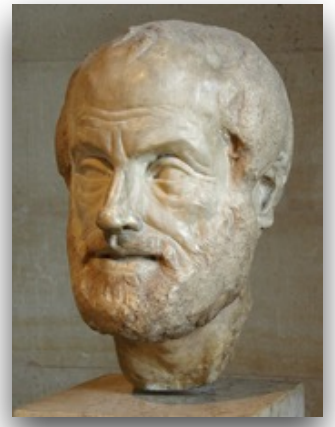
Logik und Grundlagen der Informatik

Stephan Schulz

stephan.schulz@dhbw-stuttgart.de



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

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

DHBW Stuttgart
Duale Hochschule
Baden-Württemberg

Inhaltsverzeichnis

Einführung
Mengenlehre
- Zahlenkonstruktion und Termalgebra
- Mengenoperationen
- Kartesische Produkte, Potenzmengen
- Relationen
- Funktionen

Funktionales Programmieren mit Scheme
Formale Logik
- Aussagenlogik
- Prädikatenlogik

Anhang: Kurzübersicht Scheme
Bonusaufgabe: Türme von Hanoi
Einige Lösungen
Einzelvorlesungen
- 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
- Vorlesung 22

Berufsbild

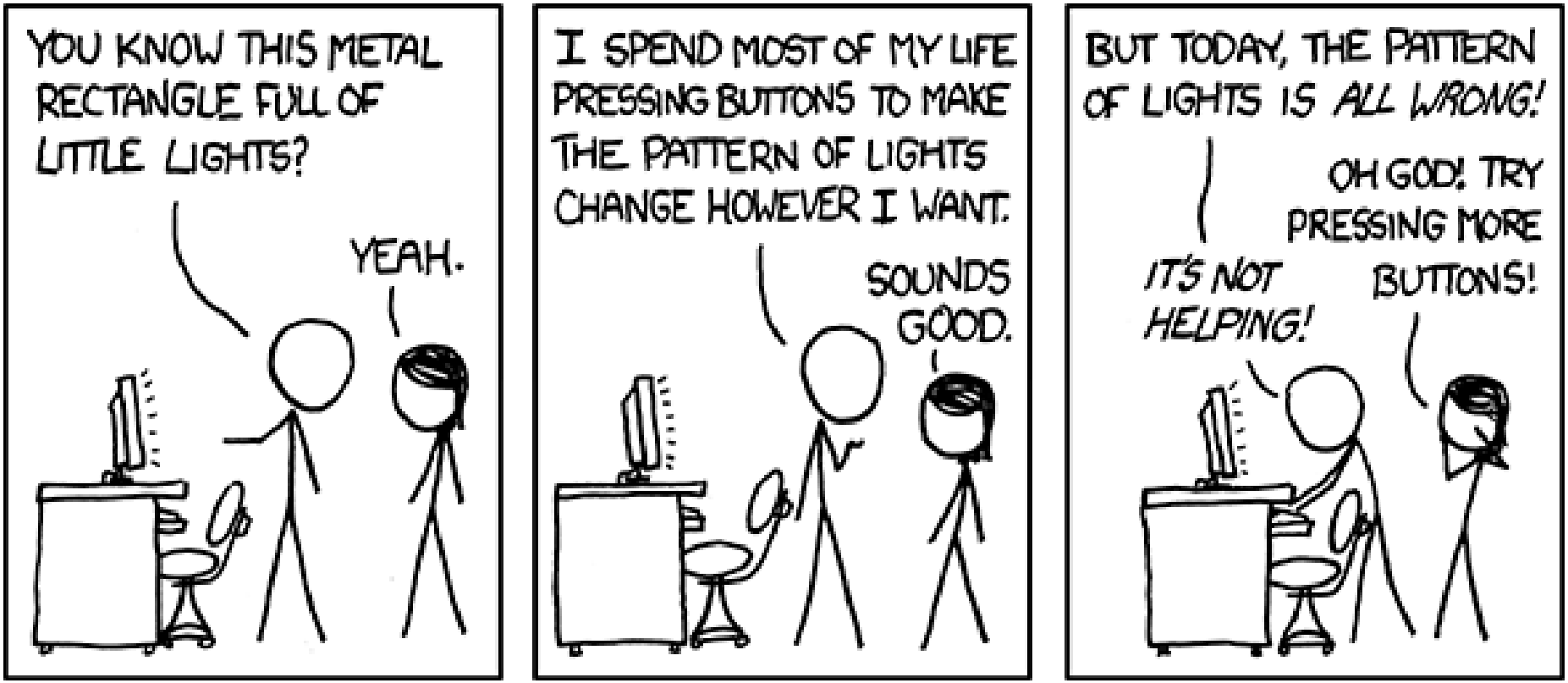


Image credit: `http://xkcd.com/722/`

Semesterübersicht

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

Respect Logic!

