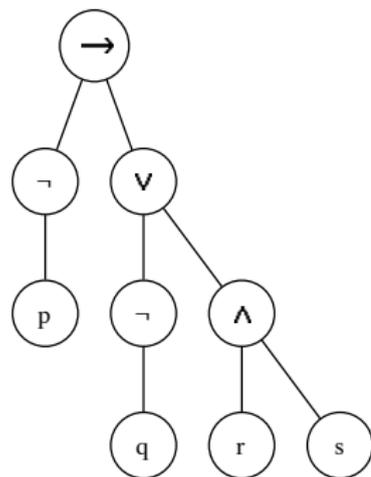
## ► Graphische Repräsentation

- Informatische Bäume wachsen (meist) von oben nach unten.
- Die Richtung der Nachfolger-Relation ist durch die Höhe vorgegeben (Pfeile sind unnötig).
- Eine Ordnung unter den Nachfolgern ist durch die Anordnung der Äste implizit gegeben.
- Knoten können in der graphischen Darstellung die gleiche Beschriftung tragen, aber doch verschieden sein.



Baumdarstellung von  
 $(\neg p \rightarrow (\neg q \vee (r \wedge s)))$

# Übung: Bäume



- ▶ Wie viele Äste hat der Baum (und welche)?
- ▶ Wie viele innere Knoten hat der Baum?
- ▶ Was sind die Nachfolger der Wurzel?
- ▶ Gibt es Pfade, die keine Äste sind? Beispiele?
- ▶ Ist der Baum binär?

Baumdarstellung von  
 $(\neg p \rightarrow (\neg q \vee (r \wedge s)))$

# Analytische Tableaux

- ▶ Ein Tableau ist ein binärer Baum, bei dem die Knoten mit Formeln beschriftet sind.
- ▶ Ein Tableau kann wie folgt erweitert werden:
  - ▶ Tableau-Formeln werden zerlegt
  - ▶ Die Teilformeln werden unter den Blättern angehängt
- ▶ Ein vollständiges Tableau ist entweder
  - ▶ offen – dann kann man eine erfüllende Interpretation für die ursprüngliche Formel ablesen – oder
  - ▶ geschlossen – dann ist die ursprüngliche Formel unerfüllbar
- ▶ Historisch:
  - ▶ Evert Willem Beth: Idee der Semantischen Tableaux
  - ▶ Jaakko Hintikka: Hintikka-Mengen, Hintikka-Tableaux
  - ▶ Raymond Smullyan: Heutige Form der Tableaux

Einzahl: Tableau, Mehrzahl: Tableaux - gesprochen ungefähr gleich. . .



E.W. Beth



Jaakko Hintikka

## Fallunterscheidung nach Formeltyp

- ▶ **Konjunktive** Formeln
  - ▶ Zerlegbar. Um zu 1 ausgewertet zu werden, bestehen Anforderungen an **alle Teile**
  - ▶ Beispiel:  $(a \wedge b)$
- ▶ **Disjunktive** Formeln
  - ▶ Zerlegbar. Um zu 1 ausgewertet zu werden, besteht Anforderung an **mindestens einen Teil**
  - ▶ Beispiel:  $(a \vee b)$
- ▶ **Literale**
  - ▶ Im Tableaux-Kalkül nicht weiter zerlegbar (aber alleine immer erfüllbar)
  - ▶ Beispiele:  $a, \neg b$

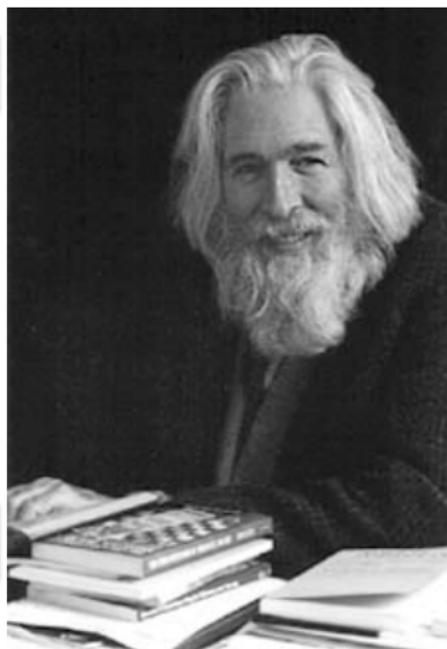
# Uniforme Notation: $\alpha$ und $\beta$

## Konjunktive Formeln: Typ $\alpha$

- ▶  $\neg\neg A$
- ▶  $A \wedge B$
- ▶  $\neg(A \vee B)$
- ▶  $\neg(A \rightarrow B)$
- ▶  $A \leftrightarrow B$

## Disjunktive Formeln: Typ $\beta$

- ▶  $\neg(A \wedge B)$
- ▶  $A \vee B$
- ▶  $A \rightarrow B$
- ▶  $\neg(A \leftrightarrow B)$



# Uniforme Notation: Zerlegung

## Zuordnungsregeln Formeln / Unterformeln

| $\alpha$                | $\alpha_1$        | $\alpha_2$        |
|-------------------------|-------------------|-------------------|
| $A \wedge B$            | $A$               | $B$               |
| $\neg(A \vee B)$        | $\neg A$          | $\neg B$          |
| $\neg(A \rightarrow B)$ | $A$               | $\neg B$          |
| $A \leftrightarrow B$   | $A \rightarrow B$ | $B \rightarrow A$ |
| $\neg\neg A$            | $A$               | $A$               |

# Uniforme Notation: Zerlegung

## Zuordnungsregeln Formeln / Unterformeln

| $\alpha$                | $\alpha_1$        | $\alpha_2$        | $\beta$                     | $\beta_1$         | $\beta_2$         |
|-------------------------|-------------------|-------------------|-----------------------------|-------------------|-------------------|
| $A \wedge B$            | $A$               | $B$               | $\neg(A \wedge B)$          | $\neg A$          | $\neg B$          |
| $\neg(A \vee B)$        | $\neg A$          | $\neg B$          | $A \vee B$                  | $A$               | $B$               |
| $\neg(A \rightarrow B)$ | $A$               | $\neg B$          | $A \rightarrow B$           | $\neg A$          | $B$               |
| $A \leftrightarrow B$   | $A \rightarrow B$ | $B \rightarrow A$ | $\neg(A \leftrightarrow B)$ | $A \wedge \neg B$ | $\neg A \wedge B$ |
| $\neg\neg A$            | $A$               | $A$               |                             |                   |                   |

# Uniforme Notation: Zerlegung

## Zuordnungsregeln Formeln / Unterformeln

| $\alpha$                | $\alpha_1$        | $\alpha_2$        | $\beta$                     | $\beta_1$         | $\beta_2$         |
|-------------------------|-------------------|-------------------|-----------------------------|-------------------|-------------------|
| $A \wedge B$            | $A$               | $B$               | $\neg(A \wedge B)$          | $\neg A$          | $\neg B$          |
| $\neg(A \vee B)$        | $\neg A$          | $\neg B$          | $A \vee B$                  | $A$               | $B$               |
| $\neg(A \rightarrow B)$ | $A$               | $\neg B$          | $A \rightarrow B$           | $\neg A$          | $B$               |
| $A \leftrightarrow B$   | $A \rightarrow B$ | $B \rightarrow A$ | $\neg(A \leftrightarrow B)$ | $A \wedge \neg B$ | $\neg A \wedge B$ |
| $\neg\neg A$            | $A$               | $A$               |                             |                   |                   |

- ▶  $\alpha$  ist wahr, wenn  $\alpha_1$  und  $\alpha_2$  wahr sind
- ▶  $\beta$  ist wahr, wenn  $\beta_1$  oder  $\beta_2$  wahr ist

## Definition (Tableaux-Kalkül für Aussagenlogik):

- ▶ Der Tableaux-Kalkül für die Aussagenlogik umfasst die folgenden Regeln:
  - ▶ Startregel: Erzeuge ein initiales Tableau mit einem Knoten, der mit der zu untersuchenden Formel  $F$  beschriftet ist
  - ▶  $\alpha$ -Regel: 
$$\frac{\alpha}{\alpha_1 \quad \alpha_2}$$
  - ▶  $\beta$ -Regel: 
$$\frac{\beta}{\beta_1 \mid \beta_2}$$
  - ▶ Schluss-Regeln: 
$$\frac{A \quad \neg A}{\times} \qquad \frac{\neg A \quad A}{\times}$$

# Erläuterungen zu den Regeln

►  $\alpha$ -Regel:

- Wähle einen beliebigen Knoten mit einer  $\alpha$ -Formel  $A$ , auf die noch keine Regel angewendet wurde
- Erweitere **jeden** offenen Ast, der durch den Knoten  $A$  geht, durch Anhängen von

$$\begin{array}{c} | \\ \alpha_1 \\ | \\ \alpha_2 \end{array}$$

►  $\beta$ -Regel:

- Wähle einen beliebigen Knoten mit einer  $\beta$ -Formel  $A$ , auf die noch keine Regel angewendet wurde
- Erweitere **jeden** offenen Ast, der durch den Knoten  $A$  geht, durch Anhängen von

$$\begin{array}{c} / \ \backslash \\ \beta_1 \ \beta_2 \end{array}$$

- Schluss-Regel: Ein Ast, auf dem eine Formel und ihre Negation vorkommt, kann als geschlossen markiert werden.

## **Definition (Tableaux für eine Formel):**

Sei  $F \in For_0\Sigma$  eine Formel der Aussagenlogik.

- ▶ Der Baum mit einem Knoten, der mit  $F$  beschriftet ist, ist ein Tableau für  $F$
- ▶ Wenn  $T$  ein Tableau für  $F$  ist, und  $T'$  durch Anwenden einer Tableaux-Regel auf  $T$  entsteht, dann ist  $T'$  ein Tableau für  $F$
- ▶ Eine Ableitung im Tableaux-Kalküle erzeugt also eine Folge von Tableaux
- ▶ Ein Nachfolgetableau ist entweder eine Erweiterung des Vorgängers, oder entsteht durch Markierung eines Astes als geschlossen

# Tableaux-Beispiel

$(\neg((p \vee (q \wedge r)) \rightarrow ((p \vee q) \wedge (p \vee r))))$

| $\alpha$                | $\alpha_1$        | $\alpha_2$        |
|-------------------------|-------------------|-------------------|
| $A \wedge B$            | $A$               | $B$               |
| $\neg(A \vee B)$        | $\neg A$          | $\neg B$          |
| $\neg(A \rightarrow B)$ | $A$               | $\neg B$          |
| $A \leftrightarrow B$   | $A \rightarrow B$ | $B \rightarrow A$ |
| $\neg\neg A$            | $A$               | $A$               |

| $\beta$                     | $\beta_1$         | $\beta_2$         |
|-----------------------------|-------------------|-------------------|
| $\neg(A \wedge B)$          | $\neg A$          | $\neg B$          |
| $A \vee B$                  | $A$               | $B$               |
| $A \rightarrow B$           | $\neg A$          | $B$               |
| $\neg(A \leftrightarrow B)$ | $A \wedge \neg B$ | $\neg A \wedge B$ |

# Tableaux-Beispiel

$(\neg((p \vee (q \wedge r)) \rightarrow ((p \vee q) \wedge (p \vee r)))) \quad \checkmark$

|

$(p \vee (q \wedge r))$

|

$(\neg((p \vee q) \wedge (p \vee r)))$

| $\alpha$                | $\alpha_1$        | $\alpha_2$        |
|-------------------------|-------------------|-------------------|
| $A \wedge B$            | $A$               | $B$               |
| $\neg(A \vee B)$        | $\neg A$          | $\neg B$          |
| $\neg(A \rightarrow B)$ | $A$               | $\neg B$          |
| $A \leftrightarrow B$   | $A \rightarrow B$ | $B \rightarrow A$ |
| $\neg\neg A$            | $A$               | $A$               |

| $\beta$                     | $\beta_1$         | $\beta_2$         |
|-----------------------------|-------------------|-------------------|
| $\neg(A \wedge B)$          | $\neg A$          | $\neg B$          |
| $A \vee B$                  | $A$               | $B$               |
| $A \rightarrow B$           | $\neg A$          | $B$               |
| $\neg(A \leftrightarrow B)$ | $A \wedge \neg B$ | $\neg A \wedge B$ |

# Tableaux-Beispiel

$(\neg((p \vee (q \wedge r)) \rightarrow ((p \vee q) \wedge (p \vee r)))) \quad \checkmark$

$$\begin{array}{c}
 | \\
 (p \vee (q \wedge r)) \quad \checkmark \\
 | \\
 (\neg((p \vee q) \wedge (p \vee r))) \\
 / \quad \backslash \\
 p \qquad (q \wedge r)
 \end{array}$$

| $\alpha$                | $\alpha_1$        | $\alpha_2$        |
|-------------------------|-------------------|-------------------|
| $A \wedge B$            | $A$               | $B$               |
| $\neg(A \vee B)$        | $\neg A$          | $\neg B$          |
| $\neg(A \rightarrow B)$ | $A$               | $\neg B$          |
| $A \leftrightarrow B$   | $A \rightarrow B$ | $B \rightarrow A$ |
| $\neg\neg A$            | $A$               | $A$               |

| $\beta$                     | $\beta_1$         | $\beta_2$         |
|-----------------------------|-------------------|-------------------|
| $\neg(A \wedge B)$          | $\neg A$          | $\neg B$          |
| $A \vee B$                  | $A$               | $B$               |
| $A \rightarrow B$           | $\neg A$          | $B$               |
| $\neg(A \leftrightarrow B)$ | $A \wedge \neg B$ | $\neg A \wedge B$ |

# Tableaux-Beispiel

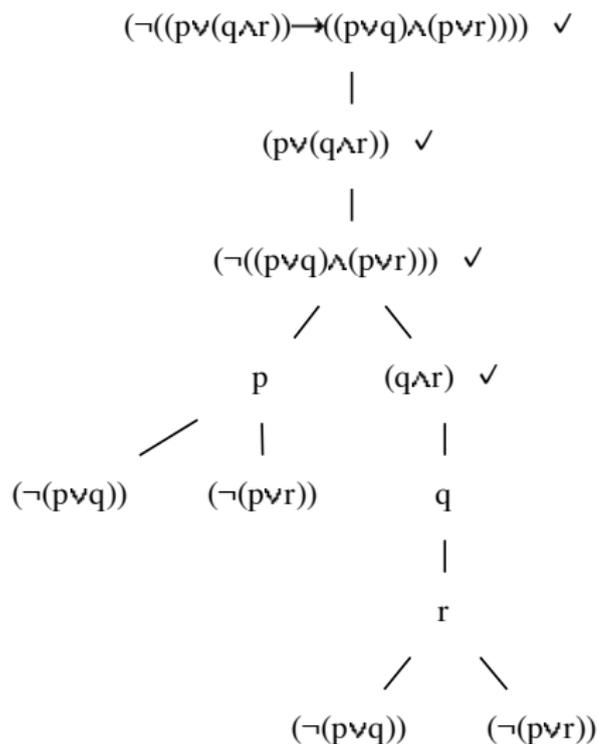
$(\neg((p \vee (q \wedge r)) \rightarrow ((p \vee q) \wedge (p \vee r)))) \quad \checkmark$

$$\begin{array}{c}
 | \\
 (p \vee (q \wedge r)) \quad \checkmark \\
 | \\
 (\neg((p \vee q) \wedge (p \vee r))) \\
 / \quad \backslash \\
 p \qquad (q \wedge r) \quad \checkmark \\
 \qquad | \\
 \qquad q \\
 \qquad | \\
 \qquad r
 \end{array}$$

| $\alpha$                | $\alpha_1$        | $\alpha_2$        |
|-------------------------|-------------------|-------------------|
| $A \wedge B$            | $A$               | $B$               |
| $\neg(A \vee B)$        | $\neg A$          | $\neg B$          |
| $\neg(A \rightarrow B)$ | $A$               | $\neg B$          |
| $A \leftrightarrow B$   | $A \rightarrow B$ | $B \rightarrow A$ |
| $\neg\neg A$            | $A$               | $A$               |

| $\beta$                     | $\beta_1$         | $\beta_2$         |
|-----------------------------|-------------------|-------------------|
| $\neg(A \wedge B)$          | $\neg A$          | $\neg B$          |
| $A \vee B$                  | $A$               | $B$               |
| $A \rightarrow B$           | $\neg A$          | $B$               |
| $\neg(A \leftrightarrow B)$ | $A \wedge \neg B$ | $\neg A \wedge B$ |

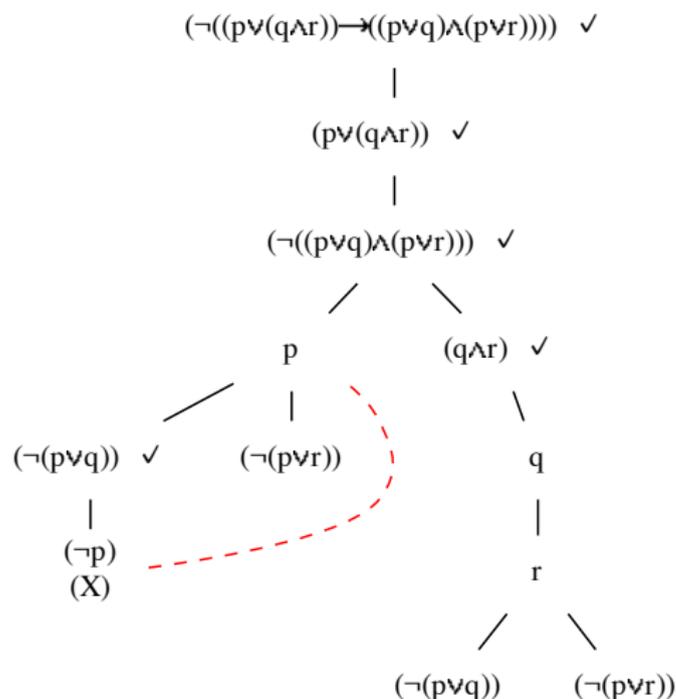
# Tableaux-Beispiel



| $\alpha$                | $\alpha_1$        | $\alpha_2$        |
|-------------------------|-------------------|-------------------|
| $A \wedge B$            | $A$               | $B$               |
| $\neg(A \vee B)$        | $\neg A$          | $\neg B$          |
| $\neg(A \rightarrow B)$ | $A$               | $\neg B$          |
| $A \leftrightarrow B$   | $A \rightarrow B$ | $B \rightarrow A$ |
| $\neg\neg A$            | $A$               | $A$               |

| $\beta$                     | $\beta_1$         | $\beta_2$         |
|-----------------------------|-------------------|-------------------|
| $\neg(A \wedge B)$          | $\neg A$          | $\neg B$          |
| $A \vee B$                  | $A$               | $B$               |
| $A \rightarrow B$           | $\neg A$          | $B$               |
| $\neg(A \leftrightarrow B)$ | $A \wedge \neg B$ | $\neg A \wedge B$ |

# Tableaux-Beispiel

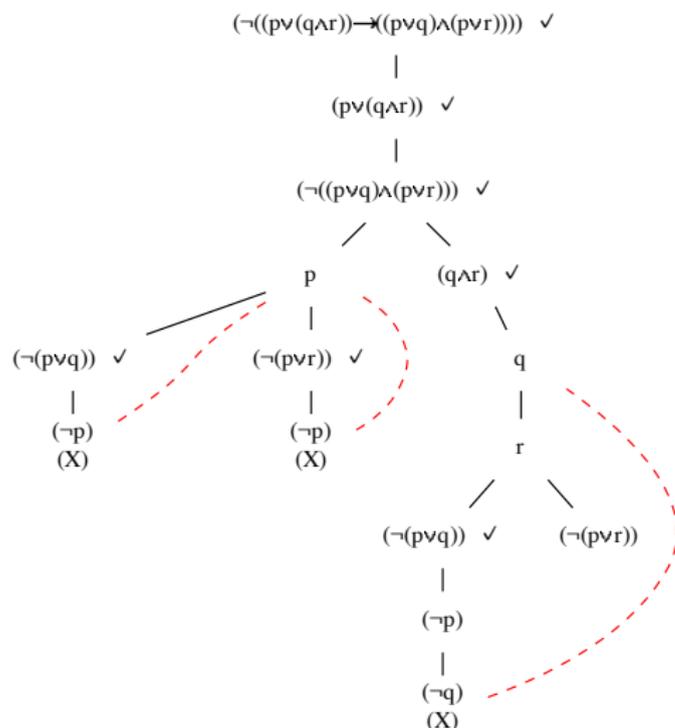


| $\alpha$                | $\alpha_1$        | $\alpha_2$        |
|-------------------------|-------------------|-------------------|
| $A \wedge B$            | $A$               | $B$               |
| $\neg(A \vee B)$        | $\neg A$          | $\neg B$          |
| $\neg(A \rightarrow B)$ | $A$               | $\neg B$          |
| $A \leftrightarrow B$   | $A \rightarrow B$ | $B \rightarrow A$ |
| $\neg\neg A$            | $A$               | $A$               |

| $\beta$                     | $\beta_1$         | $\beta_2$         |
|-----------------------------|-------------------|-------------------|
| $\neg(A \wedge B)$          | $\neg A$          | $\neg B$          |
| $A \vee B$                  | $A$               | $B$               |
| $A \rightarrow B$           | $\neg A$          | $B$               |
| $\neg(A \leftrightarrow B)$ | $A \wedge \neg B$ | $\neg A \wedge B$ |



# Tableaux-Beispiel



| $\alpha$                | $\alpha_1$        | $\alpha_2$        |
|-------------------------|-------------------|-------------------|
| $A \wedge B$            | $A$               | $B$               |
| $\neg(A \vee B)$        | $\neg A$          | $\neg B$          |
| $\neg(A \rightarrow B)$ | $A$               | $\neg B$          |
| $A \leftrightarrow B$   | $A \rightarrow B$ | $B \rightarrow A$ |
| $\neg\neg A$            | $A$               | $A$               |

| $\beta$                     | $\beta_1$         | $\beta_2$         |
|-----------------------------|-------------------|-------------------|
| $\neg(A \wedge B)$          | $\neg A$          | $\neg B$          |
| $A \vee B$                  | $A$               | $B$               |
| $A \rightarrow B$           | $\neg A$          | $B$               |
| $\neg(A \leftrightarrow B)$ | $A \wedge \neg B$ | $\neg A \wedge B$ |



## Definition (Offene und geschlossene Tableaux):

Sei  $T$  ein Tableau für  $F$ .

- ▶ Ein Ast in  $T$  heißt **geschlossen**, wenn er eine Formel  $A$  und ihre Negation  $\neg A$  enthält.
- ▶ Ein Ast heißt **offen**, wenn er nicht geschlossen ist.
- ▶ Ein Tableau heißt geschlossen, wenn alle seine Äste geschlossen sind.
- ▶ Ein Ast heißt **vollständig** oder **saturiert**, wenn alle seine  $\alpha$ - und  $\beta$ -Knoten expandiert sind.
- ▶ Ein Tableau heißt **vollständig**, wenn alle Äste geschlossen oder vollständig sind.

# Übung: Streik!

- ▶ *Quantas: Wenn die Regierung nicht eingreift, dann ist der Streik nicht vorbei, bevor er mindestens ein Jahr andauert und der Firmenchef zurücktritt*
- ▶ Weder liegt der Streikbeginn ein Jahr zurück, noch greift die Regierung ein.
- ▶ Atomare Aussagen?
  - ▶  $p \triangleq$  "Die Regierung greift ein"
  - ▶  $q \triangleq$  "Der Streik ist vorbei"
  - ▶  $r \triangleq$  "Der Streik dauert mindestens ein Jahr an"
  - ▶  $s \triangleq$  "Der Firmenchef tritt zurück"
- ▶ Formalisierung:
  - ▶  $\neg p \rightarrow (\neg q \vee (r \wedge s))$
  - ▶  $\neg(r \vee p)$
- ▶ Aufgabe:
  - ▶ Formalisieren Sie: Aus den Fakten folgt, dass der Streik andauert
  - ▶ Konstruieren Sie **eine** Formel, die **unerfüllbar** ist, wenn die Folgerung gilt.
  - ▶ Leiten Sie ein vollständiges Tableau dazu ab. Ist es geschlossen?

## **Satz (Geschlossene Tableaux):**

- ▶ Sei  $F$  eine Formel und  $T$  ein geschlossenes Tableaux für  $F$ . Dann ist  $F$  unerfüllbar.

## Satz (Geschlossene Tableaux):

- ▶ Sei  $F$  eine Formel und  $T$  ein geschlossenes Tableaux für  $F$ . Dann ist  $F$  unerfüllbar.
- ▶ In diesem Fall heißt  $T$  **Tableaux-Beweis** für (die Unerfüllbarkeit von)  $F$ .

## Satz (Geschlossene Tableaux):

- ▶ Sei  $F$  eine Formel und  $T$  ein geschlossenes Tableaux für  $F$ . Dann ist  $F$  unerfüllbar.
- ▶ In diesem Fall heißt  $T$  **Tableaux-Beweis** für (die Unerfüllbarkeit von)  $F$ .

**Beweis?**

Zurück C

**Definition (Semantik von Ästen):** Sei  $T$  ein Tableau and  $M$  ein Ast in  $T$ .

- ▶ Sei  $I$  eine Interpretation. Der Ast  $M$  heißt **wahr unter  $I$** , wenn alle Formeln auf Knoten in  $M$  unter  $I$  zu 1 ausgewertet werden.  
Schreibweise:  $I(M) = 1$ .
- ▶ Ein Ast  $M$  heißt **erfüllbar**, wenn es eine Interpretation gibt, für die  $I(M) = 1$  gilt.
- ▶ Wir schreiben einen Pfad (oder Ast) als  $\langle F_1, F_2, \dots, F_n \rangle$ , wobei die  $F_i$  die Formeln an den Knoten des Astes sind.

## Lemma (Existenz erfüllbarer Äste):

Sei  $F$  eine Formel mit  $I(F) = 1$  und sei  $T$  ein Tableau für  $F$ . Dann hat  $T$  mindestens einen Ast, der unter  $I$  wahr ist.

- ▶ Beweis per Induktion (nach der Länge der Ableitung):
  - ▶ Wenn  $T$  ein Tableau für  $F$  ist, dann existiert eine Sequenz  $T_0, T_1, \dots, T_n$ , so dass gilt:
    - ▶  $T_0$  ist das initiale Tableau für  $F$ .
    - ▶  $T_n = T$
    - ▶  $T_{i+1}$  entsteht aus  $T_i$  durch Anwendung einer Tableaux-Regel.
- ▶ Induktionsanfang:  $T_0$  ist das initiale Tableau für  $F$ . Der einzige Ast in  $T_0$  ist  $\langle F \rangle$ . Laut Annahme gilt  $I(F) = 1$ , also auch  $I(\langle F \rangle) = 1$ .
- ▶ Induktionsvoraussetzung:  $T_i$  hat Ast  $M$  mit  $I(M) = 1$

## Lemma (Fortsetzung)

- ▶ Induktionsschritt: Gehe  $T_{i+1}$  aus  $T_i$  durch Anwendung einer Tableau-Regel hervor.
- ▶ Fallunterscheidung:
  - 1)  $M$  wird durch die Regel nicht erweitert. Dann ist  $M$  ein Ast in  $T'$ , und  $I(M) = 1$ .
  - 2)  $M$  wird durch die Regel erweitert. Der expandierte Knoten sei  $A$ .  
Fallunterscheidung:
    - a)  $\alpha$ -Regel: Dann ist  $M' = \langle M, \alpha_1, \alpha_2 \rangle$  ein Ast in  $T_{i+1}$ . Da  $I(A) = 1$ , muss auch  $I(\alpha_1) = 1$  und  $I(\alpha_2) = 1$  gelten. Also:  $I(M') = 1$ . Damit existiert ein Pfad in  $T_{i+1}$  der unter  $I$  wahr ist.
    - b)  $\beta$ -Regel: Dann entstehen zwei neue Äste,  $M' = \langle M, \beta_1 \rangle$  und  $M'' = \langle M, \beta_2 \rangle$ . Da  $I(A) = 1$ , muss  $I(\beta_1) = 1$  oder  $I(\beta_2) = 1$  gelten. Im ersten Fall ist  $M'$  der gesuchte Pfad, im zweiten  $M''$ .

In beiden Fällen existiert der gesuchte Ast.

In beiden Fällen existiert der gesuchte Ast.

- ▶ Also:  $T_{i+1}$  hat einen Ast, der zu 1 ausgewertet wird.
- ▶ Per Induktion also: Jedes Tableau in der Herleitung hat einen Ast, der unter  $I$  wahr ist.

## Satz: Geschlossene Tableaux

- ▶ Sei  $F$  eine Formel und  $T$  ein geschlossenes Tableau für  $F$ . Dann ist  $F$  unerfüllbar.
- ▶  $T$  heißt **Tableaux-Beweis** für (die Unerfüllbarkeit von)  $F$ .

Beweis: Per Widerspruch

- ▶ Annahme:  $F$  ist erfüllbar. Sei  $I$  Interpretation mit  $I(F) = 1$
- ▶ Nach vorstehendem Lemma hat  $T$  dann einen Ast  $M$ , der unter  $I$  wahr ist.
- ▶ Aber: Alle Äste im Tableau sind geschlossen, und ein geschlossener Ast enthält per Definition zwei komplementäre Formeln.
- ▶ Nur eine von beiden kann unter  $I$  wahr sein, also kann  $M$  unter  $I$  nicht wahr sein.
- ▶ Widerspruch! Also: Die Annahme ist falsch. Also ist  $F$  unerfüllbar.

q.e.d.

## **Satz (Tableaux-Modelle):**

Sei  $F$  eine aussagenlogische Formel,  $T$  ein vollständiges Tableau, und  $M$  ein offener Ast in  $T$ . Dann beschreiben die Literale in  $M$  eine erfüllende Interpretation von  $F$ , d.h. jede Interpretation, die die Literale in  $T$  wahr macht, ist auch ein Modell von  $F$ .

- ▶ Wenn  $F$  erfüllbar ist, so gibt es nach dem obigen Lemma einen solche Ast
- ▶ Da  $T$  vollständig ist und  $M$  nicht geschlossen ist, ist  $M$  vollständig, d.h. alle Knoten sind expandiert - Endergebnis sind Literale
- ▶ Es ist nicht notwendigerweise für jedes Atom ein Literal auf einem offenen Ast. Werte für nicht vorkommende Atome können frei gewählt werden!

## Übung: Modellextraktion

- ▶ Betrachten Sie folgende Formel  $F: (a \vee b) \wedge (a \vee \neg b) \wedge (a \rightarrow c)$ 
  - ▶ Generieren Sie ein vollständiges Tableau für  $F$ .
  - ▶ Extrahieren Sie aus dem Tableau die Modelle von  $F$ .

## Satz (Vollständigkeit und Korrektheit des Tableaux-Kalküls)

Sei  $F \in \text{For}_{0\Sigma}$  eine aussagenlogische Formel und  $T$  ein vollständiges Tableau für  $F$ . Dann gilt:

- ▶ Wenn  $T$  offen ist, so ist  $F$  erfüllbar
  - ▶ Wenn  $T$  geschlossen ist, so ist  $F$  unerfüllbar
- 
- ▶ Zusammenfassung der letzten beiden Sätze
  - ▶ Da die Teilformeln bei jeder Zerlegung kleiner werden, wird jede Ableitung nach endlich vielen Schritten ein vollständiges Tableau erzeugen!

# Umsetzung des Tableaux-Kalküls

- ▶ Markiere expandierte Knoten
- ▶ Heuristik: Bevorzuge  $\alpha$ -Regeln
- ▶ Strategien:
  - ▶ Depth-first (Kann schneller Modelle finden)
  - ▶ Breadth-first (vollständig auch für PL1 mit unendlichen Tableaux)
- ▶ Im Rechner:
  - ▶ Tableau: Rekursive Baum-Struktur
  - ▶ Naiv: Durchsuche alle Knoten nach nächstem nicht expandierten Knoten
  - ▶ Besser: Z.B. Stack/FIFO mit nicht expandierten Knoten
    - ▶ Stack: Depth-First
    - ▶ FIFO: Breadth-First
  - ▶ Teste bei Expansion, ob Ast geschlossen wurde

# Übung: Mehr Tableaux

- ▶ Welche der folgenden Formeln sind unerfüllbar? Geben Sie für erfüllbare Formeln ein Modell an. Verwenden Sie den Tableau-Kalkül!
  - ▶  $\neg(q \rightarrow (p \rightarrow q))$
  - ▶  $\neg(((p \rightarrow q) \wedge (q \rightarrow r)) \rightarrow (p \rightarrow r))$
  - ▶  $\neg(((p \rightarrow q) \wedge (q \rightarrow r)) \rightarrow (p \rightarrow (q \wedge r)))$
  - ▶  $((p \wedge (q \rightarrow p)) \rightarrow p)$
  - ▶  $\neg((p \wedge (q \rightarrow p)) \rightarrow p)$

# Übung: Tableaux für Inspektor Craig

## Craig 1:

1.  $(A \wedge \neg B) \rightarrow C$
2.  $C \rightarrow (A \vee B)$
3.  $A \rightarrow \neg C$
4.  $A \vee B \vee C$

## Craig 2:

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

- ▶ Betrachten Sie die Formeln von Craig 1 („Wer ist mindestens schuldig“) und Craig 2 („Raubüberfall“) als Konjunktionen (implizit „verundet“).
- ▶ Wenn Ihr Nachname mit einer der Buchstaben von A bis M beginnt, bearbeiten Sie *Craig 1*, ansonsten *Craig 2*.
- ▶ Generieren Sie ein vollständiges Tableaux für die jeweilige Formel. Sie können dabei mit einem Tableau anfangen, bei dem die einzelnen Formeln bereits an einem gemeinsamen Ast stehen.

- ▶ Viele praktische Kalküle zeigen die **Unerfüllbarkeit** einer Formel(-Menge):
  - ▶ Zu zeigen: Eine Hypothese  $H$  folgt aus einer Menge von Formeln  $KB = \{F_1, F_2, \dots, F_n\}$ :
- ▶ Deduktionstheorem: Das gilt genau dann, wenn die entsprechende Implikation allgemeingültig ist, also:

$$KB \models H$$

$$\models (F_1 \wedge F_2 \wedge \dots \wedge F_n) \rightarrow H$$

- ▶ Dualitätsprinzip: Das gilt genau dann, wenn die Negation dieser Formel unerfüllbar ist:

$$\neg((F_1 \wedge F_2 \wedge \dots \wedge F_n) \rightarrow H) \text{ unerfüllbar}$$

- ▶ Fakt: Äquivalent können wir zeigen:

$$F_1 \wedge F_2 \wedge \dots \wedge F_n \wedge \neg H \text{ unerfüllbar}$$

# Widerspruchskalküle

- ▶ Viele praktische Kalküle zeigen die **Unerfüllbarkeit** einer Formel(-Menge):
  - ▶ Zu zeigen: Eine Hypothese  $H$  folgt aus einer Menge von Formeln  $KB = \{F_1, F_2, \dots, F_n\}$ :

$$KB \models H$$

- ▶ Deduktionstheorem: Das gilt genau dann, wenn die entsprechende Implikation allgemeingültig ist, also:

$$\models (F_1 \wedge F_2 \wedge \dots \wedge F_n) \rightarrow H$$

- ▶ Dualitätsprinzip: Das gilt genau dann, wenn die Negation dieser Formel unerfüllbar ist:

$$\neg((F_1 \wedge F_2 \wedge \dots \wedge F_n) \rightarrow H) \text{ unerfüllbar}$$

- ▶ Fakt: Äquivalent können wir zeigen:

$$F_1 \wedge F_2 \wedge \dots \wedge F_n \wedge \neg H \text{ unerfüllbar}$$

Im Tableauxkalkül: Herleitung eines geschlossenen Tableaus

## Definition (Logische Äquivalenz):

Zwei Formeln  $F, G \in For_{0\Sigma}$  heißen **äquivalent**, falls  $F \models G$  und  $G \models F$ .

Schreibweise:  $F \equiv G$

- ▶ In dem Fall gilt  $I(F) = I(G)$  für alle Interpretationen  $I$
- ▶ Analog zum Deduktionstheorem gilt:  $F \equiv G$  gdw.  $\models F \leftrightarrow G$
- ▶ Viele Anwendungsprobleme lassen sich als Äquivalenzen formulieren:
  - ▶ Äquivalenz von zwei Spezifikationen
  - ▶ Äquivalenz von zwei Schaltungen
  - ▶ Äquivalenz von funktionaler Beschreibung und Implementierung

# Logische Äquivalenz

## Definition (Logische Äquivalenz):

Zwei Formeln  $F, G \in \text{For}_{0\Sigma}$  heißen **äquivalent**, falls  $F \models G$  und  $G \models F$ .

Schreibweise:  $F \equiv G$

- ▶ In dem Fall gilt  $I(F) = I(G)$  für alle Interpretationen  $I$
- ▶ Analog zum Deduktionstheorem gilt:  $F \equiv G$  gdw.  $\models F \leftrightarrow G$
- ▶ Viele Anwendungsprobleme lassen sich als Äquivalenzen formulieren:
  - ▶ Äquivalenz von zwei Spezifikationen
  - ▶ Äquivalenz von zwei Schaltungen
  - ▶ Äquivalenz von funktionaler Beschreibung und Implementierung

Wir können *Teilformeln* durch äquivalente Teilformeln ersetzen, ohne den Wert der Formel unter einer beliebigen Interpretation zu verändern!

- ▶ Finden Sie äquivalente Formelpaare. . .
  - ▶ ... mit 3 gemeinsamen Aussagenvariablen
  - ▶ ... mit 2 gemeinsamen Aussagenvariablen, wobei beide Formeln keinen Operator gemeinsam haben
  - ▶ ... mit Variablen, aber ohne gemeinsame Aussagenvariablen

Zurück B

**Definition (Basis der Aussagenlogik):**

Eine Menge von Operatoren  $O$  heißt eine **Basis** der Aussagenlogik, falls gilt: Zu jeder Formel  $F$  gibt es eine Formel  $G$  mit  $F \equiv G$ , und  $G$  verwendet nur Operatoren aus  $O$ .

- ▶ Das Konzept ermöglicht es, viele Aussagen zu Erfüllbarkeit, Folgerungen, und Äquivalenz auf einfachere Formelklassen zu beschränken.

## Definition (Basis der Aussagenlogik):

Eine Menge von Operatoren  $O$  heißt eine **Basis** der Aussagenlogik, falls gilt: Zu jeder Formel  $F$  gibt es eine Formel  $G$  mit  $F \equiv G$ , und  $G$  verwendet nur Operatoren aus  $O$ .

- ▶ Das Konzept ermöglicht es, viele Aussagen zu Erfüllbarkeit, Folgerungen, und Äquivalenz auf einfachere Formelklassen zu beschränken.

## Satz:

- ▶  $\{\wedge, \vee, \neg\}$  ist eine Basis
- ▶  $\{\rightarrow, \neg\}$  ist eine Basis
- ▶  $\{\wedge, \neg\}$  ist eine Basis

# Beweisprinzip: Strukturelle Induktion

- ▶ Die **strukturelle Induktion** oder **Induktion über den Aufbau** kann verwendet werden, um Eigenschaften von rekursiv definierten Objekten zu zeigen.
- ▶ Idee:
  - ▶ Zeige die Eigenschaft für die elementaren Objekte.
  - ▶ Zeige die Eigenschaft für die zusammengesetzten Objekte unter der Annahme, dass die Teile bereits die Eigenschaft haben

# Beweisprinzip: Strukturelle Induktion

- ▶ Die **strukturelle Induktion** oder **Induktion über den Aufbau** kann verwendet werden, um Eigenschaften von rekursiv definierten Objekten zu zeigen.
- ▶ Idee:
  - ▶ Zeige die Eigenschaft für die elementaren Objekte.
  - ▶ Zeige die Eigenschaft für die zusammengesetzten Objekte unter der Annahme, dass die Teile bereits die Eigenschaft haben
- ▶ Konkret für aussagenlogische Formeln:
  - ▶ Basisfälle sind die Aussagenlogischen Variablen oder  $\top$ ,  $\perp$ .
  - ▶ Zusammengesetzte Formeln haben die Form  $\neg A$ ,  $A \vee B$ ,  $A \wedge B$ ..., und wir können annehmen, dass  $A$  und  $B$  schon die gewünschte Eigenschaft haben.

# Übung: Basis der Aussagenlogik

- ▶ Erinnerung: Eine Menge von Operatoren  $O$  heißt eine **Basis** der Aussagenlogik, falls gilt: Zu jeder Formel  $F$  gibt es eine Formel  $G$  mit  $F \equiv G$ , und  $G$  verwendet nur Operatoren aus  $O$ .
- ▶ Zeigen Sie (wahlweise):
  - ▶  $\{\wedge, \vee, \neg\}$  ist eine Basis
  - ▶  $\{\rightarrow, \neg\}$  ist eine Basis
  - ▶  $\{\wedge, \neg\}$  ist eine Basis

Zurück C

## Definition (Teilformel):

Seien  $A, B \in For_{0\Sigma}$ .  $B$  heißt **Unterformel** oder **Teilformel** von  $A$ , falls:

1.  $A = B$  oder
2.  $A = \neg C$  und  $B$  ist Teilformel von  $C$  oder
3.  $A = (C \otimes D)$ ,  $\otimes \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$ , und  $B$  ist Teilformel von  $C$  oder  $B$  ist Teilformel von  $D$

Schreibweise: Menge der Teilformeln von  $A$ :  $TF(A)$

## Definition (Teilformel):

Seien  $A, B \in For_{0\Sigma}$ .  $B$  heißt **Unterformel** oder **Teilformel** von  $A$ , falls:

1.  $A = B$  oder
2.  $A = \neg C$  und  $B$  ist Teilformel von  $C$  oder
3.  $A = (C \otimes D)$ ,  $\otimes \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$ , und  $B$  ist Teilformel von  $C$  oder  $B$  ist Teilformel von  $D$

Schreibweise: Menge der Teilformeln von  $A$ :  $TF(A)$

► Beispiel:

►  $TF((a \vee (\neg b \leftrightarrow c))) =$

## Definition (Teilformel):

Seien  $A, B \in For_{0\Sigma}$ .  $B$  heißt **Unterformel** oder **Teilformel** von  $A$ , falls:

1.  $A = B$  oder
2.  $A = \neg C$  und  $B$  ist Teilformel von  $C$  oder
3.  $A = (C \otimes D)$ ,  $\otimes \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$ , und  $B$  ist Teilformel von  $C$  oder  $B$  ist Teilformel von  $D$

Schreibweise: Menge der Teilformeln von  $A$ :  $TF(A)$

### ► Beispiel:

$$\text{► } TF((a \vee (\neg b \leftrightarrow c))) = \{(a \vee (\neg b \leftrightarrow c)), a, (\neg b \leftrightarrow c), \neg b, b, c\}$$

## Definition (Teilformel):

Seien  $A, B \in For_{0\Sigma}$ .  $B$  heißt **Unterformel** oder **Teilformel** von  $A$ , falls:

1.  $A = B$  oder
2.  $A = \neg C$  und  $B$  ist Teilformel von  $C$  oder
3.  $A = (C \otimes D)$ ,  $\otimes \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$ , und  $B$  ist Teilformel von  $C$  oder  $B$  ist Teilformel von  $D$

Schreibweise: Menge der Teilformeln von  $A$ :  $TF(A)$

### ► Beispiel:

- $TF((a \vee (\neg b \leftrightarrow c))) = \{(a \vee (\neg b \leftrightarrow c)), a, (\neg b \leftrightarrow c), \neg b, b, c\}$
- $TF(\neg\neg\neg a) =$

## Definition (Teilformel):

Seien  $A, B \in For_{0\Sigma}$ .  $B$  heißt **Unterformel** oder **Teilformel** von  $A$ , falls:

1.  $A = B$  oder
2.  $A = \neg C$  und  $B$  ist Teilformel von  $C$  oder
3.  $A = (C \otimes D)$ ,  $\otimes \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$ , und  $B$  ist Teilformel von  $C$  oder  $B$  ist Teilformel von  $D$

Schreibweise: Menge der Teilformeln von  $A$ :  $TF(A)$

### ► Beispiel:

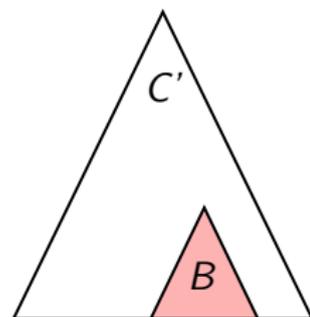
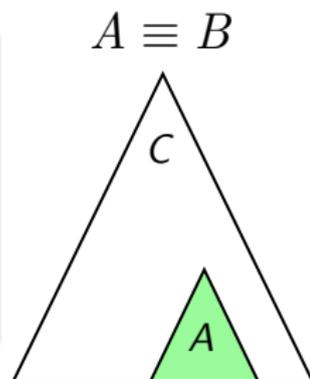
- ▶  $TF((a \vee (\neg b \leftrightarrow c))) = \{(a \vee (\neg b \leftrightarrow c)), a, (\neg b \leftrightarrow c), \neg b, b, c\}$
- ▶  $TF(\neg\neg\neg a) = \{\neg\neg\neg a, \neg\neg a, \neg a, a\}$

# Substitutionstheorem

**Satz: (Substitutionstheorem)** Sei  $A \equiv B$  und  $C'$  das Ergebnis der Ersetzung einer Unterformel  $A$  in  $C$  durch  $B$ . Dann gilt:

$$C \equiv C'$$

- ▶ Beispiel:  $p \vee q \equiv q \vee p$   
impliziert  
 $(r \wedge (p \vee q)) \rightarrow s \equiv (r \wedge (q \vee p)) \rightarrow s$



# Substitutionstheorem (Beweis 1)

## Beweis:

**Behauptung:** Sei  $A \equiv B$  und  $C'$  das Ergebnis der Ersetzung einer Unterformel  $A$  in  $C$  durch  $B$ . Dann gilt  $C \equiv C'$ .

**Zu zeigen:**  $I(C) = I(C')$  für alle Interpretationen  $I$ .

**Beweis:** Per struktureller Induktion über  $C$

**IA:**  $C$  ist elementare Formel, also  $C \equiv \top$  oder  $C \equiv \perp$  oder  $C \in \Sigma$ .  
Dann gilt notwendigerweise  $A=C$  (da  $C$  atomar ist). Also gilt:  
 $C = A \equiv B = C'$ , also  $I(C) = I(C')$  für alle  $I$ .

# Substitutionstheorem (Beweis 2)

**IV:** Behauptung gelte für alle echten Unterformeln von  $C$

**IS:** Sei  $C$  eine zusammengesetzte Formel. Dann gilt:  $C$  ist von einer der folgenden Formen:  $C = \neg D$  oder

$C = (D \otimes E)$ ,  $\otimes \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$  und  $A$  ist eine Unterformel von  $D$  oder eine Unterformel von  $E$ . Sei  $C \neq A$  (sonst: Wie IA).

- ▶ Fall 1:  $C = \neg D$ . Dann ist  $A$  Unterformel von  $D$ . Sei  $D'$  die Formel, die entsteht, wenn man  $A$  durch  $B$  ersetzt. Laut IV gilt:  $\text{val}_I(D) = \text{val}_I(D')$  für alle  $I$ . Wir zeigen: Dann gilt auch  $\text{val}_I(C) = \text{val}_I(C')$ .
  - ▶ Fall 1a):  $\text{val}_I(C) = 0 \implies \text{val}_I(D) = 1 = \text{val}_I(D') \implies \text{val}_I(C') = 0$   
per Definition  $\text{val}_I$
  - ▶ Fall 1b):  $\text{val}_I(C) = 1 \implies \text{val}_I(D) = 0 = \text{val}_I(D') \implies \text{val}_I(C') = 1$   
per Definition  $\text{val}_I$

# Substitutionstheorem (Beweis 3)

## IS: (Fortgesetzt)

- ▶ Fall 2:  $C = D \vee E$ . Sei oBdA  $A$  Unterformel von  $D$ 
  - ▶ Fall 2a(i):  $\text{val}_I(D) = 1 \implies \text{val}_I(D') = 1$  per IV  
 $\implies \text{val}_I(D' \vee E) = 1 \implies \text{val}_I(C') = 1$
  - ▶ Fall 2a(ii):  $\text{val}_I(E) = 1 \implies \text{val}_I(D' \vee E) = 1 \implies \text{val}_I(C') = 1$
  - ▶ Fall 2b):  $\text{val}_I(C) = 0 \implies \text{val}_I(D) = 0$  und  
 $\text{val}_I(E) = 0 \implies \text{val}_I(D') = 0$  (per IV)  
 $\implies \text{val}_I(D' \vee E) = 0 \implies \text{val}_I(C') = 0$
- ▶ Fall 3- n:  $C = (D \otimes E)$ ,  $\otimes \in \{\wedge, \rightarrow, \leftrightarrow\}$ : Analog (aber mühsam).

q.e.d.

# Warum ersetzen?

- ▶ Das Substitutionstheorem erlaubt uns, eine Menge von Äquivalenzen zu formulieren, mit denen wir eine Formel umformen können
- ▶ Insbesondere:
  - ▶ Wir können jede allgemeingültige Formel in  $\top$  umformen
  - ▶ Wir können jede unerfüllbare Formel in  $\perp$  umformen
  - ▶ Wir können Formeln systematisch in Normalformen bringen und Kalküle entwickeln, die nur auf diesen Normalformen arbeiten.

# Wichtige Äquivalenzen (1)

►  $(A \wedge B) \equiv (B \wedge A)$

Kommutativität von  $\wedge$

# Wichtige Äquivalenzen (1)

▶  $(A \wedge B) \equiv (B \wedge A)$

▶  $(A \vee B) \equiv (B \vee A)$

Kommutativität von  $\wedge$

Kommutativität von  $\vee$

# Wichtige Äquivalenzen (1)

- ▶  $(A \wedge B) \equiv (B \wedge A)$
- ▶  $(A \vee B) \equiv (B \vee A)$
- ▶  $((A \wedge B) \wedge C) \equiv (A \wedge (B \wedge C))$

Kommutativität von  $\wedge$

Kommutativität von  $\vee$

Assoziativität von  $\wedge$

# Wichtige Äquivalenzen (1)

- ▶  $(A \wedge B) \equiv (B \wedge A)$
- ▶  $(A \vee B) \equiv (B \vee A)$
- ▶  $((A \wedge B) \wedge C) \equiv (A \wedge (B \wedge C))$
- ▶  $((A \vee B) \vee C) \equiv (A \vee (B \vee C))$

Kommutativität von  $\wedge$

Kommutativität von  $\vee$

Assoziativität von  $\wedge$

Assoziativität von  $\vee$

# Wichtige Äquivalenzen (1)

- ▶  $(A \wedge B) \equiv (B \wedge A)$
- ▶  $(A \vee B) \equiv (B \vee A)$
- ▶  $((A \wedge B) \wedge C) \equiv (A \wedge (B \wedge C))$
- ▶  $((A \vee B) \vee C) \equiv (A \vee (B \vee C))$
- ▶  $(A \wedge A) \equiv A$

Kommutativität von  $\wedge$

Kommutativität von  $\vee$

Assoziativität von  $\wedge$

Assoziativität von  $\vee$

Idempotenz für  $\wedge$

# Wichtige Äquivalenzen (1)

- ▶  $(A \wedge B) \equiv (B \wedge A)$
- ▶  $(A \vee B) \equiv (B \vee A)$
- ▶  $((A \wedge B) \wedge C) \equiv (A \wedge (B \wedge C))$
- ▶  $((A \vee B) \vee C) \equiv (A \vee (B \vee C))$
- ▶  $(A \wedge A) \equiv A$
- ▶  $(A \vee A) \equiv A$

Kommutativität von  $\wedge$

Kommutativität von  $\vee$

Assoziativität von  $\wedge$

Assoziativität von  $\vee$

Idempotenz für  $\wedge$

Idempotenz für  $\vee$

# Wichtige Äquivalenzen (1)

- ▶  $(A \wedge B) \equiv (B \wedge A)$
- ▶  $(A \vee B) \equiv (B \vee A)$
- ▶  $((A \wedge B) \wedge C) \equiv (A \wedge (B \wedge C))$
- ▶  $((A \vee B) \vee C) \equiv (A \vee (B \vee C))$
- ▶  $(A \wedge A) \equiv A$
- ▶  $(A \vee A) \equiv A$
- ▶  $\neg\neg A \equiv A$

Kommutativität von  $\wedge$

Kommutativität von  $\vee$

Assoziativität von  $\wedge$

Assoziativität von  $\vee$

Idempotenz für  $\wedge$

Idempotenz für  $\vee$

Doppelnegation

# Wichtige Äquivalenzen (1)

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

Kommutativität von  $\wedge$

Kommutativität von  $\vee$

Assoziativität von  $\wedge$

Assoziativität von  $\vee$

Idempotenz für  $\wedge$

Idempotenz für  $\vee$

Doppelnegation

Kontraposition

## Wichtige Äquivalenzen (2)

▶  $(A \rightarrow B) \equiv (\neg A \vee B)$

Elimination Implikation

▶  $(A \leftrightarrow B) \equiv ((A \rightarrow B) \wedge (B \rightarrow A))$

Elimination Äquivalenz

▶  $\neg(A \wedge B) \equiv (\neg A \vee \neg B)$

De-Morgans Regeln

▶  $\neg(A \vee B) \equiv (\neg A \wedge \neg B)$

De-Morgans Regeln

▶  $(A \wedge (B \vee C)) \equiv ((A \wedge B) \vee (A \wedge C))$  Distributivität von  $\wedge$  über  $\vee$

▶  $(A \vee (B \wedge C)) \equiv ((A \vee B) \wedge (A \vee C))$  Distributivität von  $\vee$  über  $\wedge$

# Wichtige Äquivalenzen (zusammengefasst)

$$\begin{aligned}(A \wedge B) &\equiv (B \wedge A) \\(A \vee B) &\equiv (B \vee A) \\((A \wedge B) \wedge C) &\equiv (A \wedge (B \wedge C)) \\((A \vee B) \vee C) &\equiv (A \vee (B \vee C)) \\(A \wedge A) &\equiv A \\(A \vee A) &\equiv A \\ \neg\neg A &\equiv A \\(A \rightarrow B) &\equiv (\neg B \rightarrow \neg A) \\(A \rightarrow B) &\equiv (\neg A \vee B) \\(A \leftrightarrow B) &\equiv ((A \rightarrow B) \wedge (B \rightarrow A)) \\ \neg(A \wedge B) &\equiv (\neg A \vee \neg B) \\ \neg(A \vee B) &\equiv (\neg A \wedge \neg B) \\(A \wedge (B \vee C)) &\equiv ((A \wedge B) \vee (A \wedge C)) \\(A \vee (B \wedge C)) &\equiv ((A \vee B) \wedge (A \vee C))\end{aligned}$$

Kommutativität von  $\wedge$   
Kommutativität von  $\vee$   
Assoziativität von  $\wedge$   
Assoziativität von  $\vee$   
Idempotenz für  $\wedge$   
Idempotenz für  $\vee$   
Doppelnegation  
Kontraposition  
Elimination Implikation  
Elimination Äquivalenz  
De-Morgans Regeln  
De-Morgans Regeln  
Distributivität von  $\wedge$  über  
Distributivität von  $\vee$  über

# Wichtige Äquivalenzen mit $\top$ und $\perp$

- ▶  $(A \wedge \neg A) \equiv \perp$
- ▶  $(A \vee \neg A) \equiv \top$
- ▶  $(A \wedge \top) \equiv A$
- ▶  $(A \wedge \perp) \equiv \perp$
- ▶  $(A \vee \top) \equiv \top$
- ▶  $(A \vee \perp) \equiv A$
- ▶  $(\neg \top) \equiv \perp$
- ▶  $(\neg \perp) \equiv \top$

Tertium non datur

## **Definition (Äquivalenzumformung)**

Eine **Äquivalenzumformung** ist das Ersetzen einer Teilformel durch eine äquivalente Teilformel entsprechend den vorausgehenden Tabellen.

## Definition (Äquivalenzumformung)

Eine **Äquivalenzumformung** ist das Ersetzen einer Teilformel durch eine äquivalente Teilformel entsprechend den vorausgehenden Tabellen.

## Definition (Kalkül der logischen Umformungen):

- ▶ Wir schreiben  $A \vdash_{LU} B$  wenn  $A$  mit Äquivalenzumformungen in  $B$  umgeformt werden kann.
- ▶ Wir schreiben  $\vdash_{LU} B$  falls  $\top \vdash_{LU} B$

# Einfaches Beispiel

► Zu zeigen:  $\vdash_{LU} (p \rightarrow (p \vee q))$

► Herleitung:

$\top \vdash_{LU} q \vee \top$

$\vdash_{LU} \top \vee q$       Kommutativität

$\vdash_{LU} (p \vee \neg p) \vee q$

$\vdash_{LU} (\neg p \vee p) \vee q$       Kommutativität

$\vdash_{LU} \neg p \vee (p \vee q)$       Assoziativität

$\vdash_{LU} p \rightarrow (p \vee q)$       Elimination Implikation

Zeigen Sie:

- ▶  $\vdash_{LU} (A \rightarrow (B \rightarrow A))$
- ▶  $\vdash_{LU} (A \rightarrow B) \vee (B \rightarrow A)$

## Satz (Korrektheit und Vollständigkeit):

- ▶ Der Kalkül der logischen Umformungen ist **korrekt**:  
 $\vdash_{LU} B$  impliziert  $\models B$
- ▶ Der Kalkül der logischen Umformungen ist **vollständig**:  
 $\models B$  impliziert  $\vdash_{LU} B$

## Satz (Korrektheit und Vollständigkeit):

- ▶ Der Kalkül der logischen Umformungen ist **korrekt**:

$\vdash_{LU} B$  impliziert  $\models B$

- ▶ Der Kalkül der logischen Umformungen ist **vollständig**:

$\models B$  impliziert  $\vdash_{LU} B$

- ▶ Anmerkung

- ▶ Der Kalkül der logischen Umformungen ist nicht vollständig für Folgerungen:  $A \models B$  impliziert nicht  $A \vdash_{UL} B$

- ▶ Beispiel:  $a \wedge b \models a$ , aber  $a \wedge b \not\vdash_{UL} a$

## Normalformen

- ▶ Aussagenlogische Formeln: Kompliziert, beliebig verschachtelt
  - ▶ Optimiert für Ausdruckskraft
  - ▶ Erlaubt kompakte Spezifikationen
- ▶ Algorithmen und Kalküle werden einfacher für einfachere Sprachen
  - ▶ Weniger Fälle zu betrachten
  - ▶ Regulärer Code
  - ▶ Höhere Effizienz

## Normalformen

- ▶ Aussagenlogische Formeln: Kompliziert, beliebig verschachtelt
  - ▶ Optimierte für Ausdruckskraft
  - ▶ Erlaubt kompakte Spezifikationen
- ▶ Algorithmen und Kalküle werden einfacher für einfachere Sprachen
  - ▶ Weniger Fälle zu betrachten
  - ▶ Regulärer Code
  - ▶ Höhere Effizienz

**Idee: Konvertierung in einfachere Teilsprache**

**Definition (Negations-Normalform (NNF)):** Eine Formel  $F \in For_{0\Sigma}$  ist in [Negations-Normalform](#), wenn folgende Bedingungen gelten:

- ▶ Als Operatoren kommen nur  $\wedge, \vee, \neg$  vor.
- ▶  $\neg$  kommt nur direkt vor Atomen vor.

**Definition (Negations-Normalform (NNF)):** Eine Formel  $F \in For_{0\Sigma}$  ist in **Negations-Normalform**, wenn folgende Bedingungen gelten:

- ▶ Als Operatoren kommen nur  $\wedge, \vee, \neg$  vor.
- ▶  $\neg$  kommt nur direkt vor Atomen vor.

▶ Beispiele:

- ▶  $F = (A \wedge \neg B) \vee C$  ist in NNF

**Definition (Negations-Normalform (NNF)):** Eine Formel  $F \in For_{0\Sigma}$  ist in **Negations-Normalform**, wenn folgende Bedingungen gelten:

- ▶ Als Operatoren kommen nur  $\wedge, \vee, \neg$  vor.
- ▶  $\neg$  kommt nur direkt vor Atomen vor.

▶ Beispiele:

- ▶  $F = (A \wedge \neg B) \vee C$  ist in NNF  
 $F$  ist eine NNF von  $(A \rightarrow B) \rightarrow C$

**Definition (Negations-Normalform (NNF)):** Eine Formel  $F \in For_{0\Sigma}$  ist in **Negations-Normalform**, wenn folgende Bedingungen gelten:

- ▶ Als Operatoren kommen nur  $\wedge, \vee, \neg$  vor.
- ▶  $\neg$  kommt nur direkt vor Atomen vor.

▶ Beispiele:

- ▶  $F = (A \wedge \neg B) \vee C$  ist in NNF
- ▶  $F$  ist eine NNF von  $(A \rightarrow B) \rightarrow C$
- ▶  $(A \vee \neg(B \wedge C))$  ist nicht in NNF
- ▶  $(\neg A \vee \neg\neg B)$  ist nicht in NNF

**Definition (Negations-Normalform (NNF)):** Eine Formel  $F \in For_{0\Sigma}$  ist in **Negations-Normalform**, wenn folgende Bedingungen gelten:

- ▶ Als Operatoren kommen nur  $\wedge, \vee, \neg$  vor.
- ▶  $\neg$  kommt nur direkt vor Atomen vor.

▶ Beispiele:

- ▶  $F = (A \wedge \neg B) \vee C$  ist in NNF  
 $F$  ist eine NNF von  $(A \rightarrow B) \rightarrow C$
- ▶  $(A \vee \neg(B \wedge C))$  ist nicht in NNF
- ▶  $(\neg A \vee \neg\neg B)$  ist nicht in NNF  
... aber  $(\neg A \vee B)$  ist in NNF

## Drei Schritte

1. Elimination von  $\leftrightarrow$

$$\text{Verwende } A \leftrightarrow B \equiv (A \rightarrow B) \wedge (B \rightarrow A)$$

2. Elimination von  $\rightarrow$

$$\text{Verwende } A \rightarrow B \equiv \neg A \vee B$$

3. „Nach innen schieben“ von  $\neg$ , Elimination  $\top, \perp$

$$\text{Verwende de-Morgans Regeln und } \neg\neg A \equiv A$$

Verwende  $\top/\perp$ -Regeln

**Ergebnis: Formel in NNF oder  $\top/\perp$**

Bestimmen Sie je eine NNF zu den folgenden Formeln:

- ▶  $\neg(a \rightarrow \top) \vee a$
- ▶  $\neg(a \vee b) \leftrightarrow (b \wedge (a \rightarrow \neg c))$
- ▶  $(a \rightarrow b) \rightarrow ((b \rightarrow c) \rightarrow (a \rightarrow c))$

**Definition (Literal):** Ein **Literal** ist ein Atom (aussagenlogische Variable) oder die Negation eines Atoms.

**Definition (Literal):** Ein **Literal** ist ein Atom (aussagenlogische Variable) oder die Negation eines Atoms.

**Definition (Klausel):** Eine **Klausel** ist eine Disjunktion von Literalen.

- ▶ Wir erlauben hier mehrstellige Disjunktionen:  $(A \vee \neg B \vee C)$ 
  - ▶ Interpretation wie bisher (links-assoziativ)
  - ▶ Aber wir betrachten die Struktur nun als flach
  - ▶ Eine Klausel wird wahr, wenn eines der Literale wahr wird!
- ▶ Spezialfälle:
  - ▶ Einstellige Disjunktionen, z.B.:  $A$  oder  $\neg B$
  - ▶ Die nullstellige Disjunktion (leere Klausel):  $\perp$  oder  $\square$ 
    - ▶ Die leere Klausel ist unerfüllbar

**Definition (Konjunktive Normalform (KNF)):** Eine Formel in **konjunktiver Normalform** ist eine Konjunktion von Klauseln.

Die Konjunktion kann mehrstellig, einstellig oder nullstellig sein.

**Definition (Konjunktive Normalform (KNF)):** Eine Formel in **konjunktiver Normalform** ist eine Konjunktion von Klauseln.

Die Konjunktion kann mehrstellig, einstellig oder nullstellig sein.

► Beispiele:

- $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$
- $A \vee B$
- $A \wedge (B \vee C)$
- $A \wedge B$
- $\top$  (als Schreibweise für die leere Konjunktion)

## Zwei Schritte

1. Transformation in NNF

2. „Nach innen schieben“ von  $\vee$   
Verwende Distributivität von  $\vee$  über  $\wedge$

## Vier Schritte

### 1. Transformation in NNF

#### 1.1 Elimination von $\leftrightarrow$

Verwende  $A \leftrightarrow B \equiv (A \rightarrow B) \wedge (B \rightarrow A)$

#### 1.2 Elimination von $\rightarrow$

Verwende  $A \rightarrow B \equiv \neg A \vee B$

#### 1.3 „Nach innen schieben“ von $\neg$ , elimination $\top, \perp$

Verwende de-Morgans Regeln und  $\neg\neg A \equiv A$

Verwende  $\top/\perp$ -Regeln

### 2. „Nach innen schieben“ von $\vee$

Verwende Distributivität von  $\vee$  über  $\wedge$

# Umformung in KNF: Beispiel

- ▶ Ausgangsformel

$$p \leftrightarrow (q \vee r)$$

# Umformung in KNF: Beispiel

- ▶ Ausgangsformel

$$p \leftrightarrow (q \vee r)$$

- ▶ 1. Elimination von  $\leftrightarrow$

$$(p \rightarrow (q \vee r)) \wedge ((q \vee r) \rightarrow p)$$

# Umformung in KNF: Beispiel

- ▶ Ausgangsformel

$$p \leftrightarrow (q \vee r)$$

- ▶ 1. Elimination von  $\leftrightarrow$

$$(p \rightarrow (q \vee r)) \wedge ((q \vee r) \rightarrow p)$$

- ▶ 2. Elimination von  $\rightarrow$

$$(\neg p \vee q \vee r) \wedge (\neg(q \vee r) \vee p)$$

# Umformung in KNF: Beispiel

- ▶ Ausgangsformel

$$p \leftrightarrow (q \vee r)$$

- ▶ 1. Elimination von  $\leftrightarrow$

$$(p \rightarrow (q \vee r)) \wedge ((q \vee r) \rightarrow p)$$

- ▶ 2. Elimination von  $\rightarrow$

$$(\neg p \vee q \vee r) \wedge (\neg(q \vee r) \vee p)$$

- ▶ 3. Nach innen schieben von  $\neg$

$$(\neg p \vee q \vee r) \wedge ((\neg q \wedge \neg r) \vee p)$$

# Umformung in KNF: Beispiel

- ▶ Ausgangsformel

$$p \leftrightarrow (q \vee r)$$

- ▶ 1. Elimination von  $\leftrightarrow$

$$(p \rightarrow (q \vee r)) \wedge ((q \vee r) \rightarrow p)$$

- ▶ 2. Elimination von  $\rightarrow$

$$(\neg p \vee q \vee r) \wedge (\neg(q \vee r) \vee p)$$

- ▶ 3. Nach innen schieben von  $\neg$

$$(\neg p \vee q \vee r) \wedge ((\neg q \wedge \neg r) \vee p)$$

- ▶ 4. Nach innen schieben von  $\vee$

$$(\neg p \vee q \vee r) \wedge (\neg q \vee p) \wedge (\neg r \vee p)$$

## Übung: KNF Transformation

Transformieren Sie die folgenden Formeln in konjunktive Normalform:

- ▶  $\neg(a \vee b) \leftrightarrow (b \wedge (a \rightarrow \neg c))$
- ▶  $(A \vee B \vee C) \wedge (A \rightarrow ((B \wedge \neg C) \vee (\neg B \wedge C))) \wedge (\neg B \rightarrow \neg C) \wedge \neg(B \wedge C \wedge \neg A) \wedge (\neg C \rightarrow \neg B)$

Tipps:

- ▶ Wenn die Formel auf der obersten Ebene bereits eine Konjunktion („und“-verknüpft) ist, kann man die beiden Teilformeln einzeln transformieren.
- ▶ Wenn eine Formel eine Disjunktion ist, und in den Unterformeln noch  $\wedge$  vorkommt, dann wendet man  $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$  in dieser Richtung an.  $A$  kann dabei durchaus eine komplexe Formel sein.

# Übung: Formalisieren in KNF

Konfiguration von Fahrzeugen:

- ▶ Ein Cabrio hat kein Schiebedach
- ▶ Ein Cabrio hat keinen Subwoofer
- ▶ Doppelauspuff gibt es nur am Cabrio und am Quattro
- ▶ Ein Quattro hat hohen Benzinverbrauch



Ich will keinen hohen Benzinverbrauch. Kann ich ein Auto mit Schiebedach und Doppelauspuff bekommen?

Formalisieren Sie das Problem und generieren Sie eine entsprechende Formel in KNF

▶ Elementare Aussagen:

- ▶  $c$ : Cabrio
- ▶  $s$ : Schiebedach
- ▶  $w$ : Subwoofer
- ▶  $d$ : Doppelauspuff
- ▶  $q$ : Quattro
- ▶  $v$ : Hoher Verbrauch

▶ Wissensbasis:

1.  $c \rightarrow \neg s$
2.  $c \rightarrow \neg w$
3.  $d \rightarrow (q \vee c)$
4.  $q \rightarrow v$

▶ Wunsch:

5.  $\neg v \wedge s \wedge d$

Erfüllbarkeit naiv:  $2^6 = 64$  Interpretationen

▶ Elementare Aussagen:

- ▶  $c$ : Cabrio
- ▶  $s$ : Schiebedach
- ▶  $w$ : Subwoover
- ▶  $d$ : Doppelauspuff
- ▶  $q$ : Quattro
- ▶  $v$ : Hoher Verbrauch

▶ In Klauselform:

1.  $\neg c \vee \neg s$
2.  $\neg c \vee \neg w$
3.  $\neg d \vee q \vee c$
4.  $\neg q \vee v$
5.  $\neg v$
6.  $s$
7.  $d$

Erfüllbarkeit naiv:  $2^6 = 64$  Interpretationen

**Resolution**

**=**

**Verfahren zum Nachweis der  
Unerfüllbarkeit einer  
Klauselmengena**

**Definition (Klauseln als Menge):** Eine **Klausel** ist eine Menge von Literalen

► Beispiele

►  $C_1 = \{\neg c, \neg s\}$

►  $C_3 = \{d, q, c\}$

►  $C_4 = \{\neg q, v\}$

**Definition (Klauseln als Menge):** Eine **Klausel** ist eine Menge von **Literalen**

▶ Beispiele

▶  $C_1 = \{\neg c, \neg s\}$

▶  $C_3 = \{d, q, c\}$

▶  $C_4 = \{\neg q, v\}$

▶ Literale einer Klausel sind (implizit) **oder**-verknüpft

▶ Schreibweise

▶  $C_1 = \neg c \vee \neg s$

▶  $C_3 = d \vee q \vee c$

▶  $C_4 = \neg q \vee v$

**Definition (Klauseln als Menge):** Eine **Klausel** ist eine Menge von **Literalen**

► Beispiele

►  $C_1 = \{\neg c, \neg s\}$

►  $C_3 = \{d, q, c\}$

►  $C_4 = \{\neg q, v\}$

► Literale einer Klausel sind (implizit) **oder**-verknüpft

► Schreibweise

►  $C_1 = \neg c \vee \neg s$

►  $C_3 = d \vee q \vee c$

►  $C_4 = \neg q \vee v$

Es besteht eine einfache Beziehung zwischen Klauseln als expliziten Disjunktionen und Klauseln als Mengen - wir unterscheiden die beiden Sichten nur, wenn notwendig!

## Definition (Interpretationen als Literalismengen):

- ▶ Eine **Interpretation** ist eine Menge von Literalen mit:
  - ▶ Für jedes  $p \in P$  ist entweder  $p \in I$  oder  $\neg p \in I$
  - ▶ Für kein  $p \in P$  ist  $p \in I$  und  $\neg p \in I$
- ▶ Eine Klausel  $C$  ist **wahr unter  $I$** , falls eines ihrer Literale in  $I$  vorkommt
- ▶ Formal:  $I(C) = \begin{cases} 1 & \text{falls } C \cap I \neq \emptyset \\ 0 & \text{sonst} \end{cases}$

## Definition (Interpretationen als Literalismengen):

- ▶ Eine **Interpretation** ist eine Menge von Literalen mit:
  - ▶ Für jedes  $p \in P$  ist entweder  $p \in I$  oder  $\neg p \in I$
  - ▶ Für kein  $p \in P$  ist  $p \in I$  und  $\neg p \in I$
- ▶ Eine Klausel  $C$  ist **wahr unter  $I$** , falls eines ihrer Literale in  $I$  vorkommt
- ▶ Formal:  $I(C) = \begin{cases} 1 & \text{falls } C \cap I \neq \emptyset \\ 0 & \text{sonst} \end{cases}$

Auch hier: Diese Definition steht in enger Beziehung zur bekannten:

$$I(a) = \begin{cases} 0 & \text{falls } \neg a \in I \\ 1 & \text{falls } a \in I \end{cases}$$

- ▶ Atome:  $P = \{c, s, w, d, q, v\}$
  - ▶ Klauseln:
    - ▶  $C_1 = \neg c \vee \neg s$
    - ▶  $C_3 = d \vee q \vee c$
    - ▶  $C_4 = \neg q \vee v$
  - ▶ Interpretationen:
    - ▶  $I_1 = \{c, s, w, d, q, v\}$
    - ▶  $I_2 = \{\neg c, s, \neg w, d, \neg q, v\}$
- ▶  $I_1(C_1) = 0$
  - ▶  $I_1(C_3) = 1$
  - ▶  $I_1(C_4) = 1$

- ▶ Atome:  $P = \{c, s, w, d, q, v\}$
  - ▶ Klauseln:
    - ▶  $C_1 = \neg c \vee \neg s$
    - ▶  $C_3 = d \vee q \vee c$
    - ▶  $C_4 = \neg q \vee v$
  - ▶ Interpretationen:
    - ▶  $I_1 = \{c, s, w, d, q, v\}$
    - ▶  $I_2 = \{\neg c, s, \neg w, d, \neg q, v\}$
- ▶  $I_1(C_1) = 0$
  - ▶  $I_1(C_3) = 1$
  - ▶  $I_1(C_4) = 1$
  - ▶  $I_2(C_1) = 1$
  - ▶  $I_2(C_3) = 1$
  - ▶  $I_2(C_4) = 1$

# Semantik von Klauselmengen

- ▶ Die Elemente einer Menge von Klauseln sind implizit **und**-verknüpft
- ▶ Eine Interpretation  $I$  **erfüllt** eine Menge von Klauseln  $S$ , falls sie jede Klausel in  $S$  wahr macht

- ▶ Formal:

$$I(S) = \begin{cases} 1 & \text{falls } I(C) = 1 \text{ für alle } C \in S \\ 0 & \text{sonst} \end{cases}$$

- ▶ Die Begriffe **Modell**, **erfüllbar** und **unerfüllbar** werden entsprechend angepasst.

Damit entspricht eine Klauselmenge einer Formel in **konjunktiver Normalform**. Für die Mengenschreibweise hat sich der Begriff **Klauselnormalform** eingebürgert. Beide Begriffe werden oft synonym verwendet.

Die leere Klausel  $\square = \{\}$  enthält keine Literale.

- ▶ Fakt: Es existiert kein  $I$  mit  $I(\square) = 1$   
... denn  $\emptyset \cap I = \emptyset$  für beliebige  $I$
- ▶ Fakt: Sei  $S$  eine Menge von Klauseln. Falls  $\square \in S$ , so ist  $S$  unerfüllbar.