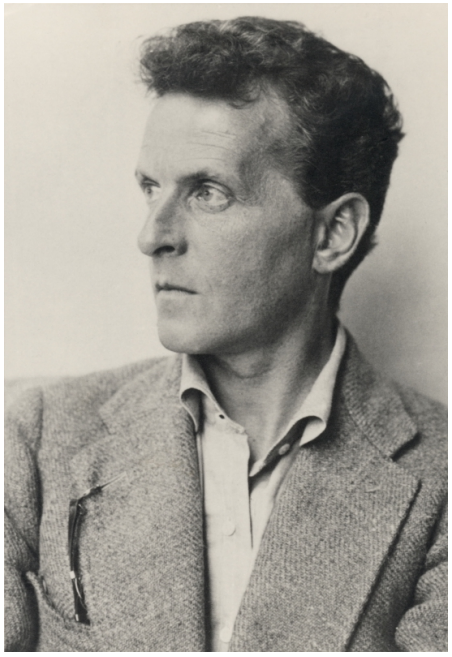
`https://www.youtube.com/watch?v=khhJSqwDF3U`

Literatur

- Allgemein
  - Dirk W. Hoffmann: Theoretische Informatik
  - Karl Stroetmann: Theoretische Informatik I - Logik und Mengenlehre (Skript der Vorlesung 2012/2013 an der DHBW),
    `http://wwwlehre.dhbw-stuttgart.de/~stroetma/Logic/logik-2013.pdf`
- Interessante Klassiker
  - Bertrand Russell: Introduction to Mathematical Philosophy (1918),
    `https://wwwlehre.dhbw-stuttgart.de/~sschulz/RESOURCES/41654-pdf.pdf`
  - Raymond M. Smullyan: First-Order Logic (1968)
- Scheme
  - Kelsey, Clinger, Rees (editors): Revised^5 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 4.0:
    `https://mitp-content-server.mit.edu/books/content/sectbyfn/books_pres_0/6515/sicp.zip/index.html`

Ziele der Vorlesung (1): Vokabular

- These von Sapir–Whorf: "*Language determines thought, linguistic categories limit and determine cognitive categories.*"
  - Wie spricht man über Argumente?
  - Wie beschreibt man Algorithmen, Datenstrukturen und Programme abstrakt?
  - Was sind ...
    - Syntax
    - Semantik
    - Interpretation
    - Modell
    - Wahrheit
    - Gültigkeit
    - Ableitbarkeit?



Die Grenzen meiner Sprache bedeuten die Grenzen meiner Welt
– Ludwig Wittgenstein