- ▶ Ziel: Zeige **Unerfüllbarkeit** einer Klauselmenge  $S$
- ▶ Methode: **Saturierung**
  - ▶ Resolution erweitert  $S$  systematisch um neue Klauseln
  - ▶ Jede neue Klausel  $C$  ist **logische Folgerung** von  $S$ 
    - ▶ D.h. für jedes Modell  $I$  von  $S$  gilt  $I(C) = 1$
  - ▶ Wenn  $\square$  hergeleitet wird, dann gilt:
    - ▶  $\square$  ist unerfüllbar
    - ▶ Also:  $\square$  hat kein Modell
    - ▶ Aber: Jedes Modell von  $S$  ist ein Modell von  $\square$
    - ▶ Also:  $S$  hat kein Modell
    - ▶ Also:  $S$  ist unerfüllbar

# Resolutionsregel

- ▶ Idee: Kombiniere Klauseln aus  $S$ , die komplementäre Literale enthalten

**Logisch**

$$\frac{C \vee p \quad D \vee \neg p}{C \vee D}$$

**Mengenschreibweise**

$$\frac{C \uplus \{p\} \quad D \uplus \{\neg p\}}{C \cup D}$$

- ▶ Beachte:
  - ▶  $C$  und  $D$  sind (potentiell leere) Klauseln
  - ▶  $C$  und  $D$  können gemeinsame Literale enthalten
  - ▶  $C \vee p$  und  $D \vee \neg p$  sind die **Prämissen**
  - ▶  $C \vee D$  ist die **Resolvente**

# Automobile in Klauselform

► Elementare Aussagen:

- $c$ : Cabrio
- $s$ : Schiebedach
- $w$ : Subwoover
- $d$ : Doppelauspuff
- $q$ : Quattro
- $v$ : Hoher Verbrauch

1.  $\neg c \vee \neg s$

2.  $\neg c \vee \neg w$

3.  $\neg d \vee q \vee c$

4.  $\neg q \vee v$

5.  $\neg v$

6.  $s$

7.  $d$

► Mögliche Resolventen:

# Automobile in Klauselform

► Elementare Aussagen:

- $c$ : Cabrio
- $s$ : Schiebedach
- $w$ : Subwoover
- $d$ : Doppelauspuff
- $q$ : Quattro
- $v$ : Hoher Verbrauch

1.  $\neg c \vee \neg s$

2.  $\neg c \vee \neg w$

3.  $\neg d \vee q \vee c$

4.  $\neg q \vee v$

5.  $\neg v$

6.  $s$

7.  $d$

► Mögliche Resolventen:

8.  $\neg s \vee \neg d \vee q$  (aus 1 und 3)

# Automobile in Klauselform

► Elementare Aussagen:

- $c$ : Cabrio
- $s$ : Schiebedach
- $w$ : Subwoover
- $d$ : Doppelauspuff
- $q$ : Quattro
- $v$ : Hoher Verbrauch

1.  $\neg c \vee \neg s$

2.  $\neg c \vee \neg w$

3.  $\neg d \vee q \vee c$

4.  $\neg q \vee v$

5.  $\neg v$

6.  $s$

7.  $d$

► Mögliche Resolventen:

8.  $\neg s \vee \neg d \vee q$  (aus 1 und 3)

9.  $\neg c$  (aus 1 und 6)

# Automobile in Klauselform

► Elementare Aussagen:

- ▶  $c$ : Cabrio
- ▶  $s$ : Schiebedach
- ▶  $w$ : Subwoover
- ▶  $d$ : Doppelauspuff
- ▶  $q$ : Quattro
- ▶  $v$ : Hoher Verbrauch

1.  $\neg c \vee \neg s$

2.  $\neg c \vee \neg w$

3.  $\neg d \vee q \vee c$

4.  $\neg q \vee v$

5.  $\neg v$

6.  $s$

7.  $d$

► Mögliche Resolventen:

8.  $\neg s \vee \neg d \vee q$  (aus 1 und 3)
9.  $\neg c$  (aus 1 und 6)
10.  $\neg q$  (aus 4 und 5)

# Automobile in Klauselform

► Elementare Aussagen:

- $c$ : Cabrio
- $s$ : Schiebedach
- $w$ : Subwoover
- $d$ : Doppelauspuff
- $q$ : Quattro
- $v$ : Hoher Verbrauch

1.  $\neg c \vee \neg s$

2.  $\neg c \vee \neg w$

3.  $\neg d \vee q \vee c$

4.  $\neg q \vee v$

5.  $\neg v$

6.  $s$

7.  $d$

► Mögliche Resolventen:

8.  $\neg s \vee \neg d \vee q$  (aus 1 und 3)

9.  $\neg c$  (aus 1 und 6)

10.  $\neg q$  (aus 4 und 5)

11.  $\neg d \vee q$  (aus 3 und 9)

# Automobile in Klauselform

► Elementare Aussagen:

- $c$ : Cabrio
- $s$ : Schiebedach
- $w$ : Subwoover
- $d$ : Doppelauspuff
- $q$ : Quattro
- $v$ : Hoher Verbrauch

1.  $\neg c \vee \neg s$

2.  $\neg c \vee \neg w$

3.  $\neg d \vee q \vee c$

4.  $\neg q \vee v$

5.  $\neg v$

6.  $s$

7.  $d$

► Mögliche Resolventen:

8.  $\neg s \vee \neg d \vee q$  (aus 1 und 3)

9.  $\neg c$  (aus 1 und 6)

10.  $\neg q$  (aus 4 und 5)

11.  $\neg d \vee q$  (aus 3 und 9)

12.  $\neg d$  (aus 11 und 10)

# Automobile in Klauselform

► Elementare Aussagen:

- $c$ : Cabrio
- $s$ : Schiebedach
- $w$ : Subwoover
- $d$ : Doppelauspuff
- $q$ : Quattro
- $v$ : Hoher Verbrauch

1.  $\neg c \vee \neg s$

2.  $\neg c \vee \neg w$

3.  $\neg d \vee q \vee c$

4.  $\neg q \vee v$

5.  $\neg v$

6.  $s$

7.  $d$

► Mögliche Resolventen:

8.  $\neg s \vee \neg d \vee q$  (aus 1 und 3)

9.  $\neg c$  (aus 1 und 6)

10.  $\neg q$  (aus 4 und 5)

11.  $\neg d \vee q$  (aus 3 und 9)

12.  $\neg d$  (aus 11 und 10)

13.  $\square$  (7 und 12)

# Automobile in Klauselform

► Elementare Aussagen:

- $c$ : Cabrio
- $s$ : Schiebedach
- $w$ : Subwoover
- $d$ : Doppelauspuff
- $q$ : Quattro
- $v$ : Hoher Verbrauch

1.  $\neg c \vee \neg s$

2.  $\neg c \vee \neg w$

3.  $\neg d \vee q \vee c$

4.  $\neg q \vee v$

5.  $\neg v$

6.  $s$

7.  $d$

► Mögliche Resolventen:

8.  $\neg s \vee \neg d \vee q$  (aus 1 und 3)

9.  $\neg c$  (aus 1 und 6)

10.  $\neg q$  (aus 4 und 5)

11.  $\neg d \vee q$  (aus 3 und 9)

12.  $\neg d$  (aus 11 und 10)

13.  $\square$  (7 und 12)

Pech gehabt!

# Automobile in Klauselform

► Elementare Aussagen:

- $c$ : Cabrio
- $s$ : Schiebedach
- $w$ : Subwoover
- $d$ : Doppelauspuff
- $q$ : Quattro
- $v$ : Hoher Verbrauch

1.  $\neg c \vee \neg s$

2.  $\neg c \vee \neg w$

3.  $\neg d \vee q \vee c$

4.  $\neg q \vee v$

5.  $\neg v$

6.  $s$

7.  $d$

► Mögliche Resolventen:

8.  $\neg s \vee \neg d \vee q$  (aus 1 und 3)

9.  $\neg c$  (aus 1 und 6)

10.  $\neg q$  (aus 4 und 5)

11.  $\neg d \vee q$  (aus 3 und 9)

12.  $\neg d$  (aus 11 und 10)

13.  $\square$  (7 und 12)

Pech gehabt! Ich fahre weiter Fahrrad!

## Übung: Resolution von Craig 2

Aussagen des Ladenbesitzers:

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

Konvertieren Sie die Aussagen in KNF und zeigen Sie per Resolution die Unerfüllbarkeit.

**Satz:** Resolution ist korrekt.

- ▶ Wenn aus  $S$  die leere Klausel ableitbar ist, so ist  $S$  unerfüllbar

**Satz:** Resolution ist vollständig.

- ▶ Wenn  $S$  unerfüllbar ist, so kann aus  $S$  die leere Klausel abgeleitet werden
- ▶ Wenn die Ableitung fair ist, so wird die leere Klausel hergeleitet werden
  - ▶ Fairness: Jede mögliche Resolvente wird irgendwann berechnet

## Übung: Resolution von Jane

1. Wenn Jane nicht krank ist und zum Meeting eingeladen wird, dann kommt sie zu dem Meeting.  
▶  $\neg K \wedge E \rightarrow M$
2. Wenn der Boss Jane im Meeting haben will, lädt er sie ein.  
▶  $B \rightarrow E$
3. Wenn der Boss Jane nicht im Meeting haben will, fliegt sie raus.  
▶  $\neg B \rightarrow F$
4. Jane war nicht im Meeting.  
▶  $\neg M$
5. Jane war nicht krank.  
▶  $\neg K$
6. **Vermutung:** Jane fliegt raus.  
▶  $F$

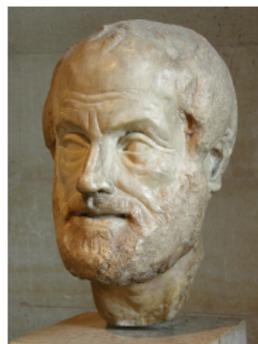
Konvertieren Sie das Folgerungsproblem in ein KNF-Problem und zeigen Sie per Resolution die Unerfüllbarkeit!

# Prädikatenlogik

# Grenzen der Aussagenlogik

Alle Menschen sind sterblich  
Sokrates ist ein Mensch  
-----  
Also ist Sokrates sterblich

Alle Vögel können fliegen  
Ein Pinguin ist ein Vogel  
-----  
Also kann ein Pinguin fliegen



# Prädikatenlogik erster Stufe

- ▶ Logik von **Relationen** und **Funktionen** über einem **Universum**
- ▶ Idee: Atome sind nicht mehr atomar
- ▶ Prädikatssybole repräsentieren **Relationen**
- ▶ Funktionssymbole repräsentieren **Funktionen**
- ▶ Variablen stehen beliebige Objekte (**Universelle** und **existentielle** Quantifizierung)

# Prädikatenlogik erster Stufe

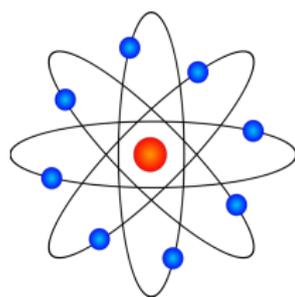
- ▶ Logik von **Relationen** und **Funktionen** über einem **Universum**
- ▶ Idee: Atome sind nicht mehr atomar
- ▶ Prädikatssybole repräsentieren **Relationen**
- ▶ Funktionssymbole repräsentieren **Funktionen**
- ▶ Variablen stehen beliebige Objekte (**Universelle** und **existentielle** Quantifizierung)

▶ Damit:  $\forall X(mensch(X) \rightarrow sterblich(X))$

$$\frac{mensch(sokrates)}{sterblich(sokrates)}$$

# Prädikatenlogik erster Stufe

- ▶ Logik von **Relationen** und **Funktionen** über einem **Universum**
- ▶ Idee: Atome sind nicht mehr atomar
- ▶ Prädikatssybole repräsentieren **Relationen**
- ▶ Funktionssymbole repräsentieren **Funktionen**
- ▶ Variablen stehen beliebige Objekte (**Universelle** und **existentielle** Quantifizierung)
- ▶ Damit:  $\forall X(mensch(X) \rightarrow sterblich(X))$   
 $\frac{mensch(sokrates)}{sterblich(sokrates)}$



Atomspaltung!

## Aussagenlogik

- ▶ Atomare Aussagen

## Prädikatenlogik

- ▶ Objekte (Elemente)
  - ▶ Leute, Häuser, Zahlen, Donald Duck, Farben, Jahre, ...
- ▶ Relationen (Prädikate/Eigenschaften)
  - ▶ rot, rund, prim, mehrstöckig, ...
  - ist Bruder von, ist größer als, ist Teil von, hat Farbe, besitzt, ...
  - =, >, ...
- ▶ Funktionen
  - ▶ +, Mittelwert von, Vater von, Anfang von, ...

- ▶ Eine Signatur  $\Sigma$  ist ein 3-Tupel  $(P, F, V)$
- ▶  $P$ : Menge der Prädikatssymbole mit Stelligkeit
  - ▶  $n$ -stellige Prädikatssymbole repräsentieren  $n$ -stellige Relationen
  - ▶ Z.B.  $P = \{\text{mensch}/1, \text{sterblich}/1, \text{lauter}/2\}$
- ▶  $F$ : Menge der Funktionssymbole mit Stelligkeit
  - ▶ Z.B.  $F = \{\text{gruppe}/2, \text{lehrer}/1, \text{sokrates}/0, \text{aristoteles}/0\}$ 
    - ▶ Konstanten (z.B. *sokrates*) sind 0-stellige Funktionssymbole!
- ▶  $V$ : Abzählbar unendliche Menge von Variablen
  - ▶ Z.B.  $V = \{X, Y, Z, U, X_1, X_2, \dots\}$ 
    - ▶ In Prolog und TPTP: Variablen fangen mit Großbuchstaben an
    - ▶ Literatur: Oft  $\{x, y, z, u, v, x_0, x_1, \dots\}$

- ▶ Gegeben:  $\Sigma = (P, F, V)$
- ▶ Definition:  $T_\Sigma$  ist die Menge der Terme über  $F, V$ 
  - ▶ Sei  $X \in V$ . Dann  $X \in T_\Sigma$
  - ▶ Sei  $f/n \in F$  und seien  $t_1, \dots, t_n \in T_\Sigma$ . Dann ist  $f(t_1, \dots, t_n) \in T_\Sigma$
  - ▶  $T_\Sigma$  ist die kleinste Menge mit diesen Eigenschaften
- ▶ Beispiele:
  - ▶  $F = \{\textit{gruppe}/2, \textit{lehrer}/1, \textit{sokrates}/0, \textit{aristoteles}/0\} \dots\}$
  - ▶ Terme:
    - ▶  $X$
    - ▶  $\textit{sokrates}()$  (normalerweise ohne Klammern geschrieben)
    - ▶  $\textit{gruppe}(X, \textit{sokrates})$
    - ▶  $\textit{guppe}(\textit{lehrer}(\textit{sokrates}), \textit{gruppe}(\textit{sokrates}, Y))$

## Bemerkungen

- ▶ Insbesondere sind alle Konstanten in  $T_\Sigma$ 
  - ▶ Wir schreiben in der Regel  $a, 1$ , statt  $a(), 1()$
- ▶ Terme ohne Variablen heißen **Grundterme** oder einfach **grund**
- ▶ Terme bezeichnen (Mengen von) Elementen des Universums
- ▶ Grundterme bezeichnen einfache Elemente

- ▶ Gegeben:  $\Sigma = (P, F, V)$
- ▶ Definition:  $A_\Sigma$  ist die Menge der Atome über  $P, F, V$ 
  - ▶ Sei  $p/n \in P$  und seien  $t_1, \dots, t_n \in T_\Sigma$ . Dann ist  $p(t_1, \dots, t_n) \in A_\Sigma$
  - ▶  $A_\Sigma$  ist die kleinste Menge mit diesen Eigenschaften
- ▶ Beispiele:
  - ▶  $P = \{\text{mensch}/1, \text{sterblich}/1, \text{lauter}/2\}$ ,
  - ▶  $F = \{\text{gruppe}/2, \text{lehrer}/1, \text{sokrates}/0, \text{aristoteles}/0\}$
- ▶ Atome:
  - ▶  $\text{mensch}(X)$
  - ▶  $\text{lauter}(\text{sokrates}, X)$
  - ▶  $\text{lauter}(\text{sokrates}, \text{sokrates})$
  - ▶  $\text{lauter}(\text{sokrates}, \text{lehrer}(\text{lehrer}(\text{lehrer}(X))))$

## Beispiel

*bruder ( kingJohn, richardTheLionheart )*

## Beispiel

*bruder* ( *kingJohn*, *richardTheLionheart* )

Prädikat      Konstante      Konstante

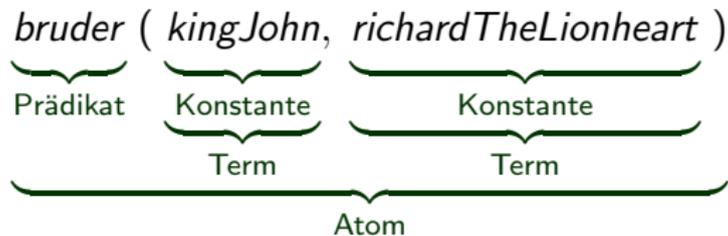
## Beispiel

*bruder* ( *kingJohn*, *richardTheLionheart* )

Prädikat      Konstante      Konstante

                    Term                      Term

## Beispiel



## Beispiel

> ( *laenge(linkesBein(richard)), laenge(linkesBein(kingJohn))* )

## Beispiel

$>$  ( *laenge*(*linkesBein*(*richard*)), *laenge*(*linkesBein*(*kingJohn*)) )

Prädikat    Funktion    Funktion    Konstante    Funktion    Funktion    Konstante

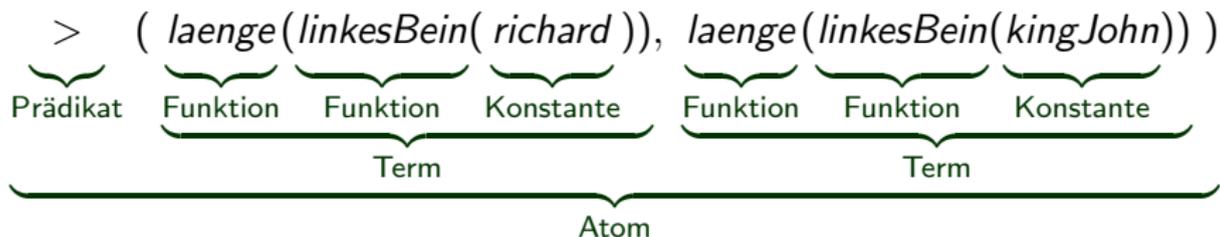
## Beispiel

$>$  ( *laenge*(*linkesBein*(*richard*)), *laenge*(*linkesBein*(*kingJohn*)) )

Prädikat      Funktion      Funktion      Konstante      Funktion      Funktion      Konstante

Term      Term

## Beispiel



Sei  $\Sigma = (\{gt/2, eq/2\}, \{s/1, 0/0\}, \{x, y, z, \dots\})$

- ▶ Geben sie 5 verschiedene Terme an
- ▶ Beschreiben Sie alle variablenfreien Terme
- ▶ Beschreiben Sie alle Terme
- ▶ Geben Sie 5 verschiedene Atome an
- ▶ Fällt Ihnen zu den Symbolen etwas ein?

# Syntax der Prädikatenlogik: Anmerkungen zum Vokabular

- ▶ Wir verlangen, dass  $V, P, F$  disjunkt sind
- ▶ Funktions- und Prädikatssymbole haben eine Stelligkeit  $n \geq 0$ 
  - ▶ Schreibweise manchmal auch:  $f|_n$  oder  $arity(f) = n$
  - ▶ Oft implizit ersichtlich aus der Verwendung
- ▶ Funktionssymbole mit Stelligkeit  $n = 0$  heißen Konstanten  
z.B.  $2, stuttgart, c$

Zurück B

# Syntax der Prädikatenlogik: Logische Zeichen

Wie in der Aussagenlogik:

- $\top$  Symbol für den Wahrheitswert „wahr“
- $\perp$  Symbol für den Wahrheitswert „falsch“
- $\neg$  Negationssymbol („nicht“)
- $\wedge$  Konjunktionssymbol („und“)
- $\vee$  Disjunktionssymbol („oder“)
- $\rightarrow$  Implikationssymbol („wenn . . . dann“)
- $\leftrightarrow$  Symbol für Äquivalenz („genau dann, wenn“)
- $( )$  die beiden Klammern

# Syntax der Prädikatenlogik: Logische Zeichen

Wie in der Aussagenlogik:

- $\top$  Symbol für den Wahrheitswert „wahr“
- $\perp$  Symbol für den Wahrheitswert „falsch“
- $\neg$  Negationssymbol („nicht“)
- $\wedge$  Konjunktionssymbol („und“)
- $\vee$  Disjunktionssymbol („oder“)
- $\rightarrow$  Implikationssymbol („wenn . . . dann“)
- $\leftrightarrow$  Symbol für Äquivalenz („genau dann, wenn“)
- $( )$  die beiden Klammern

Neu: Quantoren

- $\forall$  Allquantor („für alle“)
- $\exists$  Existenzquantor („es gibt“)

**Definition (Formeln der Prädikatenlogik 1. Stufe):** Sei  $\Sigma = (P, F, V)$  eine prädikatenlogische Signatur.

$For_\Sigma$  ist die kleinste Menge mit:

- ▶  $A_\Sigma \subseteq For_\Sigma$
- ▶  $\top \in For_\Sigma$  und  $\perp \in For_\Sigma$
- ▶ Wenn  $A, B \in For_\Sigma$ , dann auch  $\neg A$ ,  $(A \wedge B)$ ,  $(A \vee B)$ ,  $(A \rightarrow B)$ ,  $(A \leftrightarrow B)$  in  $For_\Sigma$

**Definition (Formeln der Prädikatenlogik 1. Stufe):** Sei  $\Sigma = (P, F, V)$  eine prädikatenlogische Signatur.

$For_\Sigma$  ist die kleinste Menge mit:

- ▶  $A_\Sigma \subseteq For_\Sigma$
- ▶  $\top \in For_\Sigma$  und  $\perp \in For_\Sigma$
- ▶ Wenn  $A, B \in For_\Sigma$ , dann auch  $\neg A$ ,  $(A \wedge B)$ ,  $(A \vee B)$ ,  $(A \rightarrow B)$ ,  $(A \leftrightarrow B)$  in  $For_\Sigma$
- ▶ Wenn  $A \in For_\Sigma$  und  $x \in V$ , dann  $\forall xA$ ,  $\exists xA$  in  $For_\Sigma$

Beispiel

## Beispiel

$bruder( kingJohn, richard ) \rightarrow bruder( richard, kingJohn )$

# Syntax der Prädikatenlogik: Komplexe Formeln

## Beispiel

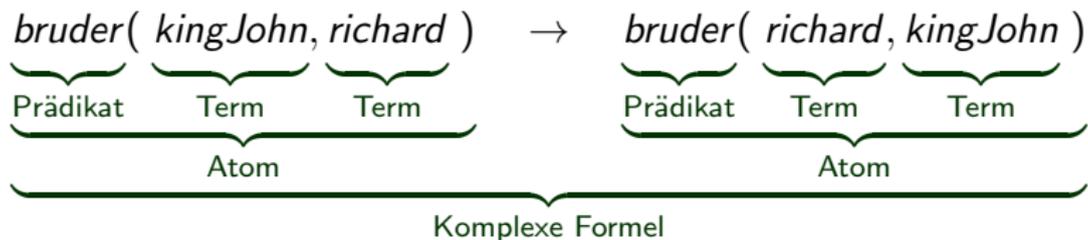
$bruder( kingJohn, richard ) \rightarrow bruder( richard, kingJohn )$

$\underbrace{\hspace{1.5cm}}_{\text{Prädikat}} \quad \underbrace{\hspace{1.5cm}}_{\text{Term}} \quad \underbrace{\hspace{1.5cm}}_{\text{Term}} \quad \rightarrow \quad \underbrace{\hspace{1.5cm}}_{\text{Prädikat}} \quad \underbrace{\hspace{1.5cm}}_{\text{Term}} \quad \underbrace{\hspace{1.5cm}}_{\text{Term}}$



# Syntax der Prädikatenlogik: Komplexe Formeln

## Beispiel



# Übung: Formeln der Prädikatenlogik 1. Stufe

Formalisieren Sie:

- ▶ „Alle, die in Stuttgart studieren, sind schlau“
- ▶ „Es gibt jemand, der in Mannheim studiert und schlau ist“
- ▶ „Die Summe zweier Primzahlen ist eine Primzahl“



Gottlob Frege  
(1879):  
*„Begriffsschrift, eine  
der arithmetischen  
nachgebildete  
Formelsprache des  
reinen Denkens“*

# Freie und gebundene Variablen

## Definition (Freie / gebundene Variable)

Ein Vorkommen einer Variablen  $X$  heißt

- ▶ **gebunden**, wenn sie im Bereich (**Skopus**) einer Quantifizierung  $\forall X / \exists X$  ist
- ▶ **frei** sonst

# Freie und gebundene Variablen

## Definition (Freie / gebundene Variable)

Ein Vorkommen einer Variablen  $X$  heißt

- ▶ **gebunden**, wenn sie im Bereich (**Skopus**) einer Quantifizierung  $\forall X / \exists X$  ist
- ▶ **frei** sonst

Beispiel

$$p(z) \rightarrow \forall x (q(x, z) \wedge \exists z r(y, z))$$

# Freie und gebundene Variablen

## Definition (Freie / gebundene Variable)

Ein Vorkommen einer Variablen  $X$  heißt

- ▶ **gebunden**, wenn sie im Bereich (Skopus) einer Quantifizierung  $\forall X / \exists X$  ist
- ▶ **frei** sonst

## Beispiel

$$p(z) \rightarrow \forall x (q(x, z) \wedge \exists z r(y, z))$$

- ▶  $x$  gebunden
- ▶  $y$  frei
- ▶  $z$  frei und gebunden (verwirrend, sollte vermieden werden!)

- ▶ Nach unserer Definition werden alle Funktions- und Prädikatssymbole in Prefix-Notation verwendet:
  - ▶  $=(1, 2)$ ,  $even(2)$ ,  $+(3, 5)$ ,  $multiply(2, 3)$
- ▶ Konvention: Zweistellige Symbole mit bekannter Semantik werden gelegentlich Infix geschrieben
  - ▶ Insbesondere das Gleichheitsprädikat  $=$
  - ▶ Im Bereich SMT auch  $>$ ,  $+$ ,  $*$ ,  $\leq$ ,  $\dots$

## Semantik?



## **Definition (Prädikatenlogische Interpretation):**

Eine (prädikatenlogische) Interpretation ist ein Paar  $\langle U, A \rangle$ , wobei

$U$ : ist eine nicht-leere Menge (Universum)

$I$ : ist eine Interpretationsfunktion – sie interpretiert

- (freie) Variablen: durch ein Element des Universums
- Prädikate: durch eine Relation auf dem Universum  
(mit passender Stelligkeit)
- Funktionen: durch eine Funktion auf dem Universum  
(mit passender Stelligkeit)

## Bemerkungen

- ▶ Im allgemeinen ist das Universum  $U$  eines prädikatenlogischen Modells unendlich
- ▶ Auch schon für ein endliches  $U$  gibt es eine riesige Zahl verschiedener Interpretationen

## Notation

Sei  $\langle U, A \rangle$  eine prädikatenlogische Interpretation und  $s \in P \cup F \cup V$ .  
Dann sei:

$$s^I = I(s)$$

Also:  $s^I$  bezeichnet  $I(s)$ , die Interpretation des Prädikats-, Funktions- oder Variablensymbols  $s$  unter der gegebenen Interpretation

## Definition (Semantik eines Terms $t$ ):

Sei eine prädikatenlogische Interpretation  $\langle U, I \rangle$  gegeben.

Die Semantik von  $t \in \text{Term}_{\Sigma}$  ist das Element  $I(t)$  aus  $U$ , das rekursiv definiert ist durch

- ▶ Ist  $t = x$  eine Variable:  $I(t) = x^I$
- ▶ Ist  $t = c$  eine Konstante:  $I(t) = c^I$
- ▶ Ist  $t = f(t_1, \dots, t_n)$ :  $I(t) = f^I(I(t_1), \dots, I(t_n))$

## Definition (Semantik einer Formel)

Sei eine prädikatenlogische Interpretation  $\langle U, I \rangle$  gegeben.

Die Semantik  $I(F)$  einer Formel  $F$  unter  $I$  ist einer der Wahrheitswerte 1 oder 0

$$I(\top) = 1$$

$$I(\perp) = 0$$

$$I(p(t_1, \dots, t_n)) = p^I(I(t_1), \dots, I(t_n))$$

und (wie in der Aussagenlogik):

$$I(\neg F) = \begin{cases} 0 & \text{falls } I(F) = 1 \\ 1 & \text{falls } I(F) = 0 \end{cases}$$

und (wie in der Aussagenlogik):

$$I(F \wedge G) = \begin{cases} 1 & \text{falls } I(F) = 1 \text{ und } I(G) = 1 \\ 0 & \text{sonst} \end{cases}$$

$$I(F \vee G) = \begin{cases} 1 & \text{falls } I(F) = 1 \text{ oder } I(G) = 1 \\ 0 & \text{sonst} \end{cases}$$

und (wie in der Aussagenlogik):

$$I(F \rightarrow G) = \begin{cases} 1 & \text{falls } I(F) = 0 \text{ oder } I(G) = 1 \\ 0 & \text{sonst} \end{cases}$$

$$I(F \leftrightarrow G) = \begin{cases} 1 & \text{falls } I(F) = I(G) \\ 0 & \text{sonst} \end{cases}$$

und zusätzlich:

$$I(\forall xF) = \begin{cases} 1 & \text{falls } I_{x/d}(F) = 1 \text{ für alle } d \in U \\ 0 & \text{sonst} \end{cases}$$

und zusätzlich:

$$I(\forall xF) = \begin{cases} 1 & \text{falls } I_{x/d}(F) = 1 \text{ für alle } d \in U \\ 0 & \text{sonst} \end{cases}$$

$$I(\exists xF) = \begin{cases} 1 & \text{falls } I_{x/d}(F) = 1 \text{ für mindestens ein } d \in U \\ 0 & \text{sonst} \end{cases}$$

und zusätzlich:

$$I(\forall xF) = \begin{cases} 1 & \text{falls } I_{x/d}(F) = 1 \text{ für alle } d \in U \\ 0 & \text{sonst} \end{cases}$$

$$I(\exists xF) = \begin{cases} 1 & \text{falls } I_{x/d}(F) = 1 \text{ für mindestens ein } d \in U \\ 0 & \text{sonst} \end{cases}$$

wobei

$I_{x/d}$  identisch mit  $I$  mit der Ausnahme, dass  $x^{I_{x/d}} = d$

## Übung: Auswertung von Formeln

Betrachten Sie die Formel  $F = \forall X(\exists Y(gt(X, plus(Y, 1)))$

- ▶ Sei  $U = \{T, F\}$  und  $I(gt) = \{(T, F)\}$ ,  $I(plus) = \{((F, F), F), ((F, T), T), ((T, F), T), ((T, T), T)\}$ ,  $I(1) = T$ . Bestimmen Sie  $I(F)$ .
- ▶ Sei  $U = \mathbb{N}$  und  $I(gt) = \Rightarrow$ ,  $I(plus) = +$ ,  $I(1) = 1$ . Bestimmen Sie  $I(F)$ .

## Quantoren gleicher Art kommutieren

$\forall x \forall y$  ist äquivalent zu  $\forall y \forall x$   
 $\exists x \exists y$  ist das gleiche wie  $\exists y \exists x$

# Eigenschaften von Quantoren

Verschiedene Quantoren kommutieren NICHT

$\exists x \forall y$  ist nicht äquivalent zu  $\forall y \exists x$

# Eigenschaften von Quantoren

Verschiedene Quantoren kommutieren NICHT

$\exists x \forall y$  ist **nicht** äquivalent zu  $\forall y \exists x$

Beispiel

$\exists x \forall y \text{ loves}(x, y)$

Es gibt eine Person, die jeden Menschen in der Welt liebt  
(einschließlich sich selbst)

$\forall y \exists x \text{ loves}(x, y)$

Jeder Mensch wird von mindestens einer Person geliebt

(Beides ist hoffentlich wahr, aber verschieden:  
das erste impliziert das zweite, aber nicht umgekehrt)

# Eigenschaften von Quantoren

Verschiedene Quantoren kommutieren NICHT

$\exists x \forall y$  ist **nicht** das gleiche wie  $\forall y \exists x$

Beispiel

$\forall x \exists y \text{ mutter}(y, x)$

Jeder hat eine Mutter (richtig)

$\exists y \forall x \text{ mutter}(y, x)$

Es gibt eine Person, die die Mutter von jedem ist (falsch)

## Dualität der Quantoren

$\forall x \dots$  ist äquivalent zu  $\neg \exists x \neg \dots$

$\exists x \dots$  ist äquivalent zu  $\neg \forall x \neg \dots$

## Dualität der Quantoren

$\forall x \dots$  ist äquivalent zu  $\neg \exists x \neg \dots$

$\exists x \dots$  ist äquivalent zu  $\neg \forall x \neg \dots$

## Beispiel

$\forall x \text{ mag}(x, \text{eiskrem})$  ist äquivalent zu  $\neg \exists x \neg \text{mag}(x, \text{eiskrem})$

$\exists x \text{ mag}(x, \text{broccoli})$  ist äquivalent zu  $\neg \forall x \neg \text{mag}(x, \text{broccoli})$

# Eigenschaften von Quantoren

$\forall$  distribuiert über  $\wedge$

$\forall x (\dots \wedge \dots)$  ist äquivalent zu  $(\forall x \dots) \wedge (\forall x \dots)$

$\forall$  distribuiert über  $\wedge$

$\forall x (\dots \wedge \dots)$  ist äquivalent zu  $(\forall x \dots) \wedge (\forall x \dots)$

Beispiel

$\forall x (\text{studiert}(x) \wedge \text{arbeitet}(x))$  ist äquivalent zu  
 $(\forall x \text{ studiert}(x)) \wedge (\forall x \text{ arbeitet}(x))$

$\exists$  distributiert über  $\vee$

$\exists x (\dots \vee \dots)$  ist äquivalent zu  $(\exists x \dots) \vee (\exists x \dots)$

# Eigenschaften von Quantoren

$\exists$  distribuiert über  $\vee$

$\exists x (\dots \vee \dots)$  ist äquivalent zu  $(\exists x \dots) \vee (\exists x \dots)$

Beispiel

$\exists x (eiskrem(x) \vee broccoli(x))$  ist äquivalent zu  
 $(\exists x eiskrem(x)) \vee (\exists x broccoli(x))$

# Eigenschaften von Quantoren

$\forall$  distribuiert **NICHT** über  $\vee$

$\forall x (\dots \vee \dots)$  ist NICHT äquivalent zu  $(\forall x \dots) \vee (\forall x \dots)$

# Eigenschaften von Quantoren

$\forall$  distribuiert **NICHT** über  $\vee$

$\forall x (\dots \vee \dots)$  ist NICHT äquivalent zu  $(\forall x \dots) \vee (\forall x \dots)$

Beispiel

$\forall x (eiskrem(x) \vee broccoli(x))$  ist NICHT äquivalent zu  
 $(\forall x eiskrem(x)) \vee (\forall x broccoli(x))$

$\exists$  distribuiert **NICHT** über  $\wedge$

$\exists x (\dots \wedge \dots)$  ist NICHT äquivalent zu  $(\exists x \dots) \wedge (\exists x \dots)$

# Eigenschaften von Quantoren

$\exists$  distribuiert **NICHT** über  $\wedge$

$\exists x (\dots \wedge \dots)$  ist NICHT äquivalent zu  $(\exists x \dots) \wedge (\exists x \dots)$

Beispiel

$\exists x (\textit{gerade}(x) \wedge \textit{ungerade}(x))$  ist NICHT äquivalent zu  
 $(\exists x \textit{gerade}(x)) \wedge (\exists x \textit{ungerade}(x))$

## Beispiele: Familienverhältnisse

- ▶ „Brüder sind Geschwister“

$$\forall x \forall y (bruder(x, y) \rightarrow geschwister(x, y))$$

- ▶ „bruder“ ist symmetrisch

$$\forall x \forall y (bruder(x, y) \leftrightarrow bruder(y, x))$$

- ▶ „Mütter sind weibliche Elternteile“

$$\forall x \forall y (mutter(x, y) \leftrightarrow (weiblich(x) \wedge elter(x, y)))$$

- ▶ „Ein Cousin ersten Grades ist  
das Kind eines Geschwisters eines Elternteils“

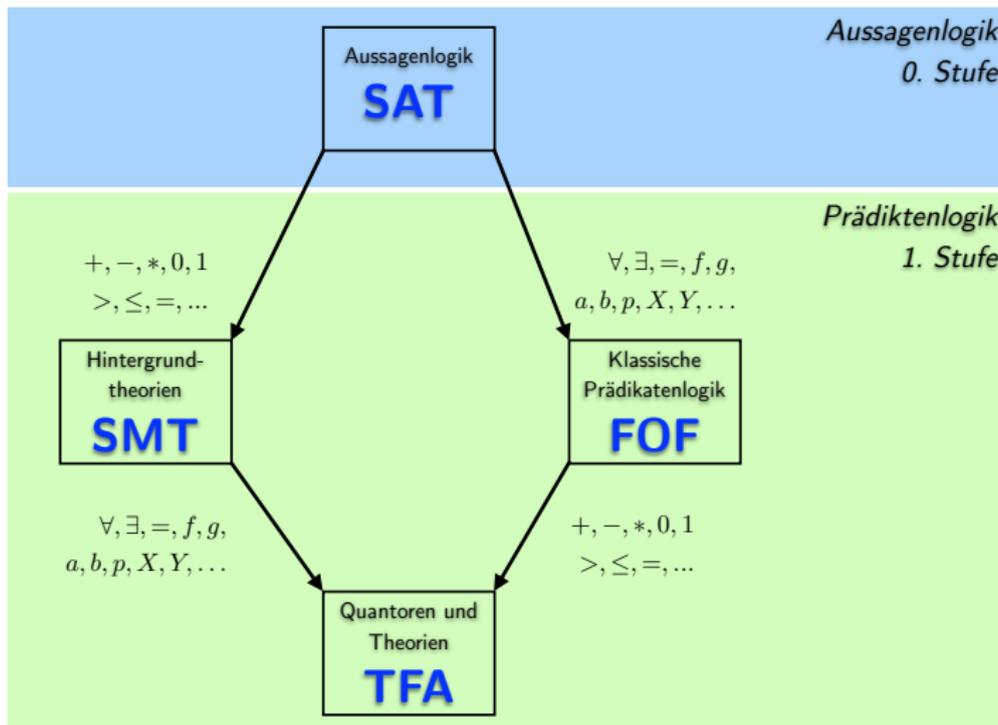
$$\forall x \forall y (cousin1(x, y) \leftrightarrow \\ \exists p \exists ps (elter(p, x) \wedge geschwister(ps, p) \wedge elter(ps, y)))$$

## **Definition:**

- ▶ Modell
- ▶ Allgemeingültigkeit
- ▶ Erfüllbarkeit
- ▶ Unerfüllbarkeit
- ▶ Logische Folgerung
- ▶ Logische Äquivalenz

sind für klassische Prädikatenlogik genauso definiert wie für Aussagenlogik  
(nur mit prädikatenlogischen Interpretationen/Modellen)

# Prädikatenlogik im Kontext



# Übung: Primzahlen

- ▶ Formalisieren Sie das Konzept *Primzahl* und die Aussage „Zu jeder Primzahl gibt es eine größere Primzahl“.
- ▶ Beschreiben Sie ein geeignetes Modell für Ihre Formel(n)
- ▶ Geben Sie eine zweite Interpretation an, in der die Aussage „Zu jeder Primzahl gibt es eine größere Primzahl“ falsch ist.

- ▶ Wir betrachten nur geschlossene Formeln (bei denen alle Variablen gebunden sind)
  - ▶ Seien  $F_1, \dots, F_n, G \in For_\Sigma$
  - ▶ Dann gilt:

$$\begin{aligned} & F_1, \dots, F_n \models G \\ & \quad \text{gdw.} \\ & \models (F_1 \wedge \dots \wedge F_n) \rightarrow G \\ & \quad \text{gdw.} \\ & F_1 \wedge \dots \wedge F_n \wedge \neg G \text{ ist unerfüllbar} \end{aligned}$$

- ▶ Wir betrachten nur geschlossene Formeln (bei denen alle Variablen gebunden sind)
  - ▶ Seien  $F_1, \dots, F_n, G \in \text{For}_\Sigma$
  - ▶ Dann gilt:

$$\begin{aligned} & F_1, \dots, F_n \models G \\ & \quad \text{gdw.} \\ & \models (F_1 \wedge \dots \wedge F_n) \rightarrow G \\ & \quad \text{gdw.} \\ & F_1 \wedge \dots \wedge F_n \wedge \neg G \text{ ist unerfüllbar} \end{aligned}$$

**Wie können wir Unerfüllbarkeit zeigen?**

- ▶ NNF: Wie in Aussagenlogik
  - ▶ Nur  $\{\neg, \vee, \wedge\}$  (aber  $\forall, \exists$  sind erlaubt)
  - ▶  $\neg$  nur direkt vor Atomen
- ▶ NNF Transformation:
  - ▶ Elimination von  $\leftrightarrow, \rightarrow$  wie bei Aussagenlogik
  - ▶ Nach-innen-schieben von  $\neg$ 
    - ▶ De-Morgan
    - ▶  $\neg(\forall x F) \equiv \exists x(\neg F)$
    - ▶  $\neg(\exists x F) \equiv \forall x(\neg F)$

# Elimination von $\exists$ : Skolemisierung

- ▶ Idee: Betrachte Formel der Form  $\exists yF$ 
  - ▶ Wenn  $\exists yF$  erfüllbar ist, dann gibt es eine Interpretation und (mindestens) einen Wert für  $y$ , mit der  $F$  wahr wird.
  - ▶ Wir können die Variable durch eine neue Konstante ersetzen und die Interpretation so anpassen, dass sie diese Konstante durch diesen Wert interpretiert.
- ▶ Idee: Betrachte  $\forall x\exists yF$ 
  - ▶ Wenn ein solches  $y$  existiert, dann hängt  $y$  nur von  $x$  ab
  - ▶ Also:  $y$  kann durch  $f(x)$  ersetzt werden, wobei  $f$  ein neues Funktionssymbol ist (und geeignet interpretiert wird).



Thoralf Skolem  
(1887–1963)

Ergebnis: Formel ohne Existenzquantor

**Definition (Literale):** Sei  $\Sigma = (P, F, V)$  eine prädikatenlogische Signatur und  $A_\Sigma$  die Menge von Atomen über  $\Sigma$ . Sei  $a \in A_\Sigma$ . Dann sind  $a, \neg a$  **Literale**.

**Definition:** Eine **Klausel** ist eine Menge von Literalen.

- ▶ Beispiele
  - ▶  $C_1 = \{\neg \text{mensch}(X), \text{sterblich}(X)\}$
  - ▶  $C_2 = \{\neg \text{lauter}(\text{sokrates}, \text{sokrates})\}$
  - ▶  $C_3 = \{\neg \text{lauter}(X, \text{sokrates}), \neg \text{sterblich}(X)\}$
- ▶ Literale einer Klausel sind (implizit) **oder**-verknüpft
- ▶ Schreibweise
  - ▶  $C_1 = \neg \text{mensch}(X) \vee \text{sterblich}(X)$
  - ▶  $C_2 = \neg \text{lauter}(\text{sokrates}, \text{sokrates})$
  - ▶  $C_3 = \neg \text{lauter}(X, \text{sokrates}) \vee \neg \text{sterblich}(X)$
- ▶ Alle Variablen in Klauseln sind implizit  $\forall$ -quantifiziert

**Definition (Klauselnormalform):**

Eine Formel in **Klauselnormalform** ist eine Menge von Klauseln.

- ▶ Die Klauseln werden als implizit „und“-verknüpft betrachtet
- ▶ Skolemisierte Formeln in NNF können durch Äquivalenzumformungen in KNF gebracht werden

## **Definition (Klauselnormalform):**

Eine Formel in **Klauselnormalform** ist eine Menge von Klauseln.

- ▶ Die Klauseln werden als implizit „und“-verknüpft betrachtet
- ▶ Skolemisierte Formeln in NNF können durch Äquivalenzumformungen in KNF gebracht werden

Gesucht: Verfahren, die Unerfüllbarkeit einer Formel in Klauselnormalform zu zeigen

# Herbrand-Universum (Jacques Herbrand (1908-1931))

**Definition (Herbrand-Universum):** Sei  $\Sigma = (P, F, V)$  eine Signatur und sei  $a/0 \in F$ . Das Herbrand-Universum zu  $\Sigma$  ist die Menge aller Grundterme über  $F$ , also die Menge aller Terme über  $F, \{\}$ .

- ▶ Das Herbrand-Universum besteht aus allen variablenfreien Termen, die aus der Signatur gebildet werden können.
- ▶ Falls die Signatur keine Konstante enthält, so fügen wir eine beliebige Konstante hinzu.

# Herbrand-Universum (Jacques Herbrand (1908-1931))

**Definition (Herbrand-Universum):** Sei  $\Sigma = (P, F, V)$  eine Signatur und sei  $a/0 \in F$ . Das Herbrand-Universum zu  $\Sigma$  ist die Menge aller Grundterme über  $F$ , also die Menge aller Terme über  $F, \{\}$ .

- ▶ Das Herbrand-Universum besteht aus allen variablenfreien Termen, die aus der Signatur gebildet werden können.
- ▶ Falls die Signatur keine Konstante enthält, so fügen wir eine beliebige Konstante hinzu.

**Satz:** Eine Formeln in Klauselnormalform ist genau dann erfüllbar, wenn es eine Modell  $\langle U, I \rangle$  gibt, wobei  $U$  das Herbrand-Universum ist und  $I$  die Funktionen als **Konstruktoren** interpretiert.

**Definition:** Eine **Substitution** ist eine Funktion  $\sigma : V \rightarrow T_\Sigma$  mit der Eigenschaft, dass  $\{X \in V \mid \sigma(X) \neq X\}$  endlich ist.

- ▶  $\sigma$  weist Variablen Terme zu
  - ▶ Wir schreiben z.B.:  $\sigma = \{X \leftarrow sokrates, Y \leftarrow lehrer(aristoteles)\}$
- ▶ Anwendung auf Terme, Atome, Literale, Klauseln möglich!
- ▶ Beispiel:
  - ▶  $\sigma(Y) = lehrer(aristoteles)$
  - ▶  $\sigma(lauter(X, aristoteles)) = lauter(sokrates, aristoteles)$
  - ▶  $\sigma(\neg mensch(X) \vee sterblich(X)) = \neg mensch(sokrates) \vee sterblich(sokrates)$

# Übung: Substitutionen

Sei  $\sigma = \{X \leftarrow a, Y \leftarrow f(X, Y), Z \leftarrow g(a)\}$ . Berechnen Sie:

- ▶  $\sigma(h(X, X, X))$
- ▶  $\sigma(f(g(X), Z))$
- ▶  $\sigma(f(Z, g(X)))$
- ▶  $\sigma(p(X) \vee \neg q(X, f(Y, Z)) \vee p(a))$

# Übung: Substitutionen

Sei  $\sigma = \{X \leftarrow a, Y \leftarrow f(X, Y), Z \leftarrow g(a)\}$ . Berechnen Sie:

- ▶  $\sigma(h(X, X, X))$
- ▶  $\sigma(f(g(X), Z))$
- ▶  $\sigma(f(Z, g(X)))$
- ▶  $\sigma(p(X) \vee \neg q(X, f(Y, Z)) \vee p(a))$

Fällt Ihnen etwas auf?

**Definition (Instanzen):** Sei  $t$  ein Term,  $a$  ein Atom,  $l$  ein Literal,  $c$  eine Klausel und  $\sigma$  eine Substitution.

- ▶  $\sigma(t)$  ist eine **Instanz** von  $t$ .
  - ▶  $\sigma(a)$  ist eine **Instanz** von  $a$ .
  - ▶  $\sigma(l)$  ist eine **Instanz** von  $l$ .
  - ▶  $\sigma(c)$  ist eine **Instanz** von  $c$ .
- 
- ▶ Wenn die Instanz keine Variablen (mehr) enthält, so heißt sie auch eine **Grundinstanz**.

**Satz:** Eine Menge von Klauseln ist genau dann unerfüllbar, wenn die Menge aller ihrer Grundinstanzen (aussagenlogisch) unerfüllbar ist.

- ▶ Dabei betrachten wir die Grundatome als aussagenlogische Variablen
- ▶ Problem: Die Menge der Grundinstanzen ist in der Regel unendlich

**Satz:** Eine Menge von Klauseln ist genau dann unerfüllbar, wenn die Menge aller ihrer Grundinstanzen (aussagenlogisch) unerfüllbar ist.

- ▶ Dabei betrachten wir die Grundatome als aussagenlogische Variablen
- ▶ Problem: Die Menge der Grundinstanzen ist in der Regel unendlich

Resolution für Prädikatenlogik kombiniert das Finden von Instanzen und das Herleiten der leeren Klausel.

# Idee: Resolution für Prädikatenlogik

- ▶ Betrachte folgende Klauseln:
  - ▶  $C_1 = p(X, a)$
  - ▶  $C_2 = \neg p(f(b), Y)$
- ▶ Dann gilt:  $\{C_1, C_2\}$  ist unerfüllbar

# Idee: Resolution für Prädikatenlogik

- ▶ Betrachte folgende Klauseln:
  - ▶  $C_1 = p(X, a)$
  - ▶  $C_2 = \neg p(f(b), Y)$
- ▶ Dann gilt:  $\{C_1, C_2\}$  ist unerfüllbar
  - ▶ Betrachte  $\sigma = \{X \leftarrow f(b), y \leftarrow a\}$ 
    - ▶  $\sigma(C_1) = p(f(b), a)$
    - ▶  $\sigma(C_2) = \neg p(f(b), a)$
  - ▶ Die beiden Instanzen sind jetzt direkt widersprüchlich.

# Idee: Resolution für Prädikatenlogik

- ▶ Betrachte folgende Klauseln:
  - ▶  $C_1 = p(X, a)$
  - ▶  $C_2 = \neg p(f(b), Y)$
- ▶ Dann gilt:  $\{C_1, C_2\}$  ist unerfüllbar
  - ▶ Betrachte  $\sigma = \{X \leftarrow f(b), y \leftarrow a\}$ 
    - ▶  $\sigma(C_1) = p(f(b), a)$
    - ▶  $\sigma(C_2) = \neg p(f(b), a)$
  - ▶ Die beiden Instanzen sind jetzt direkt widersprüchlich.

Wie finden wir ein solches  $\sigma$  automatisch?

Problem: Gleichheit der Atome reicht nicht mehr

Lösung: Finde geeignete Ersetzungen für Variablen

## Definition (Unifikator):

- ▶ Seien  $s, t \in T_\Sigma$  zwei Terme
- ▶ Seien  $a, b \in A_\Sigma$  zwei Atome

Ein **Unifikator** für  $s, t$  bzw.  $a, b$  ist eine Substitution  $\sigma$  mit  $\sigma(s) = \sigma(t)$  bzw.  $\sigma(a) = \sigma(b)$

- ▶ Fakt: (Allgemeinste) Unifikatoren können systematisch gefunden werden

# Beobachtungen zum Finden von Unifikatoren

1.  $sterblich(X)$ ,  $mensch(X)$  können nie unifizieren
  - ▶ Das erste Symbol unterscheidet sich immer
2.  $X$ ,  $lehrer(X)$  können nie unifizieren
  - ▶ Egal, was für  $X$  eingesetzt wird, ein *lehrer* bleibt immer über
  - ▶ "Occurs-Check"
3.  $X$ ,  $lehrer(sokrates)$  unifizieren mit  $\sigma = \{X \leftarrow lehrer(sokrates)\}$ 
  - ▶ Variablen und die meisten Terme machen kein Problem
4.  $lauter(X, sokrates)$ ,  $lauter(aristoteles, Y)$  unifizieren mit  $\sigma = \{X \leftarrow aristoteles, Y \leftarrow sokrates\}$ 
  - ▶ Der Unifikator setzt sich aus den einzelnen Teilen zusammen

# Unifikation als paralleles Gleichungslösen

Fakt: Das Unifikationproblem wird **einfacher**, wenn man es für Mengen von Termpaaren betrachtet!

- ▶ Gegeben:  $R = \{s_1 = t_1, s_2 = t_2, \dots, s_n = t_n\}$ 
  - ▶ Suche **gemeinsamen Unifikator**  $\sigma$  mit
    - ▶  $\sigma(s_1) = \sigma(t_1)$
    - ▶  $\sigma(s_2) = \sigma(t_2)$
    - ▶ ...
    - ▶  $\sigma(s_n) = \sigma(t_n)$
- ▶ Verwende Transformationssystem
  - ▶ Zustand:  $R, \sigma$ 
    - ▶  $R$ : Menge von Termpaaren
    - ▶  $\sigma$ : Kandidat des Unifikators
  - ▶ Anfangszustand:  $\{s = t\}, \{\}$
  - ▶ Termination:  $\{\}, \sigma$

# Unifikation: Transformationssystem

$$\text{Binden: } \frac{\{x = t\} \cup R, \sigma}{\{x \leftarrow t\}(R), \{x \leftarrow t\} \circ \sigma} \text{ falls } x \notin \text{var}(t)$$

$$\text{Orientieren: } \frac{\{t = x\} \cup R, \sigma}{\{x = t\} \cup R, \sigma} \text{ falls } t \text{ keine Variable ist}$$

$$\text{Zerlegen: } \frac{\{f(s_1, \dots, s_n) = f(t_1, \dots, t_n)\} \cup R, \sigma}{\{s_1 = t_1, \dots, s_n = t_n\} \cup R, \sigma}$$

$$\text{Occurs: } \frac{\{x = t\} \cup R, \sigma}{\text{FAIL}} \text{ falls } x \in \text{var}(t)$$

$$\text{Konflikt: } \frac{\{f(s_1, \dots, s_n) = g(t_1, \dots, t_m)\} \cup R, \sigma}{\text{FAIL}} \text{ falls } f \neq g$$

# Beispiel

| $R$                                       | $\sigma$                                                                         | Regel          |
|-------------------------------------------|----------------------------------------------------------------------------------|----------------|
| $\{f(f(X, g(g(Y))), X) = f(f(Z, Z), U)\}$ | $\{\}$                                                                           | Zerlegen       |
| $\{f(X, g(g(Y))) = f(Z, Z), X = U\}$      | $\{\}$                                                                           | Binden ( $X$ ) |
| $\{f(U, g(g(Y))) = f(Z, Z)\}$             | $\{X \leftarrow U\}$                                                             | Zerlegen       |
| $\{Z = U, g(g(Y)) = Z\}$                  | $\{X \leftarrow U\}$                                                             | Binden ( $Z$ ) |
| $\{g(g(Y)) = U\}$                         | $\{X \leftarrow U,$<br>$Z \leftarrow U\}$                                        | Orientieren    |
| $\{U = g(g(Y))\}$                         | $\{X \leftarrow U,$<br>$Z \leftarrow U\}$                                        | Binden ( $U$ ) |
| $\{\}$                                    | $\{X \leftarrow g(g(Y)),$<br>$Z \leftarrow g(g(Y)),$<br>$U \leftarrow g(g(Y))\}$ |                |

# Resolutionskalkül für die Prädikatenlogik

Der Resolutionskalkül für die Prädikatenlogik besteht aus zwei Inferenzregeln:

- ▶ Resolution:  $\frac{C \vee p \quad D \vee \neg q}{\sigma(C \vee D)}$  falls  $\sigma = \text{mgu}(p, q)$
- ▶ Faktorisieren:  $\frac{C \vee p \vee q}{\sigma(C \vee p)}$  falls  $\sigma = \text{mgu}(p, q)$

# Resolutionskalkül für die Prädikatenlogik

Der Resolutionskalkül für die Prädikatenlogik besteht aus zwei Inferenzregeln:

- ▶ Resolution:  $\frac{C \vee p \quad D \vee \neg q}{\sigma(C \vee D)}$  falls  $\sigma = \text{mgu}(p, q)$
- ▶ Faktorisieren:  $\frac{C \vee p \vee q}{\sigma(C \vee p)}$  falls  $\sigma = \text{mgu}(p, q)$

Anwendung: Leite systematisch und **fair** neue Klauseln aus einer Formelmenge ab.

- ▶ Wenn  $\square$  hergeleitet wird, so ist die Formelmenge unerfüllbar
- ▶ Wenn alle Möglichkeiten ausgeschöpft sind, ohne das  $\square$  hergeleitet wurde, so ist die Formelmenge erfüllbar

# Resolutionskalkül für die Prädikatenlogik

Der Resolutionskalkül für die Prädikatenlogik besteht aus zwei Inferenzregeln:

- ▶ Resolution:  $\frac{C \vee p \quad D \vee \neg q}{\sigma(C \vee D)}$  falls  $\sigma = \text{mgu}(p, q)$
- ▶ Faktorisieren:  $\frac{C \vee p \vee q}{\sigma(C \vee p)}$  falls  $\sigma = \text{mgu}(p, q)$

Anwendung: Leite systematisch und **fair** neue Klauseln aus einer Formelmenge ab.

- ▶ Wenn  $\square$  hergeleitet wird, so ist die Formelmenge unerfüllbar
- ▶ Wenn alle Möglichkeiten ausgeschöpft sind, ohne das  $\square$  hergeleitet wurde, so ist die Formelmenge erfüllbar
- ▶ Es gilt sogar: Wenn das Verfahren unendlich läuft, ohne  $\square$  zu erzeugen, so ist die Formelmenge erfüllbar (aber wir merken es nie)

# Beispiel: Resolution mit Unifikation

- ▶ Sterbliche Philosophen
  - ▶  $C_1 = \neg \text{mensch}(X) \vee \text{sterblich}(X)$
  - ▶  $C_2 = \text{mensch}(\text{sokrates})$
  - ▶  $C_3 = \neg \text{sterblich}(\text{sokrates})$
- ▶ Saturierung
  - ▶ Unifiziere  $\text{sterblich}(\text{sokrates})$  ( $C_3$ ) und  $\text{sterblich}(X)$  ( $C_1$ )
    - ▶  $\sigma = \{X \leftarrow \text{sokrates}\}$
    - ▶  $\sigma(C_1) = \neg \text{mensch}(\text{sokrates}) \vee \text{sterblich}(\text{sokrates})$
    - ▶  $\sigma(C_3) = C_3$
    - ▶ Resolution zwischen  $\sigma(C_1)$  und  $\sigma(C_3)$ :  
 $C_4 = \neg \text{mensch}(\text{sokrates})$
  - ▶ Resolution zwischen  $C_2$  und  $C_4$ :  $\square$
- ▶ Also:  $\{C_1, C_2, C_4\}$  ist unerfüllbar - es gibt keine Möglichkeit, das Sokrates gleichzeitig menschlich und unsterblich ist

# Übung: Tierische Resolution

- ▶ Axiome:
  - ▶ Alle Hunde heulen Nachts
  - ▶ Wer Katzen hat, hat keine Mäuse
  - ▶ Empfindlichen Menschen haben keine Tiere, die Nachts heulen
  - ▶ John hat eine Katze oder einen Hund
  - ▶ John ist empfindlich
- ▶ Behauptung:
  - ▶ John hat keine Mäuse
- ▶ Formalisieren Sie das Problem und zeigen Sie die Behauptung per Resolution

# Übung: Tierische Resolution

- ▶ Axiome:
  - ▶ Alle Hunde heulen Nachts
    - ▶  $h(X) \rightarrow l(X)$
  - ▶ Wer Katzen hat, hat keine Mäuse
    - ▶  $k(X) \wedge \text{hat}(Y, X) \rightarrow (\neg(\text{hat}(Y, Z) \wedge m(Z)))$
  - ▶ Empfindlichen Menschen haben keine Tiere, die Nachts heulen
    - ▶  $e(X) \rightarrow \neg(\text{hat}(X, Y) \wedge l(Y))$
  - ▶ John hat eine Katze oder einen Hund
    - ▶  $\text{hat}(\text{john}, \text{tier}) \wedge (h(\text{tier}) \vee k(\text{tier}))$
  - ▶ John ist empfindlich
    - ▶  $e(\text{john})$
- ▶ Behauptung:
  - ▶ John hat keine Mäuse
    - ▶  $\neg(\text{hat}(\text{john}, \text{maus}) \wedge m(\text{maus}))$
- ▶ Formalisieren Sie das Problem und zeigen Sie die Behauptung per Resolution

# Übung: Tierische Resolution

- ▶ Axiome:
  - ▶ Alle Hunde heulen Nachts
    - ▶  $\neg h(X) \vee l(X)$
  - ▶ Wer Katzen hat, hat keine Mäuse
    - ▶  $\neg k(X) \vee \neg \text{hat}(Y, X) \vee \neg \text{hat}(Y, Z) \vee \neg m(Z)$
  - ▶ Empfindlichen Menschen haben keine Tiere, die Nachts heulen
    - ▶  $\neg e(X) \vee \neg \text{hat}(X, Y) \vee \neg l(Y)$
  - ▶ John hat eine Katze oder einen Hund
    - ▶  $\text{hat}(\text{john}, \text{tier}) \wedge (h(\text{tier}) \vee k(\text{tier}))$
  - ▶ John ist empfindlich
    - ▶  $e(\text{john})$
- ▶ Negierte Behauptung:
  - ▶ John hat Mäuse
    - ▶  $\text{hat}(\text{john}, \text{maus}) \wedge m(\text{maus})$
- ▶ Formalisieren Sie das Problem und zeigen Sie die Behauptung per Resolution

# Übung: Tierische Resolution

► Klauselmenge:

1.  $\neg h(X) \vee l(X)$
2.  $\neg k(X) \vee \neg \text{hat}(Y, X) \vee \neg \text{hat}(Y, Z) \vee \neg m(Z)$
3.  $\neg e(X) \vee \neg \text{hat}(X, Y) \vee \neg l(Y)$
4.  $\text{hat}(\text{john}, \text{tier})$
5.  $h(\text{tier}) \vee k(\text{tier})$
6.  $e(\text{john})$
7.  $\text{hat}(\text{john}, \text{maus})$
8.  $m(\text{maus})$

Zeigen Sie per Resolution, dass dieses System unerfüllbar ist.

- ▶ Resolution und Unifikation: Grundlagen moderner Kalküle für Prädikatenlogik
- ▶ Verfeinerungen
  - ▶ Geordnete Resolution
  - ▶ Hyperresolution
  - ▶ Superposition (Gleichheitsbehandlung)
  - ▶ Sortierte Kalküle und interpretierte Theorien
- ▶ Anwendungen:
  - ▶ Verifikation
  - ▶ Formale Mathematik
  - ▶ Expertensysteme
  - ▶ Prolog
  - ▶ ...

**The End**

## Kurzübersicht Scheme

# Scheme-Übersicht (1 - Definitionen und Konditionale)

Auf den nächsten Seiten werden die in der Vorlesung verwendeten Scheme-Konzepte noch einmal kurz zusammengefasst.

`(define var)`

Definiert eine neue Variable. Optional: Initialisierung mit Wert oder Funktion

`(if tst expr1 expr2)`

Wertet Test und je nach Test eine von zwei Alternativen aus

`cond`

Auswahl zwischen mehreren Alternativen

`and`

Faules "und" (Auswertung nur, bis das Ergebnis feststeht. Rückgabewert ist der Wert des letzten ausgewerteten Ausdrucks)

`or`

Faules "oder" - siehe `and`

`not`

Logische Negation

## Scheme-Übersicht (2 - Gleichheit, Programmstruktur)

`equal?`

`let, let*`

`begin`

Gleichheit von beliebigen Objekten

Einführung lokaler Variablen. Wert  
ist Wert des Rumpfes

Sequenz. Wert ist Wert des letzten  
Ausdrucks der Sequenz

## Scheme-Übersicht (3 - Listen 1)

|                                   |                                                                                                                                                            |
|-----------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>list</code>                 | Liste der Argumente                                                                                                                                        |
| <code>'()</code>                  | Konstante für die leere Liste                                                                                                                              |
| <code>cons</code>                 | Listen-Konstruktor: Gibt ein <code>cons</code> -Paar mit seinen beiden Argumenten zurück. Normalfall: Hängt neues Element vor Liste, gibt Ergebnis zurück. |
| <code>car</code>                  | Gibt erstes Element eines <code>cons</code> -Paares zurück. Normalfall: Erstes Element einer Liste                                                         |
| <code>cdr</code>                  | Gibt zweites Element eines <code>cons</code> -Paares zurück. Normalfall: Liste ohne das erste Element.                                                     |
| <code>caar, cadr, cddr ...</code> | Verschachtelungen von <code>car</code> und <code>cdr</code>                                                                                                |
| <code>append</code>               | Hängt zwei (oder mehr) Listen zusammen                                                                                                                     |

## Scheme-Übersicht (4 - Listen 2)

|                        |                                                           |
|------------------------|-----------------------------------------------------------|
| <code>pair?</code>     | Prüft, ob das Argument ein cons-Paar ist                  |
| <code>list?</code>     | Prüft, ob das Argument eine Liste ist                     |
| <code>null?</code>     | Prüft, ob das Argument die leere Liste ist                |
| <code>list-ref</code>  | Gibt das $k$ -te Element einer Liste zurück               |
| <code>list-tail</code> | Gibt Liste ohne die ersten $k$ Elemente zurück            |
| <code>member</code>    | Sucht Objekt in Liste                                     |
| <code>assoc</code>     | Sucht in Liste von Paaren nach Eintrag mit Objekt im car. |

# Scheme-Übersicht (5 - Booleans, Operationen auf Zahlen)

#t, #f

=

>

<

\*

-

+

/

Boolsche Konstanten

Gleichheit von Zahlen

Größer-Vergleich der Argumente

Kleiner-Vergleich der Argumente

Multiplikation der Argumente

Subtraktion

Addition

Division

## Scheme-Übersicht (6 - I/O)

|                                |                                                                          |
|--------------------------------|--------------------------------------------------------------------------|
| <code>port?</code>             | Test, ob ein Objekt ein IO-Port ist.                                     |
| <code>display</code>           | Freundliche Ausgabe eines Objekts<br>(optional: Port)                    |
| <code>newline</code>           | Zeilenumbruch in der Ausgabe                                             |
| <code>read</code>              | Liest ein Scheme-Objekt                                                  |
| <code>write</code>             | Schreibe ein Scheme-Objekt                                               |
| <code>read-char</code>         | Liest ein Zeichen                                                        |
| <code>write-char</code>        | Schreibt ein Zeichen                                                     |
| <code>peek-char</code>         | Gibt das nächste Zeichen oder <i>eof-object</i> zurück, ohne es zu lesen |
| <code>eof-object?</code>       | Prüft, ob das Argument ein End-of-File repräsentiert                     |
| <code>open-input-port</code>   | Öffne Datei zum Lesen                                                    |
| <code>open-output-port</code>  | Öffne Datei zum Schreiben                                                |
| <code>close-input-port</code>  | Schließt Datei                                                           |
| <code>close-output-port</code> | Schließt Datei                                                           |

## Scheme-Übersicht (7 - Funktional und destruktiv)

|                       |                                                                                                                                                     |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>lambda</code>   | Gibt eine neue Funktion zurück                                                                                                                      |
| <code>map</code>      | Wendet Funktion auf alle Elemente einer Liste an, gibt Liste der Ergebnisse zurück. Bei mehrstelligen Funktionen entsprechend viele Argumentlisten! |
| <code>quote</code>    | Gib einen Ausdruck unausgewertet zurück. Kurzform: <code>'</code>                                                                                   |
| <code>eval</code>     | Wertet einen Scheme-Ausdruck in der mitgegebenen Umgebung aus                                                                                       |
| <code>apply</code>    | Ruft eine Funktion mit einer Argumentenliste auf, gibt das Ergebnis zurück                                                                          |
| <code>set!</code>     | Setzt eine Variable auf einen neuen Wert                                                                                                            |
| <code>set-car!</code> | Setzt das <code>car</code> einer existierenden <code>cons</code> -Zelle                                                                             |
| <code>set-cdr!</code> | Setzt das <code>cdr</code> einer existierenden <code>cons</code> -Zelle                                                                             |

# Scheme-Übersicht (8 - Typsystem)

- ▶ Typprädikate (jedes Objekt hat genau einen dieser Typen):
  - boolean?        #t und #f
  - pair?            cons-Zellen (damit auch nicht-leere Listen)
  - symbol?        Normale Bezeichner, z.B. hallo, \*, symbol?. Achtung: Symbole müssen gequoted werden, wenn man das Symbol, nicht seinen Wert referenzieren will!
  
  - number?        Zahlen: 1, 3.1415, ...
  - char?           Einzelne Zeichen: #\a, #\b, #\7, ...
  - string?        "Hallo", "1", "1/2 oder Otto"
  - vector?        Aus Zeitmangel nicht erwähnt (nehmen Sie Listen)
  - port?           Siehe Vorlesung zu Input/Output
  - procedure?    Ausführbare Funktionen (per define oder lambda)
  - null?           Sonderfall: Die leere Liste '()

## Material TINF14B

- ▶ Vorlesung 1
- ▶ Vorlesung 2
- ▶ Vorlesung 3
- ▶ Vorlesung 4
- ▶ Vorlesung 5
- ▶ Vorlesung 6
- ▶ Vorlesung 7
- ▶ Vorlesung 8
- ▶ Vorlesung 9
- ▶ Vorlesung 10
- ▶ Vorlesung 11
- ▶ Vorlesung 12
- ▶ Vorlesung 13
- ▶ Vorlesung 14
- ▶ Vorlesung 15
- ▶ Vorlesung 16
- ▶ Vorlesung 17
- ▶ Vorlesung 18
- ▶ Vorlesung 19

- ▶ Gegenseitiges Kennenlernen
- ▶ Praktische Informationen
- ▶ Übersicht und Motivation

- ▶ Ihre Erfahrungen
  - ▶ Informatik allgemein?
  - ▶ Programmieren? Sprachen?
- ▶ Ihre Erwartungen?
  - ▶ ...

- ▶ Ihre Erfahrungen
  - ▶ Informatik allgemein?
  - ▶ Programmieren? Sprachen?
- ▶ Ihre Erwartungen?
  - ▶ ...
- ▶ Feedbackrunde am Ende der Vorlesung

## ▶ Vorlesungszeiten

- ▶ Dienstags, 12:15-**14:05**

- ▶ Keine Vorlesung am 2.12., 6.1., 3.2.

- ▶ Freitags, 12:45-15:15

- ▶ Kurze Pausen nach Bedarf

- ▶ 11 Wochen Vorlesung+Klausurwoche

- ▶ Weihnachtspause: 22.12.-4.1.

## ▶ Klausur

- ▶ KW10/2015 (2.-6.3.2015)

- ▶ Voraussichtlich 90 Minuten

- ▶ Genauer Termin wird vom Sekretariat koordiniert

## ▶ Webseite zur Vorlesung

- ▶ `http:`

- `//wwwlehre.dhbw-stuttgart.de/~sschulz/lgli2014.html`

- ▶ Gegenseitiges Kennenlernen
- ▶ Praktische Informationen
- ▶ Übersicht und Motivation

Image credit, when not otherwise specified: Wikipedia, OpenClipart

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

- ▶ Kurzer Rückblick
- ▶ Mathematische Grundbegriffe
- ▶ Grundlagen der Mengenlehre
  - ▶ (eine) Konstruktion der natürlichen Zahlen
  - ▶ Mengenoperationen

- ▶ Praktische Themen
  - ▶ Rechner mit Scheme-Umgebung!
- ▶ Vorlesungsziel: Vokabular und Methoden
- ▶ Ein erstes Formales System: Das MIU-Rätsel
- ▶ Einführung Logik
  - ▶ Beispiel: Äquivalenz von Spezifikation im Bereich ATM/ATC

Zur Vorlesung

Bearbeiten Sie die Übung *Konstruktion der negativen Zahlen*.

- ▶ Kurzer Rückblick
- ▶ Mathematische Grundbegriffe
- ▶ Grundlagen der Mengenlehre
  - ▶ (eine) Konstruktion der natürlichen Zahlen
  - ▶ Mengenoperationen

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

- ▶ Rückblick/Hausaufgabe Termalgebra
- ▶ Übung/Hausaufgabe Mengenoperationen
- ▶ Kartesisches Produkt, Potenzmenge
- ▶ Mengenalgebra

- ▶ Grundbegriffe
  - ▶ Definition
  - ▶ Beweis
- ▶ Mengen
  - ▶ Definition per Aufzählung ( $\{1, 2, 4, 8, \dots\}$ )
  - ▶ Beschreibend ( $\{x \mid x = 2^n, n \in \mathbb{N}\}$  oder  $\{2^n \mid n \in \mathbb{N}\}$ )
  - ▶ Mengeneigenschaften
    - ▶ Ungeordnet (Gegensatz: z.B. Liste)
    - ▶ Keine Duplikate (Stichwort *Multimenge*)
  - ▶ Teilmengen ( $\subseteq$  vs.  $\subset$ ) und Obermengen
  - ▶ Mengengleichheit
  - ▶ Konkrete Mengen:  $\emptyset, \mathbb{N}, \mathbb{N}^+, \mathbb{Z}, \mathbb{Q}, \mathbb{R}$
- ▶ Konstruktion der natürlichen Zahlen
  - ▶ Zahlen repräsentiert als  $\emptyset, \{\emptyset\}, \{\{\emptyset\}\} \dots$  oder äquivalent  $0, s(0), s(s(0)) \dots$
  - ▶ Rekursive Definition von  $a$  (Addition) und  $m$  (Multiplikation)
- ▶ Mengenoperationen ( $\cup, \cap, \setminus, \overline{\phantom{x}}, \Delta$ )

# Rückblick/Hausaufgabe (1)

- ▶ Idee: Wir interpretieren **Mengen** oder **Terme** als Zahlen

| Mengenschreibweise          | Term            | Zahl |
|-----------------------------|-----------------|------|
| $\emptyset$ oder $\{\}$     | 0               | 0    |
| $\{\emptyset\}$             | $s(0)$          | 1    |
| $\{\{\emptyset\}\}$         | $s(s(0))$       | 2    |
| $\{\{\{\emptyset\}\}\}$     | $s(s(s(0)))$    | 3    |
| $\{\{\{\{\emptyset\}\}\}\}$ | $s(s(s(s(0))))$ | 4    |
| ...                         | ...             | ...  |

- ▶ Wir definieren **rekursive** Rechenregeln rein syntaktisch:

- ▶ Addition ( $a$ ):

- ▶  $a(X, 0) = X$
- ▶  $a(X, s(Y)) = s(a(X, Y))$

- ▶ Multiplikation ( $m$ ):

- ▶  $m(X, 0) = 0$
- ▶  $m(X, s(Y)) = a(X, m(X, Y))$

## Rückblick/Hausaufgabe (2)

► Addition ( $a$ ):

$$(1) a(X, 0) = X$$

$$(2) a(X, s(Y)) = s(a(X, Y))$$

► Multiplikation ( $m$ ):

$$(3) m(X, 0) = 0$$

$$(4) m(X, s(Y)) = a(X, m(X, Y))$$

► Beispielrechnung:  $2 \times 2$ :

$$\begin{aligned} & m(s(s(0)), s(s(0))) \\ = & a(s(s(0)), \underline{m(s(s(0)), s(0))}) && (4) \text{ mit } X = s(s(0)), Y = s(0) \\ = & a(s(s(0)), a(s(s(0)), \underline{m(s(s(0)), 0)}) && (3) \text{ mit } X = s(s(0)), Y = 0 \\ = & a(s(s(0)), \underline{a(s(s(0)), 0)}) && (3) \text{ mit } X = s(s(0)) \\ = & a(s(s(0)), s(s(0))) && (1) \text{ mit } X = s(s(0)) \\ = & s(\underline{a(s(s(0)), s(0)}) && (2) \text{ mit } X = s(s(0)), Y = s(0) \\ = & s(s(\underline{a(s(s(0)), 0})) && (2) \text{ mit } X = s(s(0)), Y = 0 \\ = & s(s(s(s(0)))) && (1) \text{ mit } X = s(s(0)), Y = 0 \end{aligned}$$

## Rückblick/Hausaufgabe (3)

- ▶ Erweiterung auf negative Zahlen:

- ▶ Idee:  $p(X) = X - 1$ ,  $n(X) = -X$ ,  $v(X, Y) = X - Y$

$$n(0) = 0$$

$$p(n(X)) = n(s(X)) \quad n(p(X)) = s(n(X))$$

$$p(s(X)) = X \quad s(p(X)) = X$$

$$a(X, 0) = X \quad a(X, s(Y)) = s(a(X, Y))$$

$$v(X, 0) = X \quad v(X, s(Y)) = p(v(X, Y))$$

$$a(X, n(Y)) = v(X, Y)$$

$$m(X, 0) = 0 \quad m(X, s(Y)) = a(X, m(X, Y))$$

$$m(X, n(Y)) = n(m(X, Y))$$

# Erinnerung: Mengenoperationen

▶ Vereinigung:

▶  $M_1 \cup M_2 = \{x \mid x \in M_1 \text{ oder } x \in M_2\}$

▶ Schnitt:

▶  $M_1 \cap M_2 = \{x \mid x \in M_1 \text{ und } x \in M_2\}$

▶ Differenz:

▶  $M_1 \setminus M_2 = \{x \mid x \in M_1 \text{ und } x \notin M_2\}$

▶ Symmetrische Differenz:

▶  $M_1 \triangle M_2 = \{x \mid x \in M_1 \text{ und } x \notin M_2\} \cup \{x \mid x \in M_2 \text{ und } x \notin M_1\}$

▶ Komplement:

▶  $\overline{M_1} = \{x \mid x \notin M_1\}$

Bearbeiten Sie die Übung *Karthesisches Produkt und Potenzmenge*.

- ▶ Rückblick/Hausaufgabe Termalgebra
- ▶ Übung/Hausaufgabe Mengenoperationen
- ▶ Kartesisches Produkt, Potenzmenge
- ▶ Mengenalgebra

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

- ▶ Rückblick und Wiederholung
- ▶ Relationen
  - ▶ Grundkonzepte
  - ▶ Eigenschaften
  - ▶ Darstellung
  - ▶ Relationenalgebra (Anfang)

- ▶ Termalgebra
- ▶ Mengenoperationen:  $\cup, \cap, \setminus, \overline{\phantom{x}}$  (Komplement),  $\Delta$
- ▶ Kartesisches Produkt und Potenzmenge
- ▶ Mengenalgebra

Zur Vorlesung

- ▶ Rückblick und Wiederholung
- ▶ Relationen
  - ▶ Grundkonzepte
  - ▶ Eigenschaften
  - ▶ Darstellung
  - ▶ Relationalenalgebra (Anfang)

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

- ▶ Rückblick und Wiederholung
- ▶ Relationen (Teil 2)
- ▶ Funktionen
- ▶ Kardinalität
- ▶ Erste Schritte mit Scheme

# Wiederholung: Relationen 1

- ▶ Relation über  $M_1, M_2, \dots, M_n$  ist  $R \subseteq M_1 \times M_2 \times \dots \times M_n$ 
  - ▶ N-Stellige Relation besteht aus N-Tupeln
- ▶ Spezialfälle:
  - ▶ Homogen: Alle Mengen sind gleich
  - ▶ Binär:  $R \subseteq M_1 \times M_2$  besteht aus Paaren
  - ▶ Homogen und binär: Wie erwartet, häufiger Sonderfall (Z.B.  $>, \geq, \neq, =, <, \leq \subseteq \mathbb{N} \times \mathbb{N}$ )
- ▶ Eigenschaften von (homogenen) binären Relationen:
  - ▶ Reflexivität:  $\forall x \in M : (x, x) \in R$
  - ▶ Symmetrie:  $\forall x, y \in M : (x, y) \in R \rightsquigarrow (y, x) \in R$
  - ▶ Transitivität:  $\forall x, y, z \in M : (x, y) \in R \text{ and } (y, z) \in R \rightsquigarrow (x, z) \in R$
  - ▶ Linkstotal:  $\forall x \in M \exists y \in M : (x, y) \in R$
  - ▶ Rechtseindeutig:  $\forall x, y, z \in M : (x, y) \in R \text{ and } (x, z) \in R \rightsquigarrow y = z$

# Wiederholung: Relationen 2

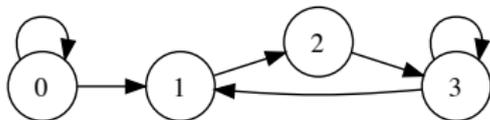
► Relationendarstellung (z.T. nur bei endlichen Mengen praktikabel):

► Als Menge von Paaren

► Aufzählung:  $R = \{(0, 0), (0, 1), (1, 2), (2, 3), (3, 3), (3, 1)\}$

► Aber auch deskriptiv:  $\leq = \{(x, x + i) | x, i \in \mathbb{N}\}$

► Als Relationsgraph



► Als Tabelle oder Matrix

|   | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 2 | 0 | 0 | 0 | 1 |
| 3 | 0 | 3 | 0 | 1 |

bzw.

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 3 & 0 & 1 \end{pmatrix}$$

## Wiederholung: Relationen 3

- ▶ Inverse Relation:  $R^{-1} = \{(y, x) \mid (x, y) \in R\}$
- ▶ Relationsverknüpfung:  
 $R_1 \circ R_2 = \{(x, y) \mid \exists z : (x, z) \in R_1 \text{ und } (z, y) \in R_2\}$
- ▶ Potenzierung einer Relation
  - ▶  $R^0 = \{(x, x) \mid x \in M\}$  (die Gleichheitsrelation oder Identität)
  - ▶  $R^n = R \circ R^{n-1}$  für  $n \in \mathbb{N}^+$
- ▶ Übung:
  - ▶  $M = \{a, b, c, d\}$
  - ▶  $R = \{(a, b), (b, c), (c, d)\}$
  - ▶  $S = \{(a, a), (b, b), (c, c)\}$
  - ▶ Zu berechnen:  $R^{-1}, R^0, R^1, R^2, R^3, R^4, S \circ S, R \circ S$

Bearbeiten Sie Übung: *Relationen für Fortgeschrittene*.

- ▶ Rückblick und Wiederholung
- ▶ Relationen (Teil 2)
- ▶ Funktionen
- ▶ Kardinalität
- ▶ Erste Schritte mit Scheme

# Feedback

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

# Feedback

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?



Schöne Pause!

- ▶ Wiederholung/Hausaufgabe
- ▶ Einführung Scheme
  - ▶ “Hello World” - Starten und Ausführen von Programmen
  - ▶ Interaktives Arbeiten
  - ▶ Einfache Rekursion

# Diskussion Hausaufgabe (1)

- ▶ Betrachten Sie die Menge  $M = \{a, b, c\}$ .
  - ▶ Wie viele (binäre homogene) Relationen über  $M$  gibt es?

# Diskussion Hausaufgabe (1)

- ▶ Betrachten Sie die Menge  $M = \{a, b, c\}$ .
  - ▶ Wie viele (binäre homogene) Relationen über  $M$  gibt es?
  - ▶ Definitionen:
    - ▶ Eine (**n-stellige**) **Relation**  $R$  über  $M_1, M_2, \dots, M_n$  ist eine Teilmenge des kartesischen Produkts der Mengen, also  $R \subseteq M_1 \times M_2 \times \dots \times M_n$
    - ▶  $R$  heißt **homogen**, falls  $M_i = M_j$  für alle  $i, j \in \{1, \dots, n\}$ .
    - ▶  $R$  heißt **binär**, falls  $n = 2$ .
  - ▶ Also: Eine binäre homogene Relation über  $M$  ist eine Teilmenge von  $M \times M$ .

# Diskussion Hausaufgabe (1)

- ▶ Betrachten Sie die Menge  $M = \{a, b, c\}$ .
  - ▶ Wie viele (binäre homogene) Relationen über  $M$  gibt es?
  - ▶ Definitionen:
    - ▶ Eine (**n-stellige**) **Relation**  $R$  über  $M_1, M_2, \dots, M_n$  ist eine Teilmenge des kartesischen Produkts der Mengen, also  $R \subseteq M_1 \times M_2 \times \dots \times M_n$
    - ▶  $R$  heißt **homogen**, falls  $M_i = M_j$  für alle  $i, j \in \{1, \dots, n\}$ .
    - ▶  $R$  heißt **binär**, falls  $n = 2$ .
  - ▶ Also: Eine binäre homogene Relation über  $M$  ist eine Teilmenge von  $M \times M$ .
    - ▶  $M \times M = \{(a, a), (a, b), (a, c), (b, a), (b, b), (b, c), (c, a), (c, b), (c, c)\}$
    - ▶  $|M \times M| = 3 \times 3 = 9$
    - ▶  $|2^{M \times M}| = 2^9 = 512$
  - ▶ Also: Es gibt 512 Teilmengen von  $M \times M$ , also auch 512 binäre homogene Relationen über  $M$ .

## Diskussion Hausaufgabe (2)

- ▶ Wie viele dieser Relationen sind
  - ▶ Linkstotal?

## Diskussion Hausaufgabe (2)

- ▶ Wie viele dieser Relationen sind

- ▶ Linkstotal?

- ▶ Betrachte tabellarische Darstellung:

|   | a | b | c |
|---|---|---|---|
| a |   |   |   |
| b |   |   |   |
| c |   |   |   |

- ▶ Linkstotal: Jede Zeile hat *mindestens eine* 1
- ▶ 7 Möglichkeiten pro Zeile: 100, 010, 001, 110, 101, 011, 111
- ▶ Insgesamt also  $7^3 = 343$

- ▶ Rechtseindeutig?

## Diskussion Hausaufgabe (2)

► Wie viele dieser Relationen sind

► Linkstotal?

► Betrachte tabellarische Darstellung:

|   | a | b | c |
|---|---|---|---|
| a |   |   |   |
| b |   |   |   |
| c |   |   |   |

► Linkstotal: Jede Zeile hat *mindestens eine* 1

► 7 Möglichkeiten pro Zeile: 100, 010, 001, 110, 101, 011, 111

► Insgesamt also  $7^3 = 343$

► Rechtseindeutig?

► Jede Zeile hat *höchstens eine* 1

► 4 Möglichkeiten: 000, 100, 010, 001

► Also:  $4^3 = 64$

## Diskussion Hausaufgabe (3)

- ▶ Wie viele dieser Relationen sind
  - ▶ Reflexiv?

# Diskussion Hausaufgabe (3)

- ▶ Wie viele dieser Relationen sind

- ▶ Reflexiv?

- ▶ Betrachte wieder tabellarische Darstellung:

|   | a | b | c |
|---|---|---|---|
| a | 1 | 2 | 3 |
| b | 4 | 5 | 6 |
| c | 7 | 8 | 9 |

- ▶ Reflexiv: Felder 1, 5, 9 sind fest 1
- ▶ Die anderen 6 Felder können frei gewählt werden
- ▶ Also:  $2^6 = 64$  reflexive Relationen

- ▶ Symmetrisch?

# Diskussion Hausaufgabe (3)

- ▶ Wie viele dieser Relationen sind

- ▶ Reflexiv?

- ▶ Betrachte wieder tabellarische Darstellung:

|   | a | b | c |
|---|---|---|---|
| a | 1 | 2 | 3 |
| b | 4 | 5 | 6 |
| c | 7 | 8 | 9 |

- ▶ Reflexiv: Felder 1, 5, 9 sind fest 1

- ▶ Die anderen 6 Felder können frei gewählt werden

- ▶ Also:  $2^6 = 64$  reflexive Relationen

- ▶ Symmetrisch?

- ▶ Felder 1, 5, 9 können frei gewählt werden

- ▶ 2 und 4 sind gekoppelt, 3 und 7 sind gekoppelt, 6 und 8 sind gekoppelt

- ▶ Also: Ich kann die Werte von 3+3 Feldern wählen

- ▶ Also:  $2^6 = 64$  symmetrische Relationen

## Diskussion Hausaufgabe (4)

- ▶ Wie viele dieser Relationen sind
  - ▶ Transitiv?

# Diskussion Hausaufgabe (4)

- ▶ Wie viele dieser Relationen sind
  - ▶ Transitiv?
    - ▶ Beobachtung: Wenn  $R$  transitiv ist, so gilt: Alles, was ich in zwei Schritten erreichen kann, kann ich auch in einem Schritt erreichen!  
Also:  $R^2 \subseteq R$
    - ▶ Kleines (Scheme-)Programm: 171 der 512 Relationen sind transitiv
  - ▶ Funktionen (einschließlich partieller Funktionen)?

# Diskussion Hausaufgabe (4)

- ▶ Wie viele dieser Relationen sind
  - ▶ Transitiv?
    - ▶ Beobachtung: Wenn  $R$  transitiv ist, so gilt: Alles, was ich in zwei Schritten erreichen kann, kann ich auch in einem Schritt erreichen!  
Also:  $R^2 \subseteq R$
    - ▶ Kleines (Scheme-)Programm: 171 der 512 Relationen sind transitiv
  - ▶ Funktionen (einschließlich partieller Funktionen)?
    - ▶ Siehe oben (rechtseindeutig)
  - ▶ Totale Funktionen?

# Diskussion Hausaufgabe (4)

- ▶ Wie viele dieser Relationen sind
  - ▶ Transitiv?
    - ▶ Beobachtung: Wenn  $R$  transitiv ist, so gilt: Alles, was ich in zwei Schritten erreichen kann, kann ich auch in einem Schritt erreichen!  
Also:  $R^2 \subseteq R$
    - ▶ Kleines (Scheme-)Programm: 171 der 512 Relationen sind transitiv
  - ▶ Funktionen (einschließlich partieller Funktionen)?
    - ▶ Siehe oben (rechtseindeutig)
  - ▶ Totale Funktionen?
    - ▶ Betrachte tabellarische Darstellung:

|   |   |   |   |
|---|---|---|---|
|   | a | b | c |
| a |   |   |   |
| b |   |   |   |
| c |   |   |   |
    - ▶ Jede Zeile hat *genau eine* 1
    - ▶ Also: 3 Möglichkeiten pro Zeile,  $3^3 = 27$  totale Funktionen

## Diskussion Hausaufgabe (5)

- ▶ Betrachten Sie folgende Relation über  $\mathbb{N}$ :  $xRy$  gdw.  
 $x = y + 2$ 
  - ▶ Was ist die transitive Hülle von  $R$ ?

## Diskussion Hausaufgabe (5)

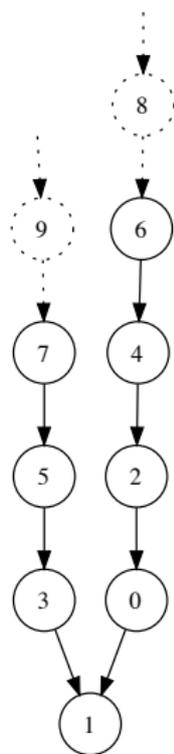
- ▶ Betrachten Sie folgende Relation über  $\mathbb{N}$ :  $xRy$  gdw.  
 $x = y + 2$ 
  - ▶ Was ist die transitive Hülle von  $R$ ?
    - ▶  $xR^+y$  gdw.  $x \bmod 2 = y \bmod 2$  und  $x > y$
  - ▶ Was ist die reflexive, symmetrische, transitive Hülle von

## Diskussion Hausaufgabe (5)

- ▶ Betrachten Sie folgende Relation über  $\mathbb{N}$ :  $xRy$  gdw.  $x = y + 2$ 
  - ▶ Was ist die transitive Hülle von  $R$ ?
    - ▶  $xR^+y$  gdw.  $x \bmod 2 = y \bmod 2$  und  $x > y$
  - ▶ Was ist die reflexive, symmetrische, transitive Hülle von  $R$ ?
    - ▶  $(x, y) \in (R \cup R^{-1})^*$  gdw.  $x \bmod 2 = y \bmod 2$  (alle geraden Zahlen stehen in Relation zueinander, und alle ungeraden Zahlen stehen in Relation zueinander)
  - ▶ Betrachten Sie  $R' = R \cup \{(0, 1)\}$ . Was ist die transitive Hülle von  $R'$ ?

# Diskussion Hausaufgabe (5)

- ▶ Betrachten Sie folgende Relation über  $\mathbb{N}$ :  $xRy$  gdw.  $x = y + 2$ 
  - ▶ Was ist die transitive Hülle von  $R$ ?
    - ▶  $xR^+y$  gdw.  $x \bmod 2 = y \bmod 2$  und  $x > y$
  - ▶ Was ist die reflexive, symmetrische, transitive Hülle von  $R$ ?
    - ▶  $(x, y) \in (R \cup R^{-1})^*$  gdw.  $x \bmod 2 = y \bmod 2$  (alle geraden Zahlen stehen in Relation zueinander, und alle ungeraden Zahlen stehen in Relation zueinander)
  - ▶ Betrachten Sie  $R' = R \cup \{(0, 1)\}$ . Was ist die transitive Hülle von  $R'$ ?
    - ▶  $R'^* = R^* \cup \{(x, 1) \mid x \in \mathbb{N}\}$
    - ▶ ... aber  $(R' \cup R'^{-1})^* = \mathbb{N} \times \mathbb{N}$



Relation  $R'$

## Diskussion Hausaufgabe (6)

- ▶ Zeigen oder widerlegen (per Gegenbeispiel) Sie:
  - ▶ Jede homogene binäre symmetrische und transitive Relation ist eine Äquivalenzrelation

## Diskussion Hausaufgabe (6)

- ▶ Zeigen oder widerlegen (per Gegenbeispiel) Sie:
  - ▶ Jede homogene binäre symmetrische und transitive Relation ist eine Äquivalenzrelation
    - ▶ Behauptung ist falsch! Gegenbeispiel: Die leere Relation ist symmetrisch, transitiv, aber nicht reflexiv und also keine Äquivalenzrelation.
  - ▶ Jede linkstotale homogene binäre symmetrische und transitive Relation ist eine Äquivalenzrelation

# Diskussion Hausaufgabe (6)

- ▶ Zeigen oder widerlegen (per Gegenbeispiel) Sie:
  - ▶ Jede homogene binäre symmetrische und transitive Relation ist eine Äquivalenzrelation
    - ▶ Behauptung ist falsch! Gegenbeispiel: Die leere Relation ist symmetrisch, transitiv, aber nicht reflexiv und also keine Äquivalenzrelation.
  - ▶ Jede linkstotale homogene binäre symmetrische und transitive Relation ist eine Äquivalenzrelation
    - ▶ Behauptung stimmt. Beweis:  
Sei  $R$  eine beliebige linkstotale homogene binäre symmetrische und transitive Relation über einer Menge  $M$ .  
Zu zeigen:  $R$  ist reflexiv  
Sei  $a \in M$  beliebig. Da  $R$  linkstotal ist, existiert  $b \in M$  mit  $aRb$ . Da  $R$  symmetrisch ist, gilt auch  $bRa$ . Da  $R$  auch transitiv ist, und  $aRb$  und  $bRa$  gilt, so gilt auch  $aRa$ . Da wir keine weiteren Annahmen gemacht haben, gilt dieses Argument für alle  $a \in M$ , also ist  $R$  reflexiv.

q.e.d.

Zur Vorlesung

- ▶ Bearbeiten Sie Übung: Fibonacci in Ruhe
- ▶ Können Sie eine Implementierung skizzieren oder sogar realisieren, die für große Zahlen schneller ist?

- ▶ Wiederholung/Hausaufgabe
- ▶ Einführung Scheme
  - ▶ “Hello World” - Starten und Ausführen von Programmen
  - ▶ Interaktives Arbeiten
  - ▶ Einfache Rekursion

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

- ▶ Rückblick/Wiederholung
- ▶ Listenverarbeitung

# Rückblick (1)

- ▶ Hintergrund/Geschichte/Anwendungen
- ▶ Scheme Programme ausführen
  - ▶ Von der Komandozeile (z.B. `guile-2.0 file.scm`)
  - ▶ Interaktiv (read-eval-print loop)
  - ▶ Laden von Programmen in den Interpreter (z.B. `(load file.scm)`)
- ▶ Syntax (*s-expressions*)
  - ▶ Atome (Zahlen, Symbole, Strings, ...)
  - ▶ Listen: `(a1 a2 a3)`
- ▶ Beispiel *Fakultät*

```
;; Factorial
(define (fak x)
 (if (= x 0)
 1
 (* x (fak (- x 1)))))
)
```

## Rückblick (2)

- ▶ Semantik
  - ▶ Strings, Zahlen, ... haben natürlichen Wert
  - ▶ Symbole haben nur dann einen Wert, wenn der (z.B. per *define*) festgelegt wird (sonst Fehler)
  - ▶ Listenauswertung (normal)
    - ▶ Werte alle Teilausdrücke (wenn nötig rekursiv) aus
    - ▶ Interpretiere den Wert des ersten Ausdrucks als Funktion, wende diese auf die Werte der anderen Ausdrücke an
  - ▶ *Special forms*
    - ▶ Z.B. (if *tst expr<sub>1</sub> expr<sub>2</sub>*)
    - ▶ (define pi 3.1415)
    - ▶ (define (plus3 x) (+ x 3))
- ▶ Datentypen und definierte Funktionen
  - ▶ equal?
  - ▶ Zahlen (=, >, +, -, \*, /)
- ▶ Übung/Hausaufgabe: Fibonacci

## Diskussion: Fibonacci (1)

Direkte Umsetzung der Definition:

```
(define (fib x)
 (if (= x 0)
 0
 (if (= x 1)
 1
 (+ (fib (- x 1))
 (fib (- x 2)))))))
```

Korrekt, aber für große  $x$  langsam!

## Diskussion: Fibonacci (2)

- ▶ Idee: Vorwärts rechnen
  - ▶ Wir brauchen immer nur die letzten beiden Werte!
  - ▶ Also Parameter
    - ▶ Letzter Wert
    - ▶ Vorletzter Wert
    - ▶ Wie viele Werte der Folge müssen noch gerechnet werden?

## Diskussion: Fibonacci (2)

- ▶ Idee: Vorwärts rechnen
  - ▶ Wir brauchen immer nur die letzten beiden Werte!
  - ▶ Also Parameter
    - ▶ Letzter Wert
    - ▶ Vorletzter Wert
    - ▶ Wie viele Wert der Folge müssen noch gerechnet werden?

```
(define (fibhelp f1 f2 x)
 (if (= x 0)
 f2
 (fibhelp (+ f1 f2) f1 (- x 1))))
```

```
(define (fibfast x)
 (fibhelp 1 0 x))
```

- ▶ Bearbeiten Sie Übung: Mengenlehre in Scheme, idealerweise komplett, mindestens bis einschließlich `intersection`.

- ▶ Rückblick/Wiederholung
- ▶ Listenverarbeitung

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

## Ziele Vorlesung 8

- ▶ Rückblick/Wiederholung
- ▶ Diskussion: Mengenlehre in Scheme
- ▶ Berechnungsmodell und Speichermodell
- ▶ Neue *special forms*
  - ▶ Sequenzen und `begin`
  - ▶ `cond`
  - ▶ `and/or/not`
  - ▶ `let/let*`
- ▶ Sortieren (Teil 1)

- ▶ Listenfunktionen
  - ▶ Leere liste '()
  - ▶ `null?`
  - ▶ `cons`, `car` und `cdr`
  - ▶ `append` und `list`
- ▶ Rekursion über Listen
  - ▶ Basisfall (`null? 1`)
  - ▶ Rekursion:
    - ▶ Mach was mit (`car 1`)
    - ▶ Rufe die Funktion mit (`cdr 1`) rekursiv auf
    - ▶ Kombiniere die Ergebnisse
- ▶ Beispiel: `revert`

## revert

```
;;; Einfach rekursiv
(define (revert l)
 (if (null? l)
 l
 (append (revert (cdr l)) (list (car l))))))

;;; Alternative: Mit Hilfsliste
(define (revert-help l1 l2)
 (if (null? l1)
 l2
 (revert-help (cdr l1) (cons (car l1) l2))))

(define (revert2 l)
 (revert-help l '()))
```

# Übung: Mengenlehre in Scheme

- ▶ Repräsentieren Sie im folgenden Mengen als Listen
- ▶ Erstellen Sie Scheme-Funktionen für die folgenden Mengen-Operationen:
  - ▶ Einfügen:
    - ▶ `(insert 4 '(1 2 3)) ==> (1 2 3 4)` (Reihenfolge egal)
    - ▶ `(insert 2 '(1 2 3)) ==> (1 2 3)`
  - ▶ Vereinigung:
    - ▶ `(union '(1 2 3) '(3 4 5)) ==> (1 2 3 4 5)`
  - ▶ Schnittmenge:
    - ▶ `(intersection '(1 2 3) '(3 4 5)) ==> (3)`
  - ▶ Kartesisches Produkt:
    - ▶ `(kart '(1 2 3) '(a b c)) ==> ((1 a) (2 a) (3 a) (1 b) (2 b) (3 b) (1 c) (2 c) (3 c))`
  - ▶ Potenzmenge:
    - ▶ `(powerset '(1 2 3)) ==> (() (3) (2) (2 3) (1) (1 3) (1 2) (1 2 3))`
- ▶ Bonus: Implementieren Sie eine Funktion, die die Verkettung von zwei Relationen realisieren!

## Diskussion: Mengenlehre in Scheme (1)

```
(define (is-element? x set)
 (if (null? set)
 #f
 (if (equal? x (car set))
 #t
 (is-element? x (cdr set)))))
```

```
(define (set-insert x set)
 (if (is-element? x set)
 set
 (cons x set)))
```

```
(define (set-union set1 set2)
 (if (null? set1)
 set2
 (set-insert (car set1)
 (set-union (cdr set1) set2))))
```

## Diskussion: Mengenlehre in Scheme (2)

```
(define (set-intersection set1 set2)
 (if (null? set1)
 set1
 (if (is-element? (car set1) set2)
 (cons (car set1)
 (set-intersection (cdr set1) set2))
 (set-intersection (cdr set1) set2))))
```

```
(define (make-pairs m x)
 (if (null? m)
 m
 (cons (list (car m) x)
 (make-pairs (cdr m) x))))
```

```
(define (kart m1 m2)
 (if (null? m2)
 m2
 (append (make-pairs m1 (car m2))
 (kart m1 (cdr m2)))))
```

## Diskussion: Mengenlehre in Scheme (3)

```
(define (add-to-sets sets element)
 (if (null? sets)
 sets
 (cons (cons element (car sets))
 (add-to-sets (cdr sets) element))))
```

```
(define (powerset set)
 (if (null? set)
 (list set)
 (append (powerset (cdr set))
 (add-to-sets (powerset (cdr set))
 (car set)))))
```

Zur Vorlesung

- ▶ Bearbeiten Sie Übung: InsertSort.

- ▶ Rückblick/Wiederholung
- ▶ Diskussion: Mengenlehre in Scheme
- ▶ Berechnungsmodell und Speichermodell
- ▶ Neue *special forms*
  - ▶ Sequenzen und `begin`
  - ▶ `cond`
  - ▶ `and/or/not`
  - ▶ `let/let*`
- ▶ Sortieren (Teil 1)

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

- ▶ Rückblick/Wiederholung
- ▶ (Keine) Seiteneffekte
- ▶ Diskussion: Sortieren durch Einfügen
- ▶ Generisches Sortieren
- ▶ Mergesort

- ▶ Mengenlehre in Scheme
- ▶ Umgebungen und Funktionsaufrufe
- ▶ *Special Forms*
  - ▶ (Pseudo-) Boolesche Verknüpfungen
  - ▶ Lokale Variable mit `let/let*`
- ▶ Sortieren durch Einfügen

## (Keine) Seiteneffekte

- ▶ Anmerkung: Bis jetzt programmieren wir (fast) rein funktional
  - ▶ Funktionen werden **nur** wegen ihres Wertes berechnet
- ▶ Einzige Ausnahmen bisher:
  - ▶ `define` fügt neue Namen zur globalen Umgebung hinzu
  - ▶ Output mit `display`, `newline`
- ▶ Insbesondere:
  - ▶ `(cdr 1)` gibt den Rest von `1` zurück, aber verändert die Liste nicht
  - ▶ `(+ x 10)` gibt die Summe von `x` und `10` zurück, aber verändert nicht den Wert von `x` (vergleiche auch `(+ 10 x)` und `(+ x y)`)

```
> (define l '(1 2 3))
> l
=> (1 2 3)
> (cdr l)
=> (2 3)
> l
=> (1 2 3)
```

## (Keine) Seiteneffekte

- ▶ Anmerkung: Bis jetzt programmieren wir (fast) rein funktional
  - ▶ Funktionen werden **nur** wegen ihres Wertes berechnet
- ▶ Einzige Ausnahmen bisher:
  - ▶ `define` fügt neue Namen zur globalen Umgebung hinzu
  - ▶ Output mit `display`, `newline`
- ▶ Insbesondere:
  - ▶ `(cdr l)` gibt den Rest von `l` zurück, aber verändert die Liste nicht
  - ▶ `(+ x 10)` gibt die Summe von `x` und `10` zurück, aber verändert nicht den Wert von `x` (vergleiche auch `(+ 10 x)` und `(+ x y)`)

```
> (define l '(1 2 3))
> l
=> (1 2 3)
> (cdr l)
=> (2 3)
> l
=> (1 2 3)
```

Wir verwenden (bisher) keine Funktionen, die Objekte verändern!

?

## Sortiertes Einfügen

```
> (define (insert k lst)
 (if (null? lst)
 (list k)
 (if (< k (car lst))
 (cons k lst)
 (cons (car lst)
 (insert k (cdr lst)))))))
```

```
> (insert 7 '(1 3 8 12))
```

```
⇒ (1 3 7 8 12)
```

```
> (insert 2 '())
```

```
⇒ (2)
```

```
> (insert 1 '(2 3))
```

```
⇒ (1 2 3)
```

```
> (insert 8 '(1 2 3))
```

```
⇒ (1 2 3 8)
```

## Sortieren durch Einfügen

```
> (define (isort l)
 (if (null? l)
 l
 (insert (car l) (isort (cdr l)))))
```

```
> (isort '(2 4 1 0 9))
```

```
==> (0 1 2 4 9)
```

```
> (isort '(1 2 4 1 0 9 1))
```

```
==> (0 1 1 1 2 4 9)
```

# Sortieren durch Einfügen

```
> (define (isort l)
 (if (null? l)
 l
 (insert (car l) (isort (cdr l)))))
```

```
> (isort '(2 4 1 0 9))
```

```
⇒ (0 1 2 4 9)
```

```
> (isort '(1 2 4 1 0 9 1))
```

```
⇒ (0 1 1 1 2 4 9)
```

## Alternative:

```
(define (isort l)
 (if (null? l)
 l
 (let* ((first (car l))
 (rest (isort (cdr l))))
 (insert first rest))))
```

- ▶ Bearbeiten Sie Übung: Mergesort.

- ▶ Rückblick/Wiederholung
- ▶ (Keine) Seiteneffekte
- ▶ Diskussion: Sortieren durch Einfügen
- ▶ Generisches Sortieren
- ▶ Mergesort

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

# Ziele Vorlesung 10

- ▶ Rückblick/Wiederholung
- ▶ Hausaufgabe/Mergesort
- ▶ Rekursion abstrakt
- ▶ Input/Output
- ▶ Die Wahrheit<sup>TM</sup> über Listen

- ▶ Sortieren durch Einfügen
- ▶ Generisches Sortieren - Funktionen als Parameter
- ▶ Mergesort

# Mergesort in Scheme

- ▶ Direkt Umsetzung
- ▶ Hilfsfunktionen
  - ▶ `split` teilt eine einzelne Liste in eine Liste von zwei etwa gleichgroße Listen auf
  - ▶ `merge` mischt zwei sortierte Listen sortiert zusammen

```
(define (mergesort l)
 (if (or (null? l)
 (null? (cdr l)))
 l
 (let* ((res (split l))
 (l1 (mergesort (car res)))
 (l2 (mergesort (car (cdr res)))))
 (merge l1 l2))))
```

## Hilfsfunktion `merge`

- ▶ Ist eine der Listen leer, gib die andere zurück
- ▶ Ansonsten:
  - ▶ Extrahiere das kleinste Element, mische den Rest, hänge das kleinste vor das Ergebnis

```
(define (merge l1 l2)
 (cond ((null? l1)
 l2)
 ((null? l2)
 l1)
 ((< (car l1) (car l2))
 (cons (car l1) (merge (cdr l1) l2)))
 (else
 (cons (car l2) (merge l1 (cdr l2))))))
```

## Hilfsfunktion split

```
(define (split l)
 (cond ((null? l) ; Leeres l -> ('() '())
 (list '() '()))
 ((null? (cdr l)) ; Nur ein Element: ('() l)
 (list '() l))
 (else ; Sonst: Rest splitten
 (let* ((res (split (cdr (cdr l))))
 (l1 (car res)) ;; Erster Teil
 (l2 (car (cdr res))) ;; Zweiter Teil
)
 (list (cons (car l) l1)
 (cons (car (cdr l)) l2)))))))
```

Zur Vorlesung

- ▶ Stellen Sie Übung: Komplexität von Sortierverfahren fertig. Wenn Sie die Aufgabe auch quantitativ bearbeiten wollen:
  - ▶ Wenn Sie Guile verwenden, dann liefert die Funktion (`gettimeofday`) ein Paar von ganzen Zahlen, dessen `car` die Zahl der Sekunden seit 1970 ist, während das `cdr` die Microsekunden der angefangenen nächsten Sekunde enthält.
  - ▶ In Racket/DrScheme gibt die Funktion (`current-inexact-milliseconds`) die Anzahl der Milisekunden seit 1970 als Gleitkommazahl zurück, alternativ akzeptiert die *Special Form* (`time body`) einen beliebigen Ausdruck, evaluiert ihn, und gibt die Laufzeit auf dem aktuellen Ausgabeport aus.

- ▶ Rückblick/Wiederholung
- ▶ Hausaufgabe/Mergesort
- ▶ Rekursion abstrakt
- ▶ Input/Output
- ▶ Die Wahrheit<sup>TM</sup> über Listen

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

# Ziele Vorlesung 11

- ▶ Rückblick/Wiederholung
- ▶ Programmieraufgabe: Türme von Hanoi

# Ziele Vorlesung 11

- ▶ Rückblick/Wiederholung
- ▶ Programmieraufgabe: Türme von Hanoi

Es gibt jetzt einen Anhang mit einer Übersicht der von uns behandelten Scheme-Befehle.

- ▶ Rekursion als Fallunterscheidung
  - ▶ Einfache Fälle sofort
  - ▶ Komplexe Fälle zerlegen, kleinere Teilproblem rekursiv lösen, zusammensetzen
- ▶ Input/Output
  - ▶ `read/write`
  - ▶ Ports
  - ▶ `display/newline`
  - ▶ `read-char/write-char/peek-char`
- ▶ Kommandozeile und Programmende
- ▶ Listen als verkettete cons-Paare
- ▶ Assoc-Lists
- ▶ Hausaufgabe: Vergleich von Sortieralgorithmen

# Erinnerung: Hausaufgabe

- ▶ Stellen Sie Übung: Komplexität von Sortierverfahren fertig. Wenn Sie die Aufgabe auch quantitativ bearbeiten wollen:
  - ▶ Wenn Sie Guile verwenden, dann liefert die Funktion (`gettimeofday`) ein Paar von ganzen Zahlen, dessen `car` die Zahl der Sekunden seit 1970 ist, während das `cdr` die Microsekunden der angefangenen nächsten Sekunde enthält.
  - ▶ In Racket/DrScheme gibt die Funktion (`current-inexact-milliseconds`) die Anzahl der Milisekunden seit 1970 als Gleitkommazahl zurück, alternativ akzeptiert die *Special Form* (`time body`) einen beliebigen Ausdruck, evaluiert ihn, und gibt die Laufzeit auf dem aktuellen Ausgabeport aus.

# Diskussion: Komplexität von Sortierverfahren

Sorting list with 1000 elements

isort start: 0.025104045867919922 s

mergesort start: 0.00238800048828125 s

Sorting list with 2000 elements

isort start: 0.10665297508239746 s

mergesort start: 0.006776094436645508 s

Sorting list with 4000 elements

isort start: 0.4172329902648926 s

mergesort start: 0.012583017349243164 s

Sorting list with 8000 elements

isort start: 1.6808171272277832 s

mergesort start: 0.02836322784423828 s

Was heißt das?

Zur Vorlesung

- ▶ Stellen Sie die Übung: Türme von Hanoi fertig

- ▶ Rückblick/Wiederholung
- ▶ Programmieraufgabe: Türme von Hanoi

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

## Ziele Vorlesung 12

- ▶ Rückblick/Wiederholung
- ▶ Diskussion: Türme von Hanoi
- ▶ Funktionale Funktionen
- ▶ Zerstörung
- ▶ Typsystem in Scheme
- ▶ Lücken

## Ziele Vorlesung 12

- ▶ Rückblick/Wiederholung
- ▶ Diskussion: Türme von Hanoi
- ▶ Funktionale Funktionen
- ▶ Zerstörung
- ▶ Typsystem in Scheme
- ▶ Lücken

**Erinnerung: Am Dienstag, 3.2. (für 14B) und Mittwoch, 4.2. (für 14C) fällt die Vorlesung jeweils aus. Der Dozent muß was lernen...**

- ▶ Aufgabe: Türme von Hanoi

# Diskussion: Türme von Hanoi



Zur Vorlesung

- ▶ Stellen Sie die Übung: Höfliche Damen fertig.

- ▶ Rückblick/Wiederholung
- ▶ Diskussion: Türme von Hanoi
- ▶ Funktionale Funktionen
- ▶ Zerstörung
- ▶ Typsystem in Scheme
- ▶ Lücken

Haben Sie in den letzten 3 Wochen etwas gelernt?

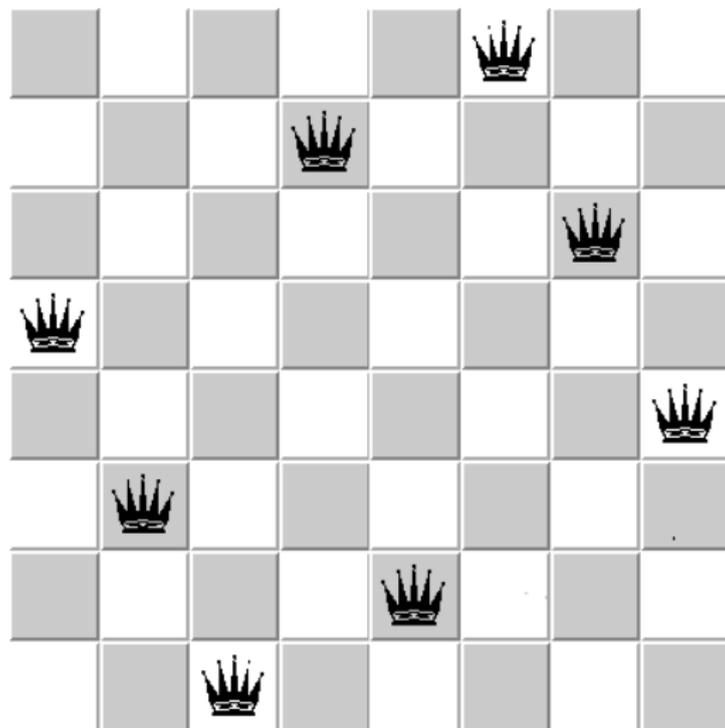
- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

- ▶ Rückblick/Wiederholung
- ▶ Hausaufgabe: Damen plazieren
- ▶ Aussagenlogik
  - ▶ Einführung
  - ▶ Syntax
  - ▶ Semantik
  - ▶ Formalisieren

# Rückblick/Wiederholung

- ▶ Funktionale Features
  - ▶ `lambda`, `map`, `apply`, `eval`
- ▶ Destruktive Funktionen:
  - ▶ `set!`, `set-car!`, `set-cdr!`
- ▶ Typsystem
- ▶ Lücken
  - ▶ Zahlen
  - ▶ Strings
  - ▶ Vektoren
  - ▶ Variadische Funktionen
  - ▶ Macros

# Diskussion: Damen



Zur Vorlesung

- ▶ Bearbeiten Sie Übung: Formalisierung von Raubüberfällen

- ▶ Rückblick/Wiederholung
- ▶ Hausaufgabe: Damen plazieren
- ▶ Aussagenlogik
  - ▶ Einführung
  - ▶ Syntax
  - ▶ Semantik
  - ▶ Formalisieren

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

- ▶ Rückblick/Wiederholung
- ▶ Unerfüllbarkeit
- ▶ Logisches Folgern
- ▶ Wahrheitstafeln
- ▶ Auf dem Weg zum Tableaux-Kalkül: Bäume
- ▶ Einführung: Analytische Tableaux

- ▶ Beispiel: Craig 1
- ▶ Aussagenlogik
  - ▶ Einführung
  - ▶ Syntax
  - ▶ Semantik (Interpretationen, Modelle)
  - ▶ Allgemeingültigkeit/Tautologie

## Erinnerung: Formalisierung von Raubüberfällen

Mr. McGregor, ein Londoner Ladeninhaber, rief bei Scotland Yard an und teilte mit, dass sein Laden ausgeraubt worden sei. Drei Verdächtige, A, B und C, wurden zum Verhör geholt. Folgende Tatbestände wurden ermittelt:

1. Jeder der Männer A, B und C war am Tag des Geschehens in dem Laden gewesen, und kein anderer hatte den Laden an dem Tag betreten.
  2. Wenn A schuldig ist, so hat er genau einen Komplizen.
  3. Wenn B unschuldig ist, so ist auch C unschuldig.
  4. Wenn genau zwei schuldig sind, dann ist A einer von ihnen.
  5. Wenn C unschuldig ist, so ist auch B unschuldig.
- ▶ Beschreiben Sie die Ermittlungsergebnisse als logische Formeln.
  - ▶ Lösen sie das Verbrechen!

nach R. Smullyan: "Wie heißt dieses Buch?"

## Lösung: Raubüberfälle (Formalisierung)



- ▶ Jeder der Männer A, B und C war am Tag des Geschehens in dem Laden gewesen, und kein anderer hatte den Laden an dem Tag betreten.
- ▶ Wenn A schuldig ist, so hat er genau einen Komplizen.
- ▶ Wenn B unschuldig ist, so ist auch C unschuldig.
- ▶ Wenn genau zwei schuldig sind, dann ist A einer von ihnen.
- ▶ Wenn C unschuldig ist, so ist auch B unschuldig.

## Lösung: Raubüberfälle (Formalisierung)



- ▶ Jeder der Männer A, B und C war am Tag des Geschehens in dem Laden gewesen, und kein anderer hatte den Laden an dem Tag betreten.
  - ▶  $A \vee B \vee C$
- ▶ Wenn A schuldig ist, so hat er genau einen Komplizen.
- ▶ Wenn B unschuldig ist, so ist auch C unschuldig.
- ▶ Wenn genau zwei schuldig sind, dann ist A einer von ihnen.
- ▶ Wenn C unschuldig ist, so ist auch B unschuldig.

## Lösung: Raubüberfälle (Formalisierung)



- ▶ Jeder der Männer A, B und C war am Tag des Geschehens in dem Laden gewesen, und kein anderer hatte den Laden an dem Tag betreten.
  - ▶  $A \vee B \vee C$
- ▶ Wenn A schuldig ist, so hat er genau einen Komplizen.
  - ▶  $A \rightarrow ((B \wedge \neg C) \vee (\neg B \wedge C))$
- ▶ Wenn B unschuldig ist, so ist auch C unschuldig.
- ▶ Wenn genau zwei schuldig sind, dann ist A einer von ihnen.
- ▶ Wenn C unschuldig ist, so ist auch B unschuldig.

# Lösung: Raubüberfälle (Formalisierung)



- ▶ Jeder der Männer A, B und C war am Tag des Geschehens in dem Laden gewesen, und kein anderer hatte den Laden an dem Tag betreten.
  - ▶  $A \vee B \vee C$
- ▶ Wenn A schuldig ist, so hat er genau einen Komplizen.
  - ▶  $A \rightarrow ((B \wedge \neg C) \vee (\neg B \wedge C))$
- ▶ Wenn B unschuldig ist, so ist auch C unschuldig.
  - ▶  $\neg B \rightarrow \neg C$
- ▶ Wenn genau zwei schuldig sind, dann ist A einer von ihnen.
- ▶ Wenn C unschuldig ist, so ist auch B unschuldig.

# Lösung: Raubüberfälle (Formalisierung)



- ▶ Jeder der Männer A, B und C war am Tag des Geschehens in dem Laden gewesen, und kein anderer hatte den Laden an dem Tag betreten.
  - ▶  $A \vee B \vee C$
- ▶ Wenn A schuldig ist, so hat er genau einen Komplizen.
  - ▶  $A \rightarrow ((B \wedge \neg C) \vee (\neg B \wedge C))$
- ▶ Wenn B unschuldig ist, so ist auch C unschuldig.
  - ▶  $\neg B \rightarrow \neg C$
- ▶ Wenn genau zwei schuldig sind, dann ist A einer von ihnen.
  - ▶  $\neg(B \wedge C \wedge \neg A)$
- ▶ Wenn C unschuldig ist, so ist auch B unschuldig.

# Lösung: Raubüberfälle (Formalisierung)



- ▶ Jeder der Männer A, B und C war am Tag des Geschehens in dem Laden gewesen, und kein anderer hatte den Laden an dem Tag betreten.
  - ▶  $A \vee B \vee C$
- ▶ Wenn A schuldig ist, so hat er genau einen Komplizen.
  - ▶  $A \rightarrow ((B \wedge \neg C) \vee (\neg B \wedge C))$
- ▶ Wenn B unschuldig ist, so ist auch C unschuldig.
  - ▶  $\neg B \rightarrow \neg C$
- ▶ Wenn genau zwei schuldig sind, dann ist A einer von ihnen.
  - ▶  $\neg(B \wedge C \wedge \neg A)$
- ▶ Wenn C unschuldig ist, so ist auch B unschuldig.
  - ▶  $\neg C \rightarrow \neg B$

# Lösung: Raubüberfälle (Analyse)



# Lösung: Raubüberfälle (Analyse)



| A | B | C | 1. | 2. | 3. | 4. | 5. |
|---|---|---|----|----|----|----|----|
| 0 | 0 | 0 | 0  | 1  | 1  | 1  | 1  |
| 0 | 0 | 1 | 1  | 1  | 0  | 1  | 1  |
| 0 | 1 | 0 | 1  | 1  | 1  | 1  | 0  |
| 0 | 1 | 1 | 1  | 1  | 1  | 0  | 1  |
| 1 | 0 | 0 | 1  | 0  | 1  | 1  | 1  |
| 1 | 0 | 1 | 1  | 1  | 0  | 1  | 1  |
| 1 | 1 | 0 | 1  | 1  | 1  | 1  | 0  |
| 1 | 1 | 1 | 1  | 0  | 1  | 1  | 1  |

# Lösung: Raubüberfälle (Analyse)



| A | B | C | 1. | 2. | 3. | 4. | 5. |
|---|---|---|----|----|----|----|----|
| 0 | 0 | 0 | 0  | 1  | 1  | 1  | 1  |
| 0 | 0 | 1 | 1  | 1  | 0  | 1  | 1  |
| 0 | 1 | 0 | 1  | 1  | 1  | 1  | 0  |
| 0 | 1 | 1 | 1  | 1  | 1  | 0  | 1  |
| 1 | 0 | 0 | 1  | 0  | 1  | 1  | 1  |
| 1 | 0 | 1 | 1  | 1  | 0  | 1  | 1  |
| 1 | 1 | 0 | 1  | 1  | 1  | 1  | 0  |
| 1 | 1 | 1 | 1  | 0  | 1  | 1  | 1  |

**Es gibt kein Modell der Aussagen**

# Lösung: Raubüberfälle (Analyse)



| A | B | C | 1. | 2. | 3. | 4. | 5. |
|---|---|---|----|----|----|----|----|
| 0 | 0 | 0 | 0  | 1  | 1  | 1  | 1  |
| 0 | 0 | 1 | 1  | 1  | 0  | 1  | 1  |
| 0 | 1 | 0 | 1  | 1  | 1  | 1  | 0  |
| 0 | 1 | 1 | 1  | 1  | 1  | 0  | 1  |
| 1 | 0 | 0 | 1  | 0  | 1  | 1  | 1  |
| 1 | 0 | 1 | 1  | 1  | 0  | 1  | 1  |
| 1 | 1 | 0 | 1  | 1  | 1  | 1  | 0  |
| 1 | 1 | 1 | 1  | 0  | 1  | 1  | 1  |

**Es gibt kein Modell der Aussagen**



**Versicherungsbetrug durch den Ladenbesitzer!**

Zur Vorlesung

- ▶ Bearbeiten Sie Übung: Aussagenlogik in Scheme

- ▶ Rückblick/Wiederholung
- ▶ Unerfüllbarkeit
- ▶ Logisches Folgern
- ▶ Wahrheitstafeln
- ▶ Auf dem Weg zum Tableaux-Kalkül: Bäume
- ▶ Einführung: Analytische Tableaux

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

- ▶ Rückblick/Wiederholung
- ▶ Hausaufgabe: Aussagenlogik in Scheme
- ▶ Tableaux-Kalkül für die Aussagenlogik
  - ▶ Üben!
  - ▶ Korrektheit und Vollständigkeit
  - ▶ Strategien
- ▶ Logische Äquivalenz

- ▶  $F$  ist allgemeingültig:  $I(F) = 1$  für alle  $I$
- ▶  $F$  ist unerfüllbar:  $I(F) = 0$  für alle  $I$
- ▶  $F$  allgemeingültig gdw.  $\neg F$  unerfüllbar (aber es gibt viele Formeln, die weder-noch sind!)
- ▶ Logisches Folgern
  - ▶  $KB \models F$ : Alle Modelle von  $KB$  sind auch Modelle von  $F$
  - ▶ Wahrheitstafelmethode
  - ▶ Deduktionstheorem:  $F_1, \dots, F_n \models G$  gdw.  $\models (F_1 \wedge \dots \wedge F_n) \rightarrow G$
- ▶ Bäume
  - ▶ Knoten, Kanten, Wurzel, Pfade, Äste
- ▶ Einführung: Tableaux-Kalkül für die Aussagenlogik
  - ▶  $\alpha$ - und  $\beta$ -Zerlegung



# Aussagenlogik in Scheme (1)

# Aussagenlogik in Scheme (1)

- ▶ Meine Version:
  - ▶ Interpretationen sind Scheme-Funktionen, die Atome auf Scheme-Wahrheitswerte abbilden
  - ▶ Formeln sind Scheme-Ausdrücke mit Operatoren: 'not, 'or, 'and, 'implies, 'equiv
- ▶ Beispielformel:  $a \rightarrow (b \wedge \neg c) \vee (\neg b \wedge c)$ 

```
(define g2
 '(implies a (or (and b (not c)) (and (not b) c))))
```
- ▶ Beispielinterpretation:

```
(define (make-var-interpretation true-vars)
 (lambda (x) (if (member x true-vars) #t #f)))

(define I1 (make-var-interpretation '(a b)))
(define I2 (make-var-interpretation '(a b c)))
```

## Aussagenlogik in Scheme(2)

```
(define (prop-eval form var-interpretation)
 (if (symbol? form)
 (var-interpretation form)
 (let ((op (car form))
 (args (map (lambda (x)
 (prop-eval x var-interpretation))
 (cdr form))))
 (cond ((equal? op 'not)
 (not (car args)))
 ((equal? op 'or)
 (or (car args) (cadr args)))
 ((equal? op 'and)
 (and (car args) (cadr args)))
 ((equal? op 'implies)
 (or (not (car args)) (cadr args)))
 ((equal? op 'equiv)
 (equal? (car args) (cadr args)))
 (else
 (display "Error_in_formula")
 (newline)))))))
```

## Aussagenlogik in Scheme(3)

```
> (define g2
 '(implies a (or (and b (not c)) (and (not b) c))))
```

```
> (define I1 (make-var-interpretation '(a b)))
```

```
> (define I2 (make-var-interpretation '(a b c)))
```

```
> (prop-eval g2 I1)
```

```
==> #t
```

```
> (prop-eval g2 I2)
```

```
==> #f
```

Zur Vorlesung

- ▶ Bearbeiten Sie Übung: Tableaux für Inspektor Craig

- ▶ Rückblick/Wiederholung
- ▶ Hausaufgabe: Aussagenlogik in Scheme
- ▶ Tableaux-Kalkül für die Aussagenlogik
  - ▶ Üben!
  - ▶ Korrektheit und Vollständigkeit
  - ▶ Strategien
- ▶ Logische Äquivalenz

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

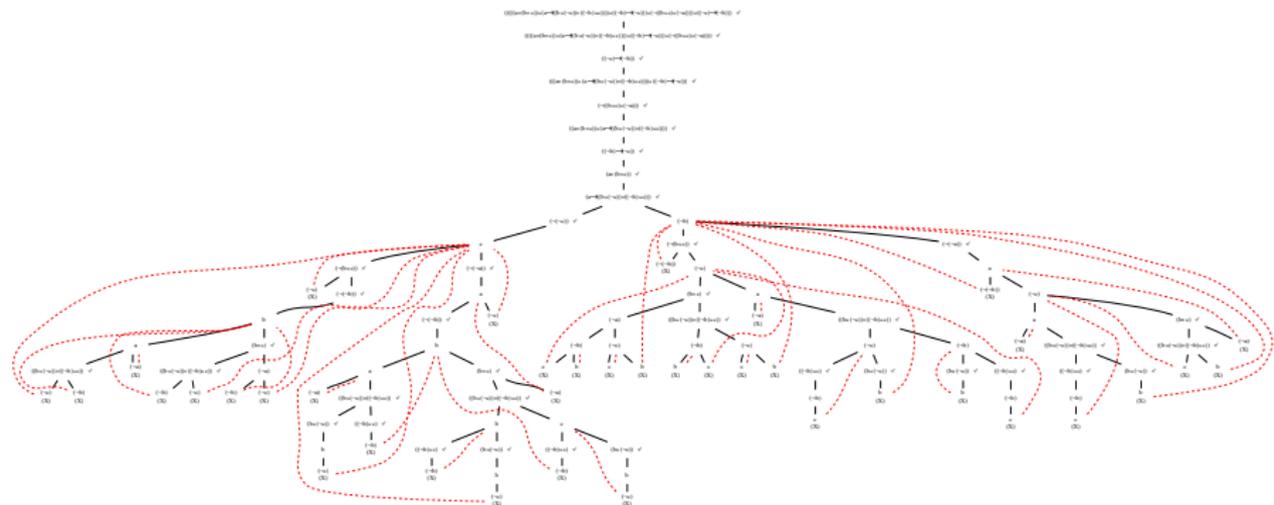
- ▶ Rückblick/Wiederholung
- ▶ Basen der Aussagenlogik
- ▶ Äquivalenzumformungen (kurz)
- ▶ Normalformen (NNF/KNF)

- ▶ Abschluß des Tableaux-Kalküls
  - ▶ Korrektheit und Vollständigkeit
  - ▶ Modellextraktion
  - ▶ Strategien
- ▶ Logische Äquivalenz
  - ▶ Zwei Formeln sind logisch äquivalent, wenn sie unter allen Interpretationen gleich ausgewertet werden

# Tableau: Craig 1



# Tableau: Craig 2



## **Definition (Basis der Aussagenlogik):**

Eine Menge von Operatoren  $O$  heißt eine **Basis** der Aussagenlogik, falls gilt: Zu jeder Formel  $F$  gibt es eine Formel  $G$  mit  $F \equiv G$ , und  $G$  verwendet nur Operatoren aus  $O$ .

- ▶ Das Konzept ermöglicht es, viele Aussagen zu Erfüllbarkeit, Folgerungen, und Äquivalenz auf einfachere Formelklassen zu beschränken.

## **Satz:**

- ▶  $\{\wedge, \vee, \neg\}$  ist eine Basis
- ▶  $\{\rightarrow, \neg\}$  ist eine Basis
- ▶  $\{\wedge, \neg\}$  ist eine Basis

# Beweisprinzip: Strukturelle Induktion

- ▶ Die **strukturelle Induktion** oder **Induktion über den Aufbau** kann verwendet werden, um Eigenschaften von rekursiv definierten Objekten zu zeigen.
- ▶ Idee:
  - ▶ Zeige die Eigenschaft für die elementaren Objekte.
  - ▶ Zeige die Eigenschaft für die zusammengesetzten Objekte unter der Annahme, dass die Teile bereits die Eigenschaft haben
- ▶ Konkret für aussagenlogische Formeln:
  - ▶ Basisfälle sind die Aussagenlogischen Variablen oder  $\top$ ,  $\perp$ .
  - ▶ Zusammengesetzte Formeln haben die Form  $\neg A$ ,  $A \vee B$ ,  $A \wedge B \dots$ , und wir können annehmen, dass  $A$  und  $B$  schon die gewünschte Eigenschaft haben.

- ▶ Erinnerung: Eine Menge von Operatoren  $O$  heißt eine **Basis** der Aussagenlogik, falls gilt: Zu jeder Formel  $F$  gibt es eine Formel  $G$  mit  $F \equiv G$ , und  $G$  verwendet nur Operatoren aus  $O$ .
- ▶ Zeigen Sie:
  - ▶  $\{\rightarrow, \neg\}$  ist eine Basis

## $\{\rightarrow, \neg\}$ ist eine Basis der Aussagenlogik (1)

Zu zeigen: Für jedes  $F \in For_{0\Sigma}$  existiert ein  $G \in For_{0\Sigma}$  so dass  $F \equiv G$  und  $G$  nur die Operatoren  $\neg, \rightarrow$  enthält.

## $\{\rightarrow, \neg\}$ ist eine Basis der Aussagenlogik (1)

Zu zeigen: Für jedes  $F \in For_{0\Sigma}$  existiert ein  $G \in For_{0\Sigma}$  so dass  $F \equiv G$  und  $G$  nur die Operatoren  $\neg, \rightarrow$  enthält.

Beweis per struktureller Induktion

## $\{\rightarrow, \neg\}$ ist eine Basis der Aussagenlogik (1)

Zu zeigen: Für jedes  $F \in For_{0\Sigma}$  existiert ein  $G \in For_{0\Sigma}$  so dass  $F \equiv G$  und  $G$  nur die Operatoren  $\neg, \rightarrow$  enthält.

Beweis per struktureller Induktion

**Induktionsanfang:** Sei  $F$  eine elementare Formel. Dann gilt einer der folgenden Fälle:

1.  $F \in \Sigma$ : Dann gilt:  $F$  enthält keine Operatoren. Also hat  $F$  bereits die geforderten Eigenschaften.
2.  $F = \top$ : Sei  $a \in \Sigma$  ein beliebiges Atom. Wähle  $G = a \rightarrow a$ . Dann gilt für jede Interpretation  $I$ :  $I(G) = 1$ . Also gilt auch  $G \equiv F$  und  $G$  enthält nur den Operator  $\rightarrow$ .
3.  $F = \perp$ : Analog mit  $G = \neg(a \rightarrow a)$ .

Damit sind alle Basisfälle abgedeckt und der Induktionsanfang ist gesichert.

## $\{\rightarrow, \neg\}$ ist eine Basis der Aussagenlogik (2)

**Induktionsvoraussetzung:** Die Behauptung gelte für beliebige  $A, B \in For_{0\Sigma}$  (also: es existieren  $A', B' \in For_{0\Sigma}$  mit  $A \equiv A'$ ,  $B \equiv B'$ , und  $A', B'$  enthalten nur die Operatoren  $\rightarrow, \neg$ ).

## $\{\rightarrow, \neg\}$ ist eine Basis der Aussagenlogik (2)

**Induktionsvoraussetzung:** Die Behauptung gelte für beliebige  $A, B \in For_{0\Sigma}$  (also: es existieren  $A', B' \in For_{0\Sigma}$  mit  $A \equiv A'$ ,  $B \equiv B'$ , und  $A', B'$  enthalten nur die Operatoren  $\rightarrow, \neg$ ).

**Induktionsschritt:** Sei  $F$  eine zusammengesetzte Formel. Dann gilt:  
 $F = (\neg A)$  oder  $F = (A \vee B)$  oder  $F = (A \wedge B)$  oder  
 $F = (A \rightarrow B)$  oder  $F = (A \leftrightarrow B)$ .

## $\{\rightarrow, \neg\}$ ist eine Basis der Aussagenlogik (2)

**Induktionsvoraussetzung:** Die Behauptung gelte für beliebige  $A, B \in \text{For}_{0\Sigma}$  (also: es existieren  $A', B' \in \text{For}_{0\Sigma}$  mit  $A \equiv A'$ ,  $B \equiv B'$ , und  $A', B'$  enthalten nur die Operatoren  $\rightarrow, \neg$ ).

**Induktionsschritt:** Sei  $F$  eine zusammengesetzte Formel. Dann gilt:  
 $F = (\neg A)$  oder  $F = (A \vee B)$  oder  $F = (A \wedge B)$  oder  
 $F = (A \rightarrow B)$  oder  $F = (A \leftrightarrow B)$ .

Fallunterscheidung:

1.  $F = (\neg A)$ : Nach IV gibt es ein  $A'$  mit  $A \equiv A'$ ,  $A'$  enthält nur die Operatoren  $\rightarrow, \neg$ . Betrachte  $G = (\neg A')$ . Dann gilt:  $I(G) = I(F)$  für alle Interpretationen, also  $G \equiv F$ .

## $\{\rightarrow, \neg\}$ ist eine Basis der Aussagenlogik (3)

2.  $F = (A \vee B)$ : Nach IV gibt es  $A'$  mit  $A \equiv A'$ ,  $B \equiv B'$ . Betrachte  $G = (\neg A' \rightarrow B')$ . Dann gilt  $I(G) = I(F)$  für alle Interpretationen (siehe Fallunterscheidung in folgender Tabelle):

| $A$ | $B$ | $F$ | $A'$ (IV) | $B'$ (IV) | $(\neg A')$ | $G$ |
|-----|-----|-----|-----------|-----------|-------------|-----|
| 0   | 0   | 0   | 0         | 0         | 1           | 0   |
| 0   | 1   | 1   | 0         | 1         | 1           | 1   |
| 1   | 0   | 1   | 1         | 0         | 0           | 1   |
| 1   | 1   | 1   | 1         | 1         | 0           | 1   |

Also:  $G \equiv F$ , und  $G$  enthält nach Konstruktion nur die Operatoren  $\rightarrow, \neg$ .

## $\{\rightarrow, \neg\}$ ist eine Basis der Aussagenlogik (4)

3.  $F = (A \wedge B)$ : Nach IV gibt es  $A'$  mit  $A \equiv A'$ ,  $B \equiv B'$ . Betrachte  $G = (\neg(A' \rightarrow \neg B'))$ . Dann gilt  $I(G) = I(F)$  für alle Interpretationen (siehe Fallunterscheidung in folgender Tabelle):

| $A$ | $B$ | $F$ | $A'$ (IV) | $B'$ (IV) | $G = (\neg(A' \rightarrow \neg B'))$ |
|-----|-----|-----|-----------|-----------|--------------------------------------|
| 0   | 0   | 0   | 0         | 0         | 0                                    |
| 0   | 1   | 0   | 0         | 1         | 0                                    |
| 1   | 0   | 0   | 1         | 0         | 0                                    |
| 1   | 1   | 1   | 1         | 1         | 1                                    |

Also:  $G \equiv F$ , und  $G$  enthält nach Konstruktion nur die Operatoren  $\rightarrow, \neg$ .

## $\{\rightarrow, \neg\}$ ist eine Basis der Aussagenlogik (5)

4.  $F = (A \rightarrow B)$ : Nach IV gibt es  $A'$  mit  $A \equiv A'$ ,  $B \equiv B'$ . Betrachte  $G = (A' \rightarrow B')$ . Dann gilt  $I(G) = I(F)$  für alle Interpretationen und  $G$  enthält nach Konstruktion nur die Operatoren  $\rightarrow, \neg$ .

## $\{\rightarrow, \neg\}$ ist eine Basis der Aussagenlogik (5)

4.  $F = (A \rightarrow B)$ : Nach IV gibt es  $A'$  mit  $A \equiv A'$ ,  $B \equiv B'$ . Betrachte  $G = (A' \rightarrow B')$ . Dann gilt  $I(G) = I(F)$  für alle Interpretationen und  $G$  enthält nach Konstruktion nur die Operatoren  $\rightarrow, \neg$ .
5.  $F = (A \leftrightarrow B)$ : Nach IV gibt es  $A'$  mit  $A \equiv A'$ ,  $B \equiv B'$ . Betrachte  $G = (\neg((A' \rightarrow B') \rightarrow (\neg(B' \rightarrow A'))))$ . Dann gilt (per Nachrechnen ;-))  $I(G) = I(F)$  für alle Interpretationen und  $G$  enthält nach Konstruktion nur die Operatoren  $\rightarrow, \neg$ .

## $\{\rightarrow, \neg\}$ ist eine Basis der Aussagenlogik (5)

4.  $F = (A \rightarrow B)$ : Nach IV gibt es  $A'$  mit  $A \equiv A'$ ,  $B \equiv B'$ . Betrachte  $G = (A' \rightarrow B')$ . Dann gilt  $I(G) = I(F)$  für alle Interpretationen und  $G$  enthält nach Konstruktion nur die Operatoren  $\rightarrow, \neg$ .
5.  $F = (A \leftrightarrow B)$ : Nach IV gibt es  $A'$  mit  $A \equiv A'$ ,  $B \equiv B'$ . Betrachte  $G = (\neg((A' \rightarrow B') \rightarrow (\neg(B' \rightarrow A'))))$ . Dann gilt (per Nachrechnen ;-))  $I(G) = I(F)$  für alle Interpretationen und  $G$  enthält nach Konstruktion nur die Operatoren  $\rightarrow, \neg$ .

Also: In allen Fällen können wir eine zu  $F$  äquivalente Formel angeben, die nur  $\rightarrow, \neg$  als Operatoren enthält. Damit gilt der IS, und damit die Behauptung.

q.e.d.

Zur Vorlesung

- ▶ Bearbeiten Sie Übung: KNF Transformation.

- ▶ Rückblick/Wiederholung
- ▶ Basen der Aussagenlogik
- ▶ Äquivalenzumformungen (kurz)
- ▶ Normalformen (NNF/KNF)

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

- ▶ Rückblick/Wiederholung
- ▶ Konjunktive/Klauselnormalform
- ▶ Resolution für Aussagenlogik
- ▶ Klausellogik erster Stufe
  - ▶ Unifikation
  - ▶ (Resolution)

- ▶ Strukturelle Induktion
- ▶ Basen der Aussagenlogik
- ▶ Äquivalenzumformungen
- ▶ NNF/KNF (Kurz)

## Zwei Schritte

1. Transformation in NNF

2. „Nach innen schieben“ von  $\vee$   
Verwende Distributivität von  $\vee$  über  $\wedge$

## Vier Schritte

### 1. Transformation in NNF

#### 1.1 Elimination von $\leftrightarrow$

Verwende  $A \leftrightarrow B \equiv (A \rightarrow B) \wedge (B \rightarrow A)$

#### 1.2 Elimination von $\rightarrow$

Verwende  $A \rightarrow B \equiv \neg A \vee B$

#### 1.3 „Nach innen schieben“ von $\neg$ , elimination $\top, \perp$

Verwende de-Morgans Regeln und  $\neg\neg A \equiv A$

Verwende  $\top/\perp$ -Regeln

### 2. „Nach innen schieben“ von $\vee$

Verwende Distributivität von  $\vee$  über  $\wedge$

## Lösung: KNF Transformation (Craig)

$$\begin{array}{l|l} (A \vee B \vee C) & (A \vee B \vee C) \\ \wedge(A \rightarrow ((B \wedge \neg C) \vee (\neg B \wedge C))) & \wedge (\neg A \vee \neg B \vee \neg C) \wedge (\neg A \vee B \vee C) \\ & \text{(siehe unten)} \\ \wedge(\neg B \rightarrow \neg C) & \wedge (B \vee \neg C) \\ \wedge\neg(B \wedge C \wedge \neg A) & \wedge (\neg B \vee \neg C \vee A) \\ \wedge(\neg C \rightarrow \neg B) & \wedge (C \vee \neg B) \end{array}$$

# Lösung: KNF Transformation (Craig)

$$\begin{array}{l|l} (A \vee B \vee C) & (A \vee B \vee C) \\ \wedge (A \rightarrow ((B \wedge \neg C) \vee (\neg B \wedge C))) & \wedge (\neg A \vee \neg B \vee \neg C) \wedge (\neg A \vee B \vee C) \\ & \text{(siehe unten)} \\ \wedge (\neg B \rightarrow \neg C) & \wedge (B \vee \neg C) \\ \wedge \neg(B \wedge C \wedge \neg A) & \wedge (\neg B \vee \neg C \vee A) \\ \wedge (\neg C \rightarrow \neg B) & \wedge (C \vee \neg B) \end{array}$$

$$\begin{aligned} & (A \rightarrow ((B \wedge \neg C) \vee (\neg B \wedge C))) \\ &= \neg A \vee ((B \wedge \neg C) \vee (\neg B \wedge C)) \\ &= \neg A \vee (((B \wedge \neg C) \vee \neg B) \wedge ((B \wedge \neg C) \vee C)) \\ &= \neg A \vee ((B \vee \neg B) \wedge (\neg C \vee \neg B) \wedge (B \vee C) \wedge (\neg C \vee C)) \\ &= \neg A \vee ((\neg C \vee \neg B) \wedge (B \vee C)) \\ &= (\neg A \vee \neg B \vee \neg C) \wedge (\neg A \vee B \vee C) \end{aligned}$$

# Lösung: KNF Transformation (Abstrakte Kunst)

$$\begin{aligned} & \neg(a \vee b) \leftrightarrow (b \wedge (a \rightarrow \neg c)) \\ = & (\neg(a \vee b) \rightarrow (b \wedge (a \rightarrow \neg c))) \wedge ((b \wedge (a \rightarrow \neg c)) \rightarrow \neg(a \vee b)) \\ = & (a \vee b \vee (b \wedge (\neg a \vee \neg c))) \wedge (\neg(b \wedge (\neg a \vee \neg c)) \vee \neg(a \vee b)) \\ = & (a \vee b \vee (b \wedge (\neg a \vee \neg c))) \wedge (\neg b \vee \neg(\neg a \vee \neg c) \vee (\neg a \wedge \neg b)) \\ = & (a \vee b \vee (b \wedge (\neg a \vee \neg c))) \wedge (\neg b \vee (a \wedge c) \vee (\neg a \wedge \neg b)) \text{ (NNF)} \\ = & (b \vee a \vee b) \wedge (\neg a \vee \neg c \vee a \vee b) \wedge (\neg a \vee a \vee \neg b) \\ & \wedge (\neg b \vee a \vee \neg b) \wedge (\neg a \vee c \vee \neg b) \wedge (\neg b \vee c \vee \neg b) \\ = & (b \vee a) \wedge (\neg b \vee a) \wedge (\neg a \vee c \vee \neg b) \wedge (\neg b \vee c) \end{aligned}$$

Zur Vorlesung

- ▶ Formeln der Prädikatenlogik werden aus Atomen mit den üblichen Operatoren und den **Quantoren**  $\forall, \exists$  gebaut
- ▶ Fakten:
  - ▶ Wir können prädikatenlogische Formeln in Klausellogik erster Stufe übersetzen
  - ▶ Wir können logische Folgerung in Prädikatenlogik auf die Unerfüllbarkeit von Klauselmengen zurückführen
- ▶ Jetzt: Klausellogik erster Stufe

- ▶ Bearbeiten Sie einige der Aufgaben aus der Aufgabensammlung.

- ▶ Rückblick/Wiederholung
- ▶ Konjunktive/Klauselnormalform
- ▶ Resolution für Aussagenlogik
- ▶ Klausellogik erster Stufe
  - ▶ Unifikation
  - ▶ Resolution

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

- ▶ Rückblick/Wiederholung
- ▶ Unifikation
- ▶ Übungen/Aufgaben

- ▶ Konjunktive/Klauselnormalform
- ▶ Resolution für Aussagenlogik
- ▶ Klausellogik erster Stufe
  - ▶ Syntax
    - ▶ Signatur:  $(P, F, V)$
    - ▶ Terme (aus  $F, V$ )
    - ▶ Atome (aus  $P$  und Termen)
    - ▶ Literale und Klauseln wie bei Aussagenlogik
    - ▶ Variablen sind all-quantifiziert!
  - ▶ Unifikation
  - ▶ (Resolution)

# Unifikation

Problem: Finde geeignete Ersetzungen für Variablen, so dass zwei Terme/Atome gleich werden.

# Unifikation

Problem: Finde geeignete Ersetzungen für Variablen, so dass zwei Terme/Atome gleich werden.

Andere Formulierung: Finde gemeinsame Instanzen von zwei Atomen!

# Unifikation

Problem: Finde geeignete Ersetzungen für Variablen, so dass zwei Terme/Atome gleich werden.

Andere Formulierung: Finde gemeinsame Instanzen von zwei Atomen!  
Also: Terme, für die beide Atome ein Aussage machen!

# Unifikation

Problem: Finde geeignete Ersetzungen für Variablen, so dass zwei Terme/Atome gleich werden.

Andere Formulierung: Finde gemeinsame Instanzen von zwei Atomen!  
Also: Terme, für die beide Atome ein Aussage machen!

## Definition (Unifikator):

- ▶ Seien  $s, t \in T_\Sigma$  zwei Terme
- ▶ Seien  $a, b \in A_\Sigma$  zwei Atome

Ein **Unifikator** für  $s, t$  bzw.  $a, b$  ist eine Substitution  $\sigma$  mit  $\sigma(s) = \sigma(t)$  bzw.  $\sigma(a) = \sigma(b)$

- ▶ Fakt: (Allgemeinste) Unifikatoren können systematisch gefunden werden

# Beobachtungen zum Finden von Unifikatoren

1.  $sterblich(X)$ ,  $mensch(X)$  können nie unifizieren
  - ▶ Das erste Symbol unterscheidet sich immer
2.  $X$ ,  $lehrer(X)$  können nie unifizieren
  - ▶ Egal, was für  $X$  eingesetzt wird, ein *lehrer* bleibt immer über
  - ▶ "Occurs-Check"
3.  $X$ ,  $lehrer(sokrates)$  unifizieren mit  $\sigma = \{X \leftarrow lehrer(sokrates)\}$ 
  - ▶ Variablen und die meisten Terme machen kein Problem
4.  $lauter(X, sokrates)$ ,  $lauter(aristoteles, Y)$  unifizieren mit  $\sigma = \{X \leftarrow aristoteles, Y \leftarrow sokrates\}$ 
  - ▶ Der Unifikator setzt sich aus den einzelnen Teilen zusammen

# Unifikation als paralleles Gleichungslösen

Fakt: Das Unifikationproblem wird **einfacher**, wenn man es für Mengen von Termpaaren betrachtet!

- ▶ Gegeben:  $R = \{s_1 = t_1, s_2 = t_2, \dots, s_n = t_n\}$ 
  - ▶ Suche **gemeinsamen Unifikator**  $\sigma$  mit
    - ▶  $\sigma(s_1) = \sigma(t_1)$
    - ▶  $\sigma(s_2) = \sigma(t_2)$
    - ▶ ...
    - ▶  $\sigma(s_n) = \sigma(t_n)$
- ▶ Verwende Transformationssystem
  - ▶ Zustand:  $R, \sigma$ 
    - ▶  $R$ : Menge von Termpaaren
    - ▶  $\sigma$ : Kandidat des Unifikators
  - ▶ Anfangszustand:  $\{s = t\}, \{\}$
  - ▶ Termination:  $\{\}, \sigma$

# Unifikation: Transformationssystem

$$\text{Binden: } \frac{\{x = t\} \cup R, \sigma}{\{x \leftarrow t\}(R), \{x \leftarrow t\} \circ \sigma} \text{ falls } x \notin \text{var}(t)$$

$$\text{Orientieren: } \frac{\{t = x\} \cup R, \sigma}{\{x = t\} \cup R, \sigma} \text{ falls } t \text{ keine Variable ist}$$

$$\text{Zerlegen: } \frac{\{f(s_1, \dots, s_n) = f(t_1, \dots, t_n)\} \cup R, \sigma}{\{s_1 = t_1, \dots, s_n = t_n\} \cup R, \sigma}$$

$$\text{Occurs: } \frac{\{x = t\} \cup R, \sigma}{\text{FAIL}} \text{ falls } x \in \text{var}(t)$$

$$\text{Konflikt: } \frac{\{f(s_1, \dots, s_n) = g(t_1, \dots, t_m)\} \cup R, \sigma}{\text{FAIL}} \text{ falls } f \neq g$$

# Beispiel

| $R$                                       | $\sigma$                                                                         | Regel          |
|-------------------------------------------|----------------------------------------------------------------------------------|----------------|
| $\{f(f(X, g(g(Y))), X) = f(f(Z, Z), U)\}$ | $\{\}$                                                                           | Zerlegen       |
| $\{f(X, g(g(Y))) = f(Z, Z), X = U\}$      | $\{\}$                                                                           | Binden ( $X$ ) |
| $\{f(U, g(g(Y))) = f(Z, Z)\}$             | $\{X \leftarrow U\}$                                                             | Zerlegen       |
| $\{Z = U, g(g(Y)) = Z\}$                  | $\{X \leftarrow U\}$                                                             | Binden ( $Z$ ) |
| $\{g(g(Y)) = U\}$                         | $\{X \leftarrow U,$<br>$Z \leftarrow U\}$                                        | Orientieren    |
| $\{U = g(g(Y))\}$                         | $\{X \leftarrow U,$<br>$Z \leftarrow U\}$                                        | Binden ( $U$ ) |
| $\{\}$                                    | $\{X \leftarrow g(g(Y)),$<br>$Z \leftarrow g(g(Y)),$<br>$U \leftarrow g(g(Y))\}$ |                |

- ▶ Unifikation
- ▶ Syntax für Logik erster Stufe
- ▶ *Quodlibet*

# Hausaufgabe



- ▶ Rückblick/Wiederholung
- ▶ Unifikation
- ▶ Übungen/Aufgaben

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

## Übungsklausur

- ▶ Gerne direkt wie immer
- ▶ Auch gerne anonym per Web-System

**Viel Glück für die Klausuren!**

Ende

## Material TINF14C

- ▶ Vorlesung 1
- ▶ Vorlesung 2
- ▶ Vorlesung 3
- ▶ Vorlesung 4
- ▶ Vorlesung 5
- ▶ Vorlesung 6
- ▶ Vorlesung 7
- ▶ Vorlesung 8
- ▶ Vorlesung 9
- ▶ Vorlesung 10
- ▶ Vorlesung 11
- ▶ Vorlesung 12
- ▶ Vorlesung 13
- ▶ Vorlesung 14
- ▶ Vorlesung 15
- ▶ Vorlesung 16
- ▶ Vorlesung 17
- ▶ Vorlesung 18
- ▶ Vorlesung 19
- ▶ Vorlesung 20
- ▶ Vorlesung 21

# Ziele Vorlesung 1

- ▶ Gegenseitiges Kennenlernen
- ▶ Praktische Informationen
- ▶ Übersicht und Motivation

- ▶ Ihre Erfahrungen
  - ▶ Informatik allgemein?
  - ▶ Programmieren? Sprachen?
- ▶ Ihre Erwartungen?
  - ▶ ...

- ▶ Ihre Erfahrungen
  - ▶ Informatik allgemein?
  - ▶ Programmieren? Sprachen?
- ▶ Ihre Erwartungen?
  - ▶ ...
- ▶ Feedbackrunde am Ende der Vorlesung

## ▶ Vorlesungszeiten

- ▶ Montags, 13:00-15:15
- ▶ Freitags, 9:00-11:15
- ▶ Kurze Pausen nach Bedarf
- ▶ 11 Wochen Vorlesung+Klausurwoche
- ▶ Weihnachtspause: 22.12.-4.1.

## ▶ Klausur

- ▶ KW10/2015 (2.-6.3.2015)
- ▶ Voraussichtlich 90 Minuten
- ▶ Genauer Termin wird vom Sekretariat koordiniert

## ▶ Webseite zur Vorlesung

- ▶ <http://www.lehre.dhbw-stuttgart.de/~sschulz/lgli2014.html>

- ▶ Gegenseitiges Kennenlernen
- ▶ Praktische Informationen
- ▶ Übersicht und Motivation

Image credit, when not otherwise specified: Wikipedia, OpenClipart

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

- ▶ Kurzer Rückblick
- ▶ Mathematische Grundbegriffe
- ▶ Grundlagen der Mengenlehre
  - ▶ (eine) Konstruktion der natürlichen Zahlen

- ▶ Praktische Themen
  - ▶ Rechner mit Scheme-Umgebung!
- ▶ Vorlesungsziel: Vokabular und Methoden
- ▶ Ein erstes Formales System: Das MIU-Rätsel
- ▶ Einführung Logik
  - ▶ Beispiel: Äquivalenz von Spezifikation im Bereich ATM/ATC

Zur Vorlesung

Bearbeiten Sie die Übung *Konstruktion der negativen Zahlen*

- ▶ Kurzer Rückblick
- ▶ Mathematische Grundbegriffe
- ▶ Grundlagen der Mengenlehre
  - ▶ (eine) Konstruktion der natürlichen Zahlen

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

- ▶ Kurzer Rückblick
- ▶ Grundlagen der Mengenlehre
  - ▶ Venn-Diagramme
  - ▶ Mengenoperationen
  - ▶ Potenzmengen

- ▶ Grundbegriffe
  - ▶ Definition
  - ▶ Beweis
- ▶ Mengen
  - ▶ Definition per Aufzählung ( $\{1, 2, 4, 8, \dots\}$ )
  - ▶ Beschreibend ( $\{x \mid x = 2^n, n \in \mathbb{N}\}$  oder  $\{2^n \mid n \in \mathbb{N}\}$ )
  - ▶ Mengeneigenschaften
    - ▶ Ungeordnet (Gegensatz: z.B. Liste)
    - ▶ Keine Duplikate (Stichwort *Multimenge*)
  - ▶ Teilmengen ( $\subseteq$  vs.  $\subset$ ) und Obermengen
  - ▶ Mengengleichheit
  - ▶ Konkrete Mengen:  $\emptyset, \mathbb{N}, \mathbb{N}^+, \mathbb{Z}, \mathbb{Q}, \mathbb{R}$
- ▶ Konstruktion der natürlichen Zahlen
  - ▶ Zahlen repräsentiert als  $\emptyset, \{\emptyset\}, \{\{\emptyset\}\} \dots$  oder äquivalent  $0, s(0), s(s(0)) \dots$
  - ▶ Rekursive Definition von  $a$  (Addition) und  $m$  (Multiplikation)

# Rückblick/Hausaufgabe (1) TINF14C

- ▶ Idee: Wir interpretieren **Mengen** oder **Terme** als Zahlen

| Mengenschreibweise          | Term            | Zahl |
|-----------------------------|-----------------|------|
| $\emptyset$ oder $\{\}$     | 0               | 0    |
| $\{\emptyset\}$             | $s(0)$          | 1    |
| $\{\{\emptyset\}\}$         | $s(s(0))$       | 2    |
| $\{\{\{\emptyset\}\}\}$     | $s(s(s(0)))$    | 3    |
| $\{\{\{\{\emptyset\}\}\}\}$ | $s(s(s(s(0))))$ | 4    |
| ...                         | ...             | ...  |

- ▶ Wir definieren **rekursive** Rechenregeln rein syntaktisch:

- ▶ Addition ( $a$ ):

- ▶  $a(X, 0) = X$
- ▶  $a(X, s(Y)) = s(a(X, Y))$

- ▶ Multiplikation ( $m$ ):

- ▶  $m(X, 0) = 0$
- ▶  $m(X, s(Y)) = a(X, m(X, Y))$

► Addition ( $a$ ):

$$(1) a(X, 0) = X$$

$$(2) a(X, s(Y)) = s(a(X, Y))$$

► Multiplikation ( $m$ ):

$$(3) m(X, 0) = 0$$

$$(4) m(X, s(Y)) = a(X, m(X, Y))$$

► Beispielrechnung:  $2 \times 2$ :

$$\begin{aligned} & m(s(s(0)), s(s(0))) \\ = & a(s(s(0)), \underline{m(s(s(0)), s(0))}) && (4) \text{ mit } X = s(s(0)), Y = s(0) \\ = & a(s(s(0)), \underline{a(s(s(0)), \underline{m(s(s(0)), 0})})} && (3) \text{ mit } X = s(s(0)), Y = 0 \\ = & a(s(s(0)), \underline{a(s(s(0)), 0)}) && (3) \text{ mit } X = s(s(0)) \\ = & a(s(s(0)), s(s(0))) && (1) \text{ mit } X = s(s(0)) \\ = & \underline{s(a(s(s(0)), s(0)))} && (2) \text{ mit } X = s(s(0)), Y = s(0) \\ = & \underline{s(s(a(s(s(0)), 0))} && (2) \text{ mit } X = s(s(0)), Y = 0 \\ = & s(s(s(s(0)))) && (1) \text{ mit } X = s(s(0)), Y = 0 \end{aligned}$$

## Rückblick/Hausaufgabe (3) TINF14C

- ▶ Erweiterung auf negative Zahlen:

▶ Idee:  $p(X) = X - 1$ ,  $n(X) = -X$ ,  $v(X, Y) = X - Y$

$$n(0) = 0$$

$$p(n(X)) = n(s(X)) \quad n(p(X)) = s(n(X))$$

$$p(s(X)) = X \quad s(p(X)) = X$$

$$a(X, 0) = X \quad a(X, s(Y)) = s(a(X, Y))$$

$$v(X, 0) = X \quad v(X, s(Y)) = p(v(X, Y))$$

$$a(X, n(Y)) = v(X, Y)$$

$$m(X, 0) = 0 \quad m(X, s(Y)) = a(X, m(X, Y))$$

$$m(X, n(Y)) = n(m(X, Y))$$

- ▶ Kurzer Rückblick
- ▶ Grundlagen der Mengenlehre
  - ▶ Venn-Diagramme
  - ▶ Mengenoperationen
  - ▶ Potenzmengen und kartesische Produkte

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

- ▶ Rückblick und Wiederholung
- ▶ Relationen und ihre Eigenschaften

- ▶ Venn-Diagramme
- ▶ Mengenoperationen:  $\cup, \cap, \setminus, \overline{\phantom{x}}$  (Komplement),  $\Delta$
- ▶ Übung zu Mengenoperationen
- ▶ Mengenalgebra
- ▶ Kartesisches Produkt und Potenzmenge

# Diskussion: Übung Mengenoperationen

- ▶ 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.
  - ▶  $M_1 \cup M_2$ 
    - ▶ Die Menge der ungeraden Zahlen und der Vielfachen von 3
    - ▶  $M_1 \cup M_2 = \{x \mid \exists i \in \mathbb{N} : x = 3i \text{ oder } x = 2i + 1\}$
  - ▶  $M_1 \cap M_2$ 
    - ▶ Die Menge der ungeraden Vielfachen von 3
    - ▶  $M_1 \cap M_2 = \{6i + 3 \mid i \in \mathbb{N}\}$
  - ▶  $M_1 \setminus M_2$ 
    - ▶ Die Menge der geraden Vielfachen von 3
    - ▶  $M_1 \setminus M_2 = \{6i \mid i \in \mathbb{N}\}$
  - ▶  $M_1 \setminus \overline{M_2}$ 
    - ▶ Siehe  $M_1 \cap M_2$
  - ▶  $M_1 \triangle M_2$ 
    - ▶ Die Menge der geraden Vielfachen von 3 und der ungeraden Zahlen, die nicht durch 3 teilbar sind
    - ▶  $M_1 \triangle M_2 = \{6i \mid i \in \mathbb{N}\} \cup \{6i + k \mid i \in \mathbb{N}, k \in \{1, 5\}\}$   
 $= \{6i + k \mid i \in \mathbb{N}, k \in \{0, 1, 5\}\}$

- ▶ Rückblick und Wiederholung
- ▶ Relationen und ihre Eigenschaften

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

- ▶ Rückblick
- ▶ Relationen (Teil 2)
- ▶ Funktionen
- ▶ Kardinalität

- ▶ Mengenalgebra
- ▶ Kartesisches Produkt
- ▶ Relationen:  $R \subseteq M_1 \times \dots \times M_n$
- ▶ Spezieller Relationen
  - ▶ Binäre Relationen:  $R \subseteq M_1 \times M_2$
  - ▶ Homogene Relationen:  $R \subseteq M^n$
  - ▶ Binäre homogene Relationen:  $R \subseteq M^2$
- ▶ Relationseigenschaften:
  - ▶ Reflexivität:  $\forall x \in M : (x, x) \in R$
  - ▶ Symmetrie:  $\forall x, y \in M : (x, y) \in R \rightsquigarrow (y, x) \in R$
  - ▶ Transitivität:  $\forall x, y, z \in M : (x, y) \in R \text{ and } (y, z) \in R \rightsquigarrow (x, z) \in R$
  - ▶ Linkstotal:  $\forall x \in M \exists y \in M : (x, y) \in R$
  - ▶ Rechtseindeutig:  $\forall x, y, z \in M : (x, y) \in R \text{ and } (x, z) \in R \rightsquigarrow y = z$
- ▶ Äquivalenzrelationen

Bearbeiten Sie die Übung *Kardinalität*.

- ▶ Rückblick
- ▶ Relationen (Teil 2)
- ▶ Funktionen
- ▶ Kardinalität

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

- ▶ Einführung in Scheme
- ▶ “Hello World” - Starten und Ausführen von Programmen
- ▶ Interaktives Arbeiten
- ▶ Einfache Rekursion
- ▶ Listenverarbeitung

Zur Vorlesung

Bearbeiten Sie Übung: *Mengenlehre in Scheme*.

- ▶ Einführung in Scheme
- ▶ “Hello World” - Starten und Ausführen von Programmen
- ▶ Interaktives Arbeiten
- ▶ Einfache Rekursion
- ▶ Listenverarbeitung

# Feedback

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

# Feedback

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?



Schöne Pause!

- ▶ Rückblick/Wiederholung
- ▶ Hausaufgabe (Mengenlehre in Scheme)
- ▶ Funktionale Programmierung und Seiteneffekte
- ▶ Speichermodell, Variablen, Werte, Umgebungen
- ▶ Wichtige *Special Forms*
  - ▶ Sequenzen, `begin`
  - ▶ Konditionale: `cond`, `and`, `or`
  - ▶ `let`, `let*`

# Rückblick (1)

- ▶ Hintergrund/Geschichte/Anwendungen
- ▶ Scheme Programme ausführen
  - ▶ Von der Komandozeile (z.B. `guile-2.0 file.scm`)
  - ▶ Interaktiv (read-eval-print loop)
  - ▶ Laden von Programmen in den Interpreter (z.B. `(load file.scm)`)
- ▶ Syntax (*s-expressions*)
  - ▶ Atome (Zahlen, Symbole, Strings, ...)
  - ▶ Listen: `(a1 a2 a3)`
- ▶ Beispiel *Fakultät*

```
;; Factorial
(define (fak x)
 (if (= x 0)
 1
 (* x (fak (- x 1)))))
)
```

## Rückblick (2)

### ▶ Semantik

- ▶ Strings, Zahlen, ... haben natürlichen Wert
- ▶ Symbole haben nur dann einen Wert, wenn der (z.B. per *define*) festgelegt wird (sonst Fehler)
- ▶ Listenauswertung (normal)
  - ▶ Werte alle Teilausdrücke (wenn nötig rekursiv) aus
  - ▶ Interpretiere den Wert des ersten Ausdrucks als Funktion, wende diese auf die Werte der anderen Ausdrücke an
- ▶ *Special forms*
  - ▶ Z.B. (if *tst expr<sub>1</sub> expr<sub>2</sub>*)
  - ▶ (define pi 3.1415)
  - ▶ (define (plus3 x) (+ x 3))

### ▶ Datentypen und definierte Funktionen

- ▶ equal?
- ▶ Zahlen (=, >, +, -, \*, /)
- ▶ Listen '(), car, cdr, cons, append, null?, list)

# Übung: Mengenlehre in Scheme

- ▶ Repräsentieren Sie im folgenden Mengen als Listen
- ▶ Erstellen Sie Scheme-Funktionen für die folgenden Mengen-Operationen:
  - ▶ Einfügen:
    - ▶ `(insert 4 '(1 2 3)) ==> (1 2 3 4)` (Reihenfolge egal)
    - ▶ `(insert 2 '(1 2 3)) ==> (1 2 3)`
  - ▶ Vereinigung:
    - ▶ `(union '(1 2 3) '(3 4 5)) ==> (1 2 3 4 5)`
  - ▶ Schnittmenge:
    - ▶ `(intersection '(1 2 3) '(3 4 5)) ==> (3)`
  - ▶ Kartesisches Produkt:
    - ▶ `(kart '(1 2 3) '(a b c)) ==> ((1 a) (2 a) (3 a) (1 b) (2 b) (3 b) (1 c) (2 c) (3 c))`
  - ▶ Potenzmenge:
    - ▶ `(powerset '(1 2 3)) ==> (() (3) (2) (2 3) (1) (1 3) (1 2) (1 2 3))`
- ▶ Bonus: Implementieren Sie eine Funktion, die die Verkettung von zwei Relationen realisieren!

## Diskussion: Mengenlehre in Scheme (1)

```
(define (is-element? x set)
 (if (null? set)
 #f
 (if (equal? x (car set))
 #t
 (is-element? x (cdr set)))))
```

```
(define (set-insert x set)
 (if (is-element? x set)
 set
 (cons x set)))
```

```
(define (set-union set1 set2)
 (if (null? set1)
 set2
 (set-insert (car set1)
 (set-union (cdr set1) set2))))
```

## Diskussion: Mengenlehre in Scheme (2)

```
(define (set-intersection set1 set2)
 (if (null? set1)
 set1
 (if (is-element? (car set1) set2)
 (cons (car set1)
 (set-intersection (cdr set1) set2))
 (set-intersection (cdr set1) set2))))
```

```
(define (make-pairs m x)
 (if (null? m)
 m
 (cons (list (car m) x)
 (make-pairs (cdr m) x))))
```

```
(define (kart m1 m2)
 (if (null? m2)
 m2
 (append (make-pairs m1 (car m2))
 (kart m1 (cdr m2)))))
```

## Diskussion: Mengenlehre in Scheme (3)

```
(define (add-to-sets sets element)
 (if (null? sets)
 sets
 (cons (cons element (car sets))
 (add-to-sets (cdr sets) element))))
```

```
(define (powerset set)
 (if (null? set)
 (list set)
 (append (powerset (cdr set))
 (add-to-sets (powerset (cdr set))
 (car set)))))
```

Zur Vorlesung

- ▶ Bearbeiten Sie Übung: Fibonacci und Potenzmenge

- ▶ Rückblick/Wiederholung
- ▶ Hausaufgabe (Mengenlehre in Scheme)
- ▶ Funktionale Programmierung und Seiteneffekte
- ▶ Speichermodell, Variablen, Werte, Umgebungen
- ▶ Wichtige *Special Forms*
  - ▶ Sequenzen, `begin`
  - ▶ Konditionale: `cond`, `and`, `or`
  - ▶ `let`, `let*`

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

- ▶ Rückblick und Wiederholung (kurz!)
- ▶ Sortieren durch Einfügen
- ▶ Generische Programmierung
- ▶ Vorstellung Mergesort

- ▶ Auswertung vs. Seiteneffekte
- ▶ Variablen, Umgebungen
- ▶ Scheme-Konstrukte:
  - ▶ Sequenzen und `begin`
  - ▶ `cond`
  - ▶ `and`, `or`, `not`
  - ▶ `let`, `let*`

# Hausaufgabe (Fibonacci)

## Fibonacci mit if

```
(define (fib x)
 (if (= x 0)
 0
 (if (= x 1)
 1
 (+ (fib (- x 1))
 (fib (- x 2)))))))
```

## Fibonacci mit cond

```
(define (fib2 x)
 (cond ((= x 0)
 0)
 ((= x 1)
 1)
 (else
 (+ (fib2 (- x 1))
 (fib2 (- x 2)))))))
```

## Hausaufgabe: Potenzmenge (Hilfsfunktion v1)

Hilfsfunktion: add-to-sets

```
;; Eingabe:
;; sets: Eine Liste von Listen (interpretiert als
;; Menge von Mengen)
;; element: Ein einzelnes Element
;; Ausgabe:
;; Liste von Listen, wobei jede der Originalliste
;; um das einzelne Element erweitert wird
```

```
(define (add-to-sets sets element)
 (if (null? sets)
 sets
 (cons (cons element (car sets))
 (add-to-sets (cdr sets) element))))
```

## Hausaufgabe: Potenzmenge (Hilfsfunktion v2)

Hilfsfunktion: add-to-sets

*;; Eingabe:*

*;; sets: Eine Liste von Listen (interpretiert als*

*;; Menge von Mengen)*

*;; element: Ein einzelnes Element*

*;; Ausgabe:*

*;; Liste von Listen, wobei jede der Originalliste*

*;; um das einzelne Element erweitert wird*

```
(define (add-to-sets sets element)
 (if (null? sets)
 sets
 (let* ((first-set (car sets))
 (rest-sets (cdr sets))
 (first-new (cons element first-set))
 (rest-new (add-to-sets rest-sets element)))
 (cons first-new rest-new))))
```

## Hausaufgabe: Potenzmenge (Hilfsfunktion Beispiele)

```
> (add-to-sets '(() (1) (1 2) (1 2 3)) 5)
```

```
⇒ ((5) (5 1) (5 1 2) (5 1 2 3))
```

```
> (add-to-sets '((1 2) (2 3) (3 4)) 5)
```

## Hausaufgabe: Potenzmenge (Hilfsfunktion Beispiele)

> (add-to-sets '(() (1) (1 2) (1 2 3)) 5)

⇒ ((5) (5 1) (5 1 2) (5 1 2 3))

> (add-to-sets '((1 2) (2 3) (3 4)) 5)

⇒ ((5 1 2) (5 2 3) (5 3 4))

> (add-to-sets '(() ) 5)

## Hausaufgabe: Potenzmenge (Hilfsfunktion Beispiele)

> (add-to-sets '(() (1) (1 2) (1 2 3)) 5)

⇒ ((5) (5 1) (5 1 2) (5 1 2 3))

> (add-to-sets '((1 2) (2 3) (3 4)) 5)

⇒ ((5 1 2) (5 2 3) (5 3 4))

> (add-to-sets '(() ) 5)

⇒ ((5))

> (add-to-sets '() 5)

## Hausaufgabe: Potenzmenge (Hilfsfunktion Beispiele)

```
> (add-to-sets '(() (1) (1 2) (1 2 3)) 5)
```

```
⇒ ((5) (5 1) (5 1 2) (5 1 2 3))
```

```
> (add-to-sets '((1 2) (2 3) (3 4)) 5)
```

```
⇒ ((5 1 2) (5 2 3) (5 3 4))
```

```
> (add-to-sets '(()) 5)
```

```
⇒ ((5))
```

```
> (add-to-sets '() 5)
```

```
⇒ ()
```

# Hausaufgabe: Potenzmenge

```
(define (powerset set)
 (if (null? set)
 (list set)
 (append (powerset (cdr set))
 (add-to-sets (powerset (cdr set))
 (car set)))))

;; Berechne Potenzmenge von l
(define (powerset2 l)
 (cond ((null? l)
 (list l))
 (else
 (let* ((sub-ps (powerset2 (cdr l)))
 (new (car l))
 (sub-new (add-to-sets sub-ps new))
 (result (append sub-ps sub-new)))
 result))))
```

- ▶ Bearbeiten Sie die Übung: Mergesort

- ▶ Rückblick und Wiederholung (kurz!)
- ▶ Sortieren durch Einfügen
- ▶ Generische Programmierung
- ▶ Vorstellung Mergesort

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

- ▶ Hausaufgabe/Mergesort
- ▶ Funktionen und Rekursion
- ▶ Input/Output

- ▶ Mengenlehre in Scheme
- ▶ Sortieren durch Einfügen
- ▶ ... in Scheme
- ▶ Sortieren mit Vergleichsfunktion als Parameter
- ▶ Vorstellung Mergesort
  - ▶ Teile Sortierproblem in zwei kleinere Probleme
  - ▶ Sortiere diese rekursiv
  - ▶ Füge die sortierten Teillisten sortiert zusammen

# Mergesort in Scheme

- ▶ Direkt Umsetzung
- ▶ Hilfsfunktionen
  - ▶ `split` teilt eine einzelne Liste in eine Liste von zwei etwa gleichgroße Listen auf
  - ▶ `merge` mischt zwei sortierte Listen sortiert zusammen

```
(define (mergesort l)
 (if (or (null? l)
 (null? (cdr l)))
 l
 (let* ((res (split l))
 (l1 (mergesort (car res)))
 (l2 (mergesort (car (cdr res))))))
 (merge l1 l2))))
```

## Hilfsfunktion `merge`

- ▶ Ist eine der Listen leer, gib die andere zurück
- ▶ Ansonsten:
  - ▶ Extrahiere das kleinste Element, mische den Rest, hänge das kleinste vor das Ergebnis

```
(define (merge l1 l2)
 (cond ((null? l1)
 l2)
 ((null? l2)
 l1)
 ((< (car l1) (car l2))
 (cons (car l1) (merge (cdr l1) l2)))
 (else
 (cons (car l2) (merge l1 (cdr l2))))))
```

# Hilfsfunktion split

```
(define (split l)
 (cond ((null? l) ; Leeres l -> ('() '())
 (list '() '()))
 ((null? (cdr l)) ; Nur ein Element: ('() l)
 (list '() l))
 (else ; Sonst: Rest splitten
 (let* ((res (split (cdr (cdr l))))
 (l1 (car res)) ;; Erster Teil
 (l2 (car (cdr res))) ;; Zweiter Teil
)
 (list (cons (car l) l1)
 (cons (car (cdr l)) l2)))))))
```

Zur Vorlesung

- ▶ Stellen Sie die Übung: Türme von Hanoi fertig

- ▶ Hausaufgabe/Mergesort
- ▶ Funktionen und Rekursion
- ▶ Input/Output

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

- ▶ Rückblick/Wiederholung
- ▶ Hausaufgabe: Vergleich von Sortierverfahren
- ▶ Die Wahrheit<sup>TM</sup> über Listen
- ▶ Ein nicht-triviales Beispiel: Türme von Hanoi

- ▶ Semantik von Funktionsaufrufen
- ▶ Rekursion im allgemeinen
- ▶ Input/Output
  - ▶ `read/write`
  - ▶ `Ports`
  - ▶ `display/newline`
  - ▶ `read-char/write-char/peek-char`
- ▶ Hausaufgabe: Vergleich von Sortieralgorithmen

# Diskussion: Komplexität von Sortierverfahren

Sorting list with 1000 elements

isort start: 0.025104045867919922 s

mergesort start: 0.00238800048828125 s

Sorting list with 2000 elements

isort start: 0.10665297508239746 s

mergesort start: 0.006776094436645508 s

Sorting list with 4000 elements

isort start: 0.4172329902648926 s

mergesort start: 0.012583017349243164 s

Sorting list with 8000 elements

isort start: 1.6808171272277832 s

mergesort start: 0.02836322784423828 s

Was heißt das?

Zur Vorlesung

- ▶ Stellen Sie die Übung: Türme von Hanoi fertig

- ▶ Rückblick/Wiederholung
- ▶ Hausaufgabe: Vergleich von Sortierverfahren
- ▶ Die Wahrheit<sup>TM</sup> über Listen
- ▶ Ein nicht-triviales Beispiel: Türme von Hanoi

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

# Ziele Vorlesung 11

- ▶ Rückblick/Wiederholung
- ▶ Diskussion: Türme von Hanoi
- ▶ Funktionale Funktionen
- ▶ Zerstörung
- ▶ Lücken

- ▶ Listenstruktur
- ▶ Listenbefehle
- ▶ Aufgabe: Türme von Hanoi

# Diskussion: Türme von Hanoi



Zur Vorlesung

- ▶ Stellen Sie die Übung: Höfliche Damen fertig.

- ▶ Rückblick/Wiederholung
- ▶ Diskussion: Türme von Hanoi
- ▶ Funktionale Funktionen
- ▶ Zerstörung
- ▶ Lücken

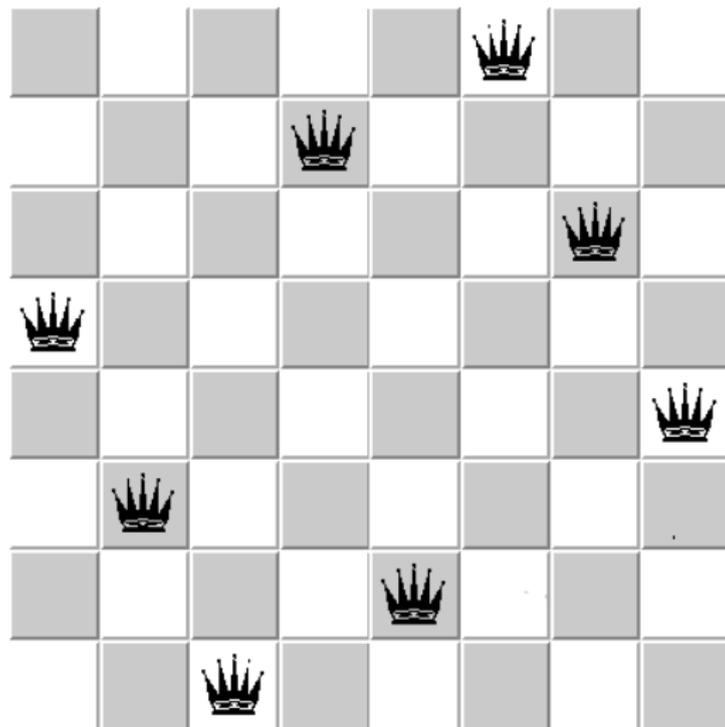
Haben Sie in den letzten 3 Wochen etwas gelernt?

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

- ▶ Rückblick/Wiederholung
- ▶ Hausaufgabe: Damen plazieren
- ▶ Aussagenlogik
  - ▶ Einführung
  - ▶ Syntax
  - ▶ Semantik
  - ▶ Formalisieren

- ▶ Diskussion: Türme von Hanoi (mal zwei)
- ▶ Funktionale Features
  - ▶ `lambda`, `map`, `apply`, `eval`
- ▶ Destruktive Funktionen:
  - ▶ `set!`, `set-car!`, `set-cdr!`
- ▶ Lücken
  - ▶ Zahlen
  - ▶ Strings
  - ▶ Vektoren
  - ▶ Variadische Funktionen
  - ▶ Macros

# Diskussion: Damen



Zur Vorlesung

- ▶ Bearbeiten Sie Übung: Formalisierung von Raubüberfällen

- ▶ Rückblick/Wiederholung
- ▶ Hausaufgabe: Damen plazieren
- ▶ Aussagenlogik
  - ▶ Einführung
  - ▶ Syntax
  - ▶ Semantik
  - ▶ Formalisieren

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

- ▶ Rückblick/Wiederholung
- ▶ Allgemeingültigkeit und Unerfüllbarkeit
- ▶ Logisches Folgern
- ▶ Wahrheitstafeln
- ▶ Bäume

- ▶ Inspektor Craig
- ▶ Logik: Syntax/Semantik/Kalkül
- ▶ Syntax der Aussagenlogik
  - ▶ Atomare Aussagen (“Variablen”)
  - ▶ Operatoren:  $\top$ ,  $\perp$ ,  $\neg$ ,  $\wedge$ ,  $\vee$ ,  $\rightarrow$ ,  $\leftrightarrow$
  - ▶ Klammern (und wie man sie vermeidet)
    - ▶ Präzedenz und Assoziativität
- ▶ Semantik der Aussagenlogik
  - ▶ Interpretationen auf Atomen
  - ▶ Fortsetzung auf Formeln
  - ▶ Modellbegriff

- ▶ Inspektor Craig
- ▶ Logik: Syntax/Semantik/Kalkül
- ▶ Syntax der Aussagenlogik
  - ▶ Atomare Aussagen (“Variablen”)
  - ▶ Operatoren:  $\top$ ,  $\perp$ ,  $\neg$ ,  $\wedge$ ,  $\vee$ ,  $\rightarrow$ ,  $\leftrightarrow$
  - ▶ Klammern (und wie man sie vermeidet)
    - ▶ Präzedenz und Assoziativität
- ▶ Semantik der Aussagenlogik
  - ▶ Interpretationen auf Atomen
  - ▶ Fortsetzung auf Formeln
  - ▶ Modellbegriff

Es gibt jetzt einen Anhang mit einer Übersicht der von uns behandelten Scheme-Befehle.

## Erinnerung: Formalisierung von Raubüberfällen

Mr. McGregor, ein Londoner Ladeninhaber, rief bei Scotland Yard an und teilte mit, dass sein Laden ausgeraubt worden sei. Drei Verdächtige, A, B und C, wurden zum Verhör geholt. Folgende Tatbestände wurden ermittelt:

1. Jeder der Männer A, B und C war am Tag des Geschehens in dem Laden gewesen, und kein anderer hatte den Laden an dem Tag betreten.
  2. Wenn A schuldig ist, so hat er genau einen Komplizen.
  3. Wenn B unschuldig ist, so ist auch C unschuldig.
  4. Wenn genau zwei schuldig sind, dann ist A einer von ihnen.
  5. Wenn C unschuldig ist, so ist auch B unschuldig.
- ▶ Beschreiben Sie die Ermittlungsergebnisse als logische Formeln.
  - ▶ Lösen sie das Verbrechen!

nach R. Smullyan: "Wie heißt dieses Buch?"

## Lösung: Raubüberfälle (Formalisierung)



- ▶ Jeder der Männer A, B und C war am Tag des Geschehens in dem Laden gewesen, und kein anderer hatte den Laden an dem Tag betreten.
- ▶ Wenn A schuldig ist, so hat er genau einen Komplizen.
- ▶ Wenn B unschuldig ist, so ist auch C unschuldig.
- ▶ Wenn genau zwei schuldig sind, dann ist A einer von ihnen.
- ▶ Wenn C unschuldig ist, so ist auch B unschuldig.



- ▶ Jeder der Männer A, B und C war am Tag des Geschehens in dem Laden gewesen, und kein anderer hatte den Laden an dem Tag betreten.
  - ▶  $A \vee B \vee C$
- ▶ Wenn A schuldig ist, so hat er genau einen Komplizen.
- ▶ Wenn B unschuldig ist, so ist auch C unschuldig.
- ▶ Wenn genau zwei schuldig sind, dann ist A einer von ihnen.
- ▶ Wenn C unschuldig ist, so ist auch B unschuldig.

## Lösung: Raubüberfälle (Formalisierung)



- ▶ Jeder der Männer A, B und C war am Tag des Geschehens in dem Laden gewesen, und kein anderer hatte den Laden an dem Tag betreten.
  - ▶  $A \vee B \vee C$
- ▶ Wenn A schuldig ist, so hat er genau einen Komplizen.
  - ▶  $A \rightarrow ((B \wedge \neg C) \vee (\neg B \wedge C))$
- ▶ Wenn B unschuldig ist, so ist auch C unschuldig.
- ▶ Wenn genau zwei schuldig sind, dann ist A einer von ihnen.
- ▶ Wenn C unschuldig ist, so ist auch B unschuldig.

# Lösung: Raubüberfälle (Formalisierung)



- ▶ Jeder der Männer A, B und C war am Tag des Geschehens in dem Laden gewesen, und kein anderer hatte den Laden an dem Tag betreten.
  - ▶  $A \vee B \vee C$
- ▶ Wenn A schuldig ist, so hat er genau einen Komplizen.
  - ▶  $A \rightarrow ((B \wedge \neg C) \vee (\neg B \wedge C))$
- ▶ Wenn B unschuldig ist, so ist auch C unschuldig.
  - ▶  $\neg B \rightarrow \neg C$
- ▶ Wenn genau zwei schuldig sind, dann ist A einer von ihnen.
- ▶ Wenn C unschuldig ist, so ist auch B unschuldig.

# Lösung: Raubüberfälle (Formalisierung)



- ▶ Jeder der Männer A, B und C war am Tag des Geschehens in dem Laden gewesen, und kein anderer hatte den Laden an dem Tag betreten.
  - ▶  $A \vee B \vee C$
- ▶ Wenn A schuldig ist, so hat er genau einen Komplizen.
  - ▶  $A \rightarrow ((B \wedge \neg C) \vee (\neg B \wedge C))$
- ▶ Wenn B unschuldig ist, so ist auch C unschuldig.
  - ▶  $\neg B \rightarrow \neg C$
- ▶ Wenn genau zwei schuldig sind, dann ist A einer von ihnen.
  - ▶  $\neg(B \wedge C \wedge \neg A)$
- ▶ Wenn C unschuldig ist, so ist auch B unschuldig.



- ▶ Jeder der Männer A, B und C war am Tag des Geschehens in dem Laden gewesen, und kein anderer hatte den Laden an dem Tag betreten.
  - ▶  $A \vee B \vee C$
- ▶ Wenn A schuldig ist, so hat er genau einen Komplizen.
  - ▶  $A \rightarrow ((B \wedge \neg C) \vee (\neg B \wedge C))$
- ▶ Wenn B unschuldig ist, so ist auch C unschuldig.
  - ▶  $\neg B \rightarrow \neg C$
- ▶ Wenn genau zwei schuldig sind, dann ist A einer von ihnen.
  - ▶  $\neg(B \wedge C \wedge \neg A)$
- ▶ Wenn C unschuldig ist, so ist auch B unschuldig.
  - ▶  $\neg C \rightarrow \neg B$

# Lösung: Raubüberfälle (Analyse)



| A | B | C | 1. | 2. | 3. | 4. | 5. |
|---|---|---|----|----|----|----|----|
| 0 | 0 | 0 | 0  | 1  | 1  | 1  | 1  |
| 0 | 0 | 1 | 1  | 1  | 0  | 1  | 1  |
| 0 | 1 | 0 | 1  | 1  | 1  | 1  | 0  |
| 0 | 1 | 1 | 1  | 1  | 1  | 0  | 1  |
| 1 | 0 | 0 | 1  | 0  | 1  | 1  | 1  |
| 1 | 0 | 1 | 1  | 1  | 0  | 1  | 1  |
| 1 | 1 | 0 | 1  | 1  | 1  | 1  | 0  |
| 1 | 1 | 1 | 1  | 0  | 1  | 1  | 1  |

# Lösung: Raubüberfälle (Analyse)



| A | B | C | 1. | 2. | 3. | 4. | 5. |
|---|---|---|----|----|----|----|----|
| 0 | 0 | 0 | 0  | 1  | 1  | 1  | 1  |
| 0 | 0 | 1 | 1  | 1  | 0  | 1  | 1  |
| 0 | 1 | 0 | 1  | 1  | 1  | 1  | 0  |
| 0 | 1 | 1 | 1  | 1  | 1  | 0  | 1  |
| 1 | 0 | 0 | 1  | 0  | 1  | 1  | 1  |
| 1 | 0 | 1 | 1  | 1  | 0  | 1  | 1  |
| 1 | 1 | 0 | 1  | 1  | 1  | 1  | 0  |
| 1 | 1 | 1 | 1  | 0  | 1  | 1  | 1  |

**Es gibt kein Modell der Aussagen**

# Lösung: Raubüberfälle (Analyse)



| A | B | C | 1. | 2. | 3. | 4. | 5. |
|---|---|---|----|----|----|----|----|
| 0 | 0 | 0 | 0  | 1  | 1  | 1  | 1  |
| 0 | 0 | 1 | 1  | 1  | 0  | 1  | 1  |
| 0 | 1 | 0 | 1  | 1  | 1  | 1  | 0  |
| 0 | 1 | 1 | 1  | 1  | 1  | 0  | 1  |
| 1 | 0 | 0 | 1  | 0  | 1  | 1  | 1  |
| 1 | 0 | 1 | 1  | 1  | 0  | 1  | 1  |
| 1 | 1 | 0 | 1  | 1  | 1  | 1  | 0  |
| 1 | 1 | 1 | 1  | 0  | 1  | 1  | 1  |

**Es gibt kein Modell der Aussagen**



**Versicherungsbetrug durch den Ladenbesitzer!**

Zur Vorlesung

- ▶ Bearbeiten Sie Übung: Aussagenlogik in Scheme

- ▶ Rückblick/Wiederholung
- ▶ Allgemeingültigkeit und Unerfüllbarkeit
- ▶ Logisches Folgern
- ▶ Wahrheitstafeln
- ▶ Bäume

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

# Ziele Vorlesung 14

- ▶ Rückblick/Wiederholung
- ▶ Tableaux-Kalkül für die Aussagenlogik
  - ▶ Vorstellung und Übung

## Ziele Vorlesung 14

- ▶ Rückblick/Wiederholung
- ▶ Tableaux-Kalkül für die Aussagenlogik
  - ▶ Vorstellung und Übung

**Erinnerung: Am Dienstag, 3.2. (für 14B) und Mittwoch, 4.2. (für 14C) fällt die Vorlesung jeweils aus. Der Dozent muß was lernen...**

- ▶  $F$  ist allgemeingültig:  $I(F) = 1$  für alle  $I$
- ▶  $F$  ist unerfüllbar:  $I(F) = 0$  für alle  $I$
- ▶ Logisches Folgern
  - ▶  $KB \models F$ : Alle Modelle von  $KB$  sind auch Modelle von  $F$
  - ▶ Wahrheitstafelmethode (gleich)
- ▶ Bäume
  - ▶ Knoten, Kanten, Wurzel, Pfade, Äste

# Die Wahrheitstafelmethode

- ▶ Wir wollen zeigen: Aus einer Formelmenge  $KB$  folgt eine Vermutung  $F$ .
- ▶ Wahrheitstafelmethode: Direkte Umsetzung der Definition von  $KB \models F$ 
  - ▶ Enumeriere alle Interpretationen in einer Tabelle
  - ▶ Für jede Interpretation:
    - ▶ Bestimme  $I(G)$  für alle  $G \in KB$
    - ▶ Bestimme  $I(F)$
    - ▶ Prüfe, ob jedes Modell von  $KB$  auch ein Modell von  $F$  ist

# Folgerungsproblem von Craig

- ▶ Inspektor Craigs Ergebnisse
  1. Wenn A schuldig und B unschuldig ist, so ist C schuldig.
  2. C arbeitet niemals allein.
  3. A arbeitet niemals mit C.
  4. Niemand außer A, B oder C war beteiligt, und mindestens einer von ihnen ist schuldig.
- ▶ Ermittlungsergebnisse formal
  1.  $(A \wedge \neg B) \rightarrow C$
  2.  $C \rightarrow (A \vee B)$
  3.  $A \rightarrow \neg C$
  4.  $A \vee B \vee C$
- ▶ Craigs erstes Problem: Sei  $KB = \{1., 2., 3., 4.\}$ . Gilt einer der folgenden Fälle?
  - ▶  $KB \models A$
  - ▶  $KB \models B$
  - ▶  $KB \models C$

# Schritt 1: Aufzählung aller möglichen Welten

## ► Ermittlungsergebnisse

1.  $(A \wedge \neg B) \rightarrow C$
2.  $C \rightarrow (A \vee B)$
3.  $A \rightarrow \neg C$
4.  $A \vee B \vee C$

## ► Vorgehen

- Enumeriere alle Interpretationen  $I$
- Berechne  $I(F)$  für alle  $F \in KB$
- Bestimme Modelle von  $KB$

| A | B | C | 1. | 2. | 3. | 4. | KB | Kommentar |
|---|---|---|----|----|----|----|----|-----------|
| 0 | 0 | 0 | 1  | 1  | 1  | 0  | 0  |           |
| 0 | 0 | 1 | 1  | 0  | 1  | 1  | 0  |           |
| 0 | 1 | 0 | 1  | 1  | 1  | 1  | 1  | Modell KB |
| 0 | 1 | 1 | 1  | 1  | 1  | 1  | 1  | Modell KB |
| 1 | 0 | 0 | 0  | 1  | 1  | 1  | 0  |           |
| 1 | 0 | 1 | 1  | 1  | 0  | 1  | 0  |           |
| 1 | 1 | 0 | 1  | 1  | 1  | 1  | 1  | Modell KB |
| 1 | 1 | 1 | 1  | 1  | 0  | 1  | 0  |           |

## Vermutung 1: A ist schuldig!

- ▶ Vermutung: A ist schuldig
- ▶ Prüfe, ob jedes Modell von  $KB$  auch ein Modell von  $A$  ist

| A | B | C | 1. | 2. | 3. | 4. | KB | Kommentar | A |
|---|---|---|----|----|----|----|----|-----------|---|
| 0 | 0 | 0 | 1  | 1  | 1  | 0  | 0  |           | 0 |
| 0 | 0 | 1 | 1  | 0  | 1  | 1  | 0  |           | 0 |
| 0 | 1 | 0 | 1  | 1  | 1  | 1  | 1  | Modell KB | 0 |
| 0 | 1 | 1 | 1  | 1  | 1  | 1  | 1  | Modell KB | 0 |
| 1 | 0 | 0 | 0  | 1  | 1  | 1  | 0  |           | 1 |
| 1 | 0 | 1 | 1  | 1  | 0  | 1  | 0  |           | 1 |
| 1 | 1 | 0 | 1  | 1  | 1  | 1  | 1  | Modell KB | 1 |
| 1 | 1 | 1 | 1  | 1  | 0  | 1  | 0  |           | 1 |

- ▶ Das ist nicht der Fall, es gilt also nicht  $KB \models A$
- ▶ Schreibweise auch  $KB \not\models A$ )

## Vermutung 2: $B$ ist schuldig!

- ▶ Vermutung:  $B$  ist schuldig
- ▶ Prüfe, ob jedes Modell von  $KB$  auch ein Modell von  $B$  ist

| <b>A</b> | <b>B</b> | <b>C</b> | <b>1.</b> | <b>2.</b> | <b>3.</b> | <b>4.</b> | <b>KB</b> | Kommentar | <b>B</b> |
|----------|----------|----------|-----------|-----------|-----------|-----------|-----------|-----------|----------|
| 0        | 0        | 0        | 1         | 1         | 1         | 0         | 0         |           | 0        |
| 0        | 0        | 1        | 1         | 0         | 1         | 1         | 0         |           | 0        |
| 0        | 1        | 0        | 1         | 1         | 1         | 1         | 1         | Modell KB | 1        |
| 0        | 1        | 1        | 1         | 1         | 1         | 1         | 1         | Modell KB | 1        |
| 1        | 0        | 0        | 0         | 1         | 1         | 1         | 0         |           | 0        |
| 1        | 0        | 1        | 1         | 1         | 0         | 1         | 0         |           | 0        |
| 1        | 1        | 0        | 1         | 1         | 1         | 1         | 1         | Modell KB | 1        |
| 1        | 1        | 1        | 1         | 1         | 0         | 1         | 0         |           | 1        |

- ▶ Das ist der Fall, es gilt also  $KB \models B$
- ▶ In allen möglichen Welten, in denen die Annahmen gelten, ist  $B$  schuldig!

# Komplexere Vermutung

- ▶ Vermutung:  $A$  oder  $B$  haben das Verbrechen begangen
- ▶ Prüfe, ob jedes Modell von  $KB$  auch ein Modell von  $A \vee B$  ist

| A | B | C | 1. | 2. | 3. | 4. | KB | Kommentar | $A \vee B$ |
|---|---|---|----|----|----|----|----|-----------|------------|
| 0 | 0 | 0 | 1  | 1  | 1  | 0  | 0  |           | 0          |
| 0 | 0 | 1 | 1  | 0  | 1  | 1  | 0  |           | 0          |
| 0 | 1 | 0 | 1  | 1  | 1  | 1  | 1  | Modell KB | 1          |
| 0 | 1 | 1 | 1  | 1  | 1  | 1  | 1  | Modell KB | 1          |
| 1 | 0 | 0 | 0  | 1  | 1  | 1  | 0  |           | 1          |
| 1 | 0 | 1 | 1  | 1  | 0  | 1  | 0  |           | 1          |
| 1 | 1 | 0 | 1  | 1  | 1  | 1  | 1  | Modell KB | 1          |
| 1 | 1 | 1 | 1  | 1  | 0  | 1  | 0  |           | 1          |

- ▶ Das ist der Fall, es gilt also (logisch)  $KB \models (A \vee B)$

# Aussagenlogik in Scheme (1)

- ▶ Meine Version:

- ▶ Interpretationen sind Scheme-Funktionen, die Atome auf Scheme-Wahrheitswerte abbilden
- ▶ Formeln sind Scheme-Ausdrücke mit Operatoren: 'not, 'or, 'and, 'implies, 'equiv

- ▶ Beispielformel:  $a \rightarrow (b \wedge \neg c) \vee (\neg b \wedge c)$

```
(define g2
 '(implies a (or (and b (not c)) (and (not b) c))))
```

- ▶ Beispielinterpretation:

```
(define (make-var-interpretation true-vars)
 (lambda (x) (if (member x true-vars) #t #f)))
```

```
(define I1 (make-var-interpretation '(a b)))
(define I2 (make-var-interpretation '(a b c)))
```

## Aussagenlogik in Scheme(2)

```
(define (prop-eval form var-interpretation)
 (if (symbol? form)
 (var-interpretation form)
 (let ((op (car form))
 (args (map (lambda (x)
 (prop-eval x var-interpretation))
 (cdr form))))
 (cond ((equal? op 'not)
 (not (car args)))
 ((equal? op 'or)
 (or (car args) (cadr args)))
 ((equal? op 'and)
 (and (car args) (cadr args)))
 ((equal? op 'implies)
 (or (not (car args)) (cadr args)))
 ((equal? op 'equiv)
 (equal? (car args) (cadr args)))
 (else
 (display "Error_in_formula")
 (newline))))))
```

## Aussagenlogik in Scheme(3)

```
> (define g2
 '(implies a (or (and b (not c)) (and (not b) c))))
```

```
> (define l1 (make-var-interpretation '(a b)))
```

```
> (define l2 (make-var-interpretation '(a b c)))
```

```
> (prop-eval g2 l1)
```

```
==> #t
```

```
> (prop-eval g2 l2)
```

```
==> #f
```

Zur Vorlesung

- ▶ Bearbeiten Sie Übung: Tableaux für Inspektor Craig

- ▶ Rückblick/Wiederholung
- ▶ Tableaux-Kalkül für die Aussagenlogik
  - ▶ Vorstellung und Übung

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

# Ziele Vorlesung 15

- ▶ Rückblick/Wiederholung
- ▶ Scheme: Typen und Symbole
- ▶ Tableaux-Kalkül für die Aussagenlogik
  - ▶ Korrektheit und Vollständigkeit
  - ▶ Strategien
- ▶ Logische Äquivalenz

# Ziele Vorlesung 15

- ▶ Rückblick/Wiederholung
- ▶ Scheme: Typen und Symbole
- ▶ Tableaux-Kalkül für die Aussagenlogik
  - ▶ Korrektheit und Vollständigkeit
  - ▶ Strategien
- ▶ Logische Äquivalenz

**Erinnerung: Am Dienstag, 3.2. (für 14B) und Mittwoch, 4.2. (für 14C) fällt die Vorlesung jeweils aus. Der Dozent muß was lernen...**

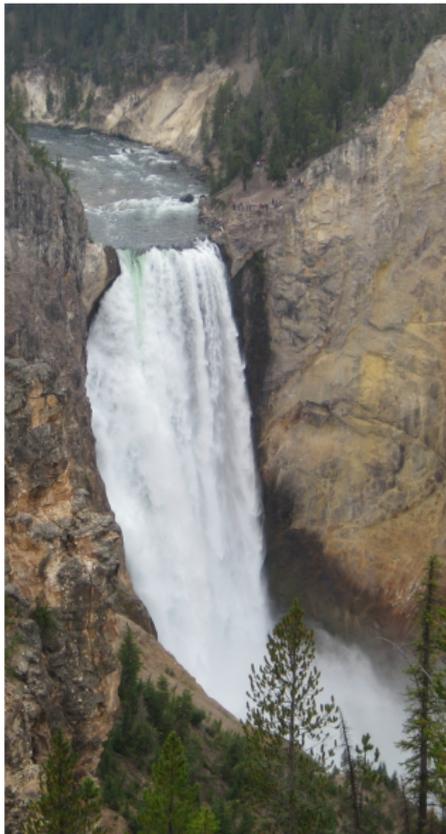
- ▶ Besprechung: Logik in Scheme
- ▶ Tableaux-Kalkül
  - ▶  $\alpha$  - /  $\beta$ -Zerlegungen
  - ▶ Tableau-Konstruktion
  - ▶ Geschlossenes Tableau = Unerfüllbare Wurzel

# Tableau: Craig 1





# Das Typsystem von Scheme



- ▶ Scheme ist eine strikt, aber **dynamisch** getypte Sprache:
  - ▶ Jedes Datenobjekt hat einen eindeutigen Basistyp
  - ▶ Dieser Typ geht direkt aus dem Objekt hervor, nicht aus seiner Speicherstelle ("Variable")
  - ▶ Variablen können an Objekte verschiedenen Typs gebunden sein
- ▶ Objekte können in Listen (und Vektoren) zu komplexeren Strukturen kombiniert werden

# Die Typen von Scheme

- ▶ Typprädikate (jedes Objekt hat genau einen dieser Typen):
  - boolean?            #t und #f
  - pair?                cons-Zellen (damit auch nicht-leere Listen)
  - symbol?             Normale Bezeichner, z.B. hallo, \*, symbol?. Achtung: Symbole müssen gequoted werden, wenn man das Symbol, nicht seinen Wert referenzieren will!
  
  - number?             Zahlen: 1, 3.1415, ...
  - char?                Einzelne Zeichen: #\a, #\b, #\7, ...
  - string?             "Hallo", "1", "1/2 oder Otto"
  - vector?             Aus Zeitmangel nicht erwähnt (nehmen Sie Listen)
  - port?                Siehe Vorlesung zu Input/Output
  - procedure?         Ausführbare Funktionen (per define oder lambda)
  - null?                Sonderfall: Die leere Liste '()

# Symbole als Werte

*Symbols are objects whose usefulness rests on the fact that two symbols are identical (in the sense of eqv?) if and only if their names are spelled the same way.*

R5RS (6.3.3)

- ▶ Symbole sind eine eigener Scheme-Datentyp
  - ▶ Sie können als Variablennamen dienen, sind aber auch selbst Werte
  - ▶ Ein Symbol wird durch **quoten** direkt verwendet (sonst in der Regel sein **Wert**)
- ▶ Beispiel:

```
> (define a 'hallo)
> a
=> hallo
> (define l '(hallo dings bums))
> l
=> (hallo dings bums)
> (equal? (car l) a)
=> #t
```

Zur Vorlesung

- ▶ Bearbeiten Sie Übung: Basis der Aussagenlogik

- ▶ Rückblick/Wiederholung
- ▶ Tableaux-Kalkül für die Aussagenlogik
  - ▶ Korrektheit und Vollständigkeit
  - ▶ Strategien
- ▶ Logische Äquivalenz

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

- ▶ Rückblick/Wiederholung
- ▶ Hausaufgabe: Basis der Aussagenlogik
- ▶ Substitutionstheorem
- ▶ Kalkül der Äquivalenzumformungen

- ▶ Scheme-Reste (Typen und Symbole)
- ▶ Abschluß des Tableaux-Kalküls
  - ▶ Korrektheit und Vollständigkeit
  - ▶ Modellextraktion
  - ▶ Strategien
- ▶ Logische Äquivalenz
  - ▶ Zwei Formeln sind logisch äquivalent, wenn sie unter allen Interpretationen gleich ausgewertet werden
- ▶ Teilformeln
- ▶ Beweisprinzip: Strukturelle Induktion

## **Definition (Basis der Aussagenlogik):**

Eine Menge von Operatoren  $O$  heißt eine **Basis** der Aussagenlogik, falls gilt: Zu jeder Formel  $F$  gibt es eine Formel  $G$  mit  $F \equiv G$ , und  $G$  verwendet nur Operatoren aus  $O$ .

- ▶ Das Konzept ermöglicht es, viele Aussagen zu Erfüllbarkeit, Folgerungen, und Äquivalenz auf einfachere Formelklassen zu beschränken.

## **Satz:**

- ▶  $\{\wedge, \vee, \neg\}$  ist eine Basis
- ▶  $\{\rightarrow, \neg\}$  ist eine Basis
- ▶  $\{\wedge, \neg\}$  ist eine Basis

# Beweisprinzip: Strukturelle Induktion

- ▶ Die **strukturelle Induktion** oder **Induktion über den Aufbau** kann verwendet werden, um Eigenschaften von rekursiv definierten Objekten zu zeigen.
- ▶ Idee:
  - ▶ Zeige die Eigenschaft für die elementaren Objekte.
  - ▶ Zeige die Eigenschaft für die zusammengesetzten Objekte unter der Annahme, dass die Teile bereits die Eigenschaft haben
- ▶ Konkret für aussagenlogische Formeln:
  - ▶ Basisfälle sind die Aussagenlogischen Variablen oder  $\top$ ,  $\perp$ .
  - ▶ Zusammengesetzte Formeln haben die Form  $\neg A$ ,  $A \vee B$ ,  $A \wedge B \dots$ , und wir können annehmen, dass  $A$  und  $B$  schon die gewünschte Eigenschaft haben.

## $\{\rightarrow, \neg\}$ ist eine Basis der Aussagenlogik (1)

Zu zeigen: Für jedes  $F \in \text{For}_{0\Sigma}$  existiert ein  $G \in \text{For}_{0\Sigma}$  so dass  $F \equiv G$  und  $G$  nur die Operatoren  $\neg, \rightarrow$  enthält.

## $\{\rightarrow, \neg\}$ ist eine Basis der Aussagenlogik (1)

Zu zeigen: Für jedes  $F \in \text{For}_{0\Sigma}$  existiert ein  $G \in \text{For}_{0\Sigma}$  so dass  $F \equiv G$  und  $G$  nur die Operatoren  $\neg, \rightarrow$  enthält.

Beweis per struktureller Induktion

## $\{\rightarrow, \neg\}$ ist eine Basis der Aussagenlogik (1)

Zu zeigen: Für jedes  $F \in For_{0\Sigma}$  existiert ein  $G \in For_{0\Sigma}$  so dass  $F \equiv G$  und  $G$  nur die Operatoren  $\neg, \rightarrow$  enthält.

Beweis per struktureller Induktion

**Induktionsanfang:** Sei  $F$  eine elementare Formel. Dann gilt einer der folgenden Fälle:

1.  $F \in \Sigma$ : Dann gilt:  $F$  enthält keine Operatoren. Also hat  $F$  bereits die geforderten Eigenschaften.
2.  $F = \top$ : Sei  $a \in \Sigma$  ein beliebiges Atom. Wähle  $G = a \rightarrow a$ . Dann gilt für jede Interpretation  $I$ :  $I(G) = 1$ . Also gilt auch  $G \equiv F$  und  $G$  enthält nur den Operator  $\rightarrow$ .
3.  $F = \perp$ : Analog mit  $G = \neg(a \rightarrow a)$ .

Damit sind alle Basisfälle abgedeckt und der Induktionsanfang ist gesichert.

## $\{\rightarrow, \neg\}$ ist eine Basis der Aussagenlogik (2)

**Induktionsvoraussetzung:** Die Behauptung gelte für beliebige  $A, B \in For_{0\Sigma}$  (also: es existieren  $A', B' \in For_{0\Sigma}$  mit  $A \equiv A'$ ,  $B \equiv B'$ , und  $A', B'$  enthalten nur die Operatoren  $\rightarrow, \neg$ ).

## $\{\rightarrow, \neg\}$ ist eine Basis der Aussagenlogik (2)

**Induktionsvoraussetzung:** Die Behauptung gelte für beliebige  $A, B \in \text{For}_{0\Sigma}$  (also: es existieren  $A', B' \in \text{For}_{0\Sigma}$  mit  $A \equiv A'$ ,  $B \equiv B'$ , und  $A', B'$  enthalten nur die Operatoren  $\rightarrow, \neg$ ).

**Induktionsschritt:** Sei  $F$  eine zusammengesetzte Formel. Dann gilt:  
 $F = (\neg A)$  oder  $F = (A \vee B)$  oder  $F = (A \wedge B)$  oder  
 $F = (A \rightarrow B)$  oder  $F = (A \leftrightarrow B)$ .

## $\{\rightarrow, \neg\}$ ist eine Basis der Aussagenlogik (2)

**Induktionsvoraussetzung:** Die Behauptung gelte für beliebige  $A, B \in For_{0\Sigma}$  (also: es existieren  $A', B' \in For_{0\Sigma}$  mit  $A \equiv A'$ ,  $B \equiv B'$ , und  $A', B'$  enthalten nur die Operatoren  $\rightarrow, \neg$ ).

**Induktionsschritt:** Sei  $F$  eine zusammengesetzte Formel. Dann gilt:  
 $F = (\neg A)$  oder  $F = (A \vee B)$  oder  $F = (A \wedge B)$  oder  
 $F = (A \rightarrow B)$  oder  $F = (A \leftrightarrow B)$ .

Fallunterscheidung:

1.  $F = (\neg A)$ : Nach IV gibt es ein  $A'$  mit  $A \equiv A'$ ,  $A'$  enthält nur die Operatoren  $\rightarrow, \neg$ . Betrachte  $G = (\neg A')$ . Dann gilt:  $I(G) = I(F)$  für alle Interpretationen, also  $G \equiv F$ .

## $\{\rightarrow, \neg\}$ ist eine Basis der Aussagenlogik (3)

2.  $F = (A \vee B)$ : Nach IV gibt es  $A'$  mit  $A \equiv A'$ ,  $B \equiv B'$ . Betrachte  $G = (\neg A' \rightarrow B')$ . Dann gilt  $I(G) = I(F)$  für alle Interpretationen (siehe Fallunterscheidung in folgender Tabelle):

| $A$ | $B$ | $F$ | $A'$ (IV) | $B'$ (IV) | $(\neg A')$ | $G$ |
|-----|-----|-----|-----------|-----------|-------------|-----|
| 0   | 0   | 0   | 0         | 0         | 1           | 0   |
| 0   | 1   | 1   | 0         | 1         | 1           | 1   |
| 1   | 0   | 1   | 1         | 0         | 0           | 1   |
| 1   | 1   | 1   | 1         | 1         | 0           | 1   |

Also:  $G \equiv F$ , und  $G$  enthält nach Konstruktion nur die Operatoren  $\rightarrow, \neg$ .

## $\{\rightarrow, \neg\}$ ist eine Basis der Aussagenlogik (4)

3.  $F = (A \wedge B)$ : Nach IV gibt es  $A'$  mit  $A \equiv A'$ ,  $B \equiv B'$ . Betrachte  $G = (\neg(A' \rightarrow \neg B'))$ . Dann gilt  $I(G) = I(F)$  für alle Interpretationen (siehe Fallunterscheidung in folgender Tabelle):

| $A$ | $B$ | $F$ | $A'$ (IV) | $B'$ (IV) | $G = (\neg(A' \rightarrow \neg B'))$ |
|-----|-----|-----|-----------|-----------|--------------------------------------|
| 0   | 0   | 0   | 0         | 0         | 0                                    |
| 0   | 1   | 0   | 0         | 1         | 0                                    |
| 1   | 0   | 0   | 1         | 0         | 0                                    |
| 1   | 1   | 1   | 1         | 1         | 1                                    |

Also:  $G \equiv F$ , und  $G$  enthält nach Konstruktion nur die Operatoren  $\rightarrow, \neg$ .

## $\{\rightarrow, \neg\}$ ist eine Basis der Aussagenlogik (5)

4.  $F = (A \rightarrow B)$ : Nach IV gibt es  $A'$  mit  $A \equiv A'$ ,  $B \equiv B'$ . Betrachte  $G = (A' \rightarrow B')$ . Dann gilt  $I(G) = I(F)$  für alle Interpretationen und  $G$  enthält nach Konstruktion nur die Operatoren  $\rightarrow, \neg$ .

## $\{\rightarrow, \neg\}$ ist eine Basis der Aussagenlogik (5)

4.  $F = (A \rightarrow B)$ : Nach IV gibt es  $A'$  mit  $A \equiv A'$ ,  $B \equiv B'$ . Betrachte  $G = (A' \rightarrow B')$ . Dann gilt  $I(G) = I(F)$  für alle Interpretationen und  $G$  enthält nach Konstruktion nur die Operatoren  $\rightarrow, \neg$ .
5.  $F = (A \leftrightarrow B)$ : Nach IV gibt es  $A'$  mit  $A \equiv A'$ ,  $B \equiv B'$ . Betrachte  $G = (\neg((A' \rightarrow B') \rightarrow (\neg(B' \rightarrow A'))))$ . Dann gilt (per Nachrechnen ;-))  $I(G) = I(F)$  für alle Interpretationen und  $G$  enthält nach Konstruktion nur die Operatoren  $\rightarrow, \neg$ .

## $\{\rightarrow, \neg\}$ ist eine Basis der Aussagenlogik (5)

4.  $F = (A \rightarrow B)$ : Nach IV gibt es  $A'$  mit  $A \equiv A'$ ,  $B \equiv B'$ . Betrachte  $G = (A' \rightarrow B')$ . Dann gilt  $I(G) = I(F)$  für alle Interpretationen und  $G$  enthält nach Konstruktion nur die Operatoren  $\rightarrow, \neg$ .
5.  $F = (A \leftrightarrow B)$ : Nach IV gibt es  $A'$  mit  $A \equiv A'$ ,  $B \equiv B'$ . Betrachte  $G = (\neg((A' \rightarrow B') \rightarrow (\neg(B' \rightarrow A'))))$ . Dann gilt (per Nachrechnen ;-))  $I(G) = I(F)$  für alle Interpretationen und  $G$  enthält nach Konstruktion nur die Operatoren  $\rightarrow, \neg$ .

Also: In allen Fällen können wir eine zu  $F$  äquivalente Formel angeben, die nur  $\rightarrow, \neg$  als Operatoren enthält. Damit gilt der IS, und damit die Behauptung.

q.e.d.

# Übung: Basis der Aussagenlogik

- ▶ Erinnerung: Eine Menge von Operatoren  $O$  heißt eine **Basis** der Aussagenlogik, falls gilt: Zu jeder Formel  $F$  gibt es eine Formel  $G$  mit  $F \equiv G$ , und  $G$  verwendet nur Operatoren aus  $O$ .
- ▶ Zeigen Sie (wahlweise):
  - ▶  $\{\wedge, \vee, \neg\}$  ist eine Basis
  - ▶  $\{\wedge, \neg\}$  ist eine Basis

Zur Vorlesung

- ▶ Vollziehen Sie den Beweis zum Substitutionstheorem nach.

- ▶ Rückblick/Wiederholung
- ▶ Hausaufgabe: Basis der Aussagenlogik
- ▶ Substitutionstheorem
- ▶ Kalkül der Äquivalenzumformungen

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

- ▶ Rückblick/Wiederholung
- ▶ Normalformen
  - ▶ NNF
  - ▶ KNF
- ▶ Resolution für Aussagenlogik

- ▶ Basen der Aussagenlogik
- ▶ Strukturelle Induktion
- ▶ Äquivalenzumformungen
- ▶ Kalkül  $\vdash_{LU}$

Zur Vorlesung

- ▶ Bearbeiten Sie Übung: Resolution von Jane

- ▶ Rückblick/Wiederholung
- ▶ Normalformen
  - ▶ NNF
  - ▶ KNF
- ▶ Resolution für Aussagenlogik

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

- ▶ Rückblick/Wiederholung
- ▶ Prädikatenlogik 1. Stufe
  - ▶ Syntax
  - ▶ Semantik

- ▶ Negations-Normalform
- ▶ Konjunktive Normalform
- ▶ Unerfüllbarkeit mit Resolution

# Lösung: KNF Transformation

$$\begin{aligned} & \neg(a \vee b) \leftrightarrow (b \wedge (a \rightarrow \neg c)) \\ = & (\neg(a \vee b) \rightarrow (b \wedge (a \rightarrow \neg c))) \wedge ((b \wedge (a \rightarrow \neg c)) \rightarrow \neg(a \vee b)) \\ = & (a \vee b \vee (b \wedge (\neg a \vee \neg c))) \wedge (\neg(b \wedge (\neg a \vee \neg c)) \vee \neg(a \vee b)) \\ = & (a \vee b \vee (b \wedge (\neg a \vee \neg c))) \wedge (\neg b \vee \neg(\neg a \vee \neg c) \vee (\neg a \wedge \neg b)) \\ = & (a \vee b \vee (b \wedge (\neg a \vee \neg c))) \wedge (\neg b \vee (a \wedge c) \vee (\neg a \wedge \neg b)) \text{ (NNF)} \\ = & (b \vee a \vee b) \wedge (\neg a \vee \neg c \vee a \vee b) \wedge (\neg a \vee a \vee \neg b) \\ & \wedge (\neg b \vee a \vee \neg b) \wedge (\neg a \vee c \vee \neg b) \wedge (\neg b \vee c \vee \neg b) \\ = & (b \vee a) \wedge (\neg b \vee a) \wedge (\neg a \vee c \vee \neg b) \wedge (\neg b \vee c) \end{aligned}$$

# Hausaufgabe: Resolution von Jane

Axiome:

1.  $\neg K \wedge E \rightarrow M$
2.  $B \rightarrow E$
3.  $\neg B \rightarrow F$
4.  $\neg M$
5.  $\neg K$

Hypothese:

6.  $F$

# Hausaufgabe: Resolution von Jane

Axiome:

1.  $\neg K \wedge E \rightarrow M$
2.  $B \rightarrow E$
3.  $\neg B \rightarrow F$
4.  $\neg M$
5.  $\neg K$

Hypothese:

6.  $F$

Axiome KNF:

1.  $K \vee \neg E \vee M$
2.  $\neg B \vee E$
3.  $B \vee F$
4.  $\neg M$
5.  $\neg K$

Hypothese negiert, KNF:

6.  $\neg F$

# Hausaufgabe: Resolution von Jane

Axiome:

1.  $\neg K \wedge E \rightarrow M$
2.  $B \rightarrow E$
3.  $\neg B \rightarrow F$
4.  $\neg M$
5.  $\neg K$

Hypothese:

6.  $F$

Resolutionsbeweis:

7.  $B$  (aus 3, 6)
8.  $E$  (aus 2, 7)
9.  $K \vee M$  (aus 1, 8)
10.  $K$  (aus 4, 9)
11.  $\square$  (aus 5, 10)

Axiome KNF:

1.  $K \vee \neg E \vee M$
2.  $\neg B \vee E$
3.  $B \vee F$
4.  $\neg M$
5.  $\neg K$

Hypothese negiert, KNF:

6.  $\neg F$

# Hausaufgabe: Resolution von Jane

Axiome:

1.  $\neg K \wedge E \rightarrow M$
2.  $B \rightarrow E$
3.  $\neg B \rightarrow F$
4.  $\neg M$
5.  $\neg K$

Hypothese:

6.  $F$

Resolutionsbeweis:

7.  $B$  (aus 3, 6)
8.  $E$  (aus 2, 7)
9.  $K \vee M$  (aus 1, 8)
10.  $K$  (aus 4, 9)
11.  $\square$  (aus 5, 10)

Axiome KNF:

1.  $K \vee \neg E \vee M$
2.  $\neg B \vee E$
3.  $B \vee F$
4.  $\neg M$
5.  $\neg K$

Hypothese negiert, KNF:

6.  $\neg F$

Hinweis: Set-of-Support Strategie  
Mindestens eine Klausel in jedem  
Schritt stammt von der Hypothese  
ab.

Zur Vorlesung

- ▶ Bearbeiten Sie Übung: Primzahlen

- ▶ Rückblick/Wiederholung
- ▶ Prädikatenlogik 1. Stufe
  - ▶ Syntax
  - ▶ Semantik

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

- ▶ Rückblick/Wiederholung
- ▶ Unerfüllbarkeit für Prädikatenlogik
- ▶ Unifikation
- ▶ Resolution

- ▶ Prädikatenlogik 1. Stufe
  - ▶ Syntax
    - ▶ Terme
    - ▶ Atome
    - ▶ Formeln (mit  $\forall, \exists$ )
  - ▶ Semantik
    - ▶ Interpretation:  $\langle U, I \rangle$
    - ▶ Universum
    - ▶ Funktionssymbole werden als Funktionen interpretiert
    - ▶ Prädikatssymbole werden als Prädikate interpretiert

- ▶ Bearbeiten Sie einige der Aufgaben aus der Aufgabensammlung.

- ▶ Rückblick/Wiederholung
- ▶ Unerfüllbarkeit für Prädikatenlogik
- ▶ Unifikation
- ▶ Resolution

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

- ▶ Rückblick/Wiederholung
- ▶ Übung

- ▶ Unerfüllbarkeit für Prädikatenlogik
- ▶ Klausellogik
- ▶ Unifikation
- ▶ Resolution

# Übung: Tierische Resolution

► Klauselmenge:

1.  $\neg h(X) \vee l(X)$
2.  $\neg k(X) \vee \neg \text{hat}(Y, X) \vee \neg \text{hat}(Y, Z) \vee \neg m(Z)$
3.  $\neg e(X) \vee \neg \text{hat}(X, Y) \vee \neg l(Y)$
4.  $\text{hat}(\text{john}, \text{tier})$
5.  $h(\text{tier}) \vee k(\text{tier})$
6.  $e(\text{john})$
7.  $\text{hat}(\text{john}, \text{maus})$
8.  $m(\text{maus})$

► Beweis:

9.  $\neg k(X) \vee \neg \text{hat}(\text{john}, X) \vee \neg m(\text{maus})$  (2,7)
10.  $\neg k(X) \vee \neg \text{hat}(\text{john}, X)$  (8,9)
11.  $\neg k(\text{tier})$  (10, 4)
12.  $\neg e(\text{john}) \vee \neg l(\text{tier})$  (3,4)
13.  $\neg l(\text{tier})$  (12, 5)
14.  $\neg h(\text{tier})$  (1,13)
15.  $k(\text{tier})$  (5,14)
16.  $\square$  (15,11)

# Hausaufgabe



- ▶ Rückblick/Wiederholung

- ▶ Was war der beste Teil der heutigen Vorlesung?
- ▶ Was kann verbessert werden?
  - ▶ Optional: Wie?

## Übungsklausur

- ▶ Gerne direkt wie immer
- ▶ Auch gerne anonym per Web-System

**Viel Glück für die Klausuren!**

Ende