Ziele der Vorlesung (2): Methoden

- 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

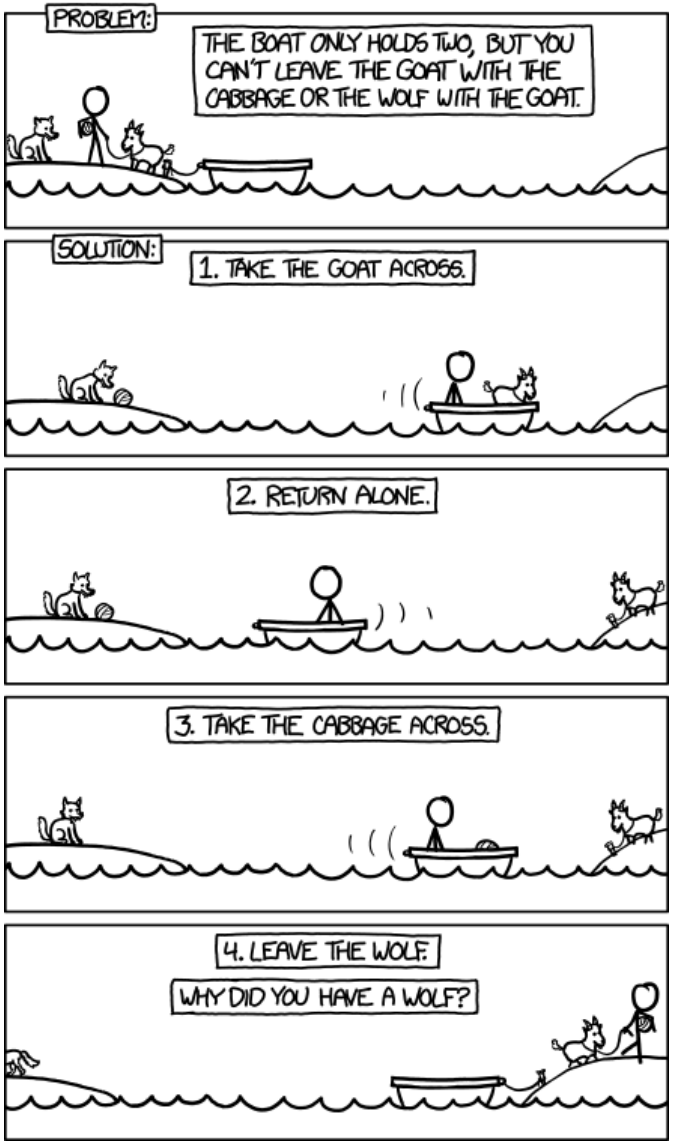


Image source: `https://xkcd.com/1134`

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. $\texttt{xI} \rightarrow \texttt{xIU}$
  2. $\texttt{xIIIy} \rightarrow \texttt{xUy}$
  3. $\texttt{xUUy} \rightarrow \texttt{xy}$
  4. $\texttt{Mx} \rightarrow \texttt{Mxx}$
- Wir schreiben $\texttt{x} \vdash \texttt{y}$, wenn `x` durch eine Anwendung einer Regel in `y` überführen läßt
  - Beispiel: $\texttt{MI} \vdash_4 \texttt{MII} \vdash_4 \texttt{MIIII} \vdash_2 \texttt{MUI} \vdash_4 \texttt{MUIUI}$
- Aufgabe: Geben Sie für die folgenden Worte Ableitungen an:
  1. `MUIU`
  2. `MIIIII`
  3. `MUUUI`
  4. `MU`

MIU Lösungen (1)

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

Lösungen

1. $\texttt{MI} \vdash \texttt{MII} \vdash \texttt{MIIII} \vdash \texttt{MIIIIU} \vdash \texttt{MUIU}$
2. $\texttt{MI} \vdash \texttt{MII} \vdash \texttt{MIIII} \vdash \texttt{MIIIIIIII} \vdash \texttt{MIIIIIIIIU} \vdash \texttt{MIIIIIUU} \vdash \texttt{MIIIII}$
3. $\texttt{MI} \vdash \texttt{MIIII} \vdash \texttt{MIIIIIIII} \vdash \texttt{MUIIIII} \vdash \texttt{MUUII} \vdash \texttt{MUUIIUUII} \vdash \texttt{MUUIIII} \vdash \texttt{MUUUI}$

MIU Lösungen (2)

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

Lösungen

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

Reflektion

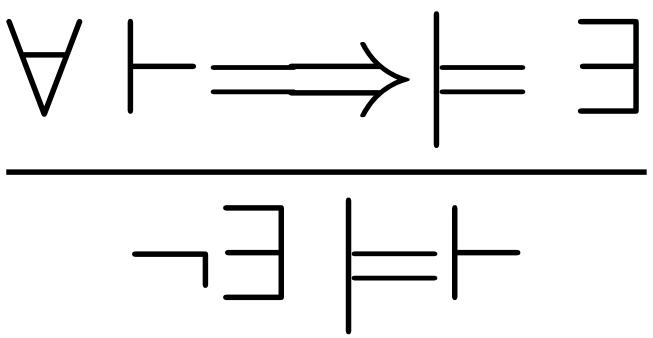
Vokabular
Methoden

Logik

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

Deduktionsmethoden

- 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?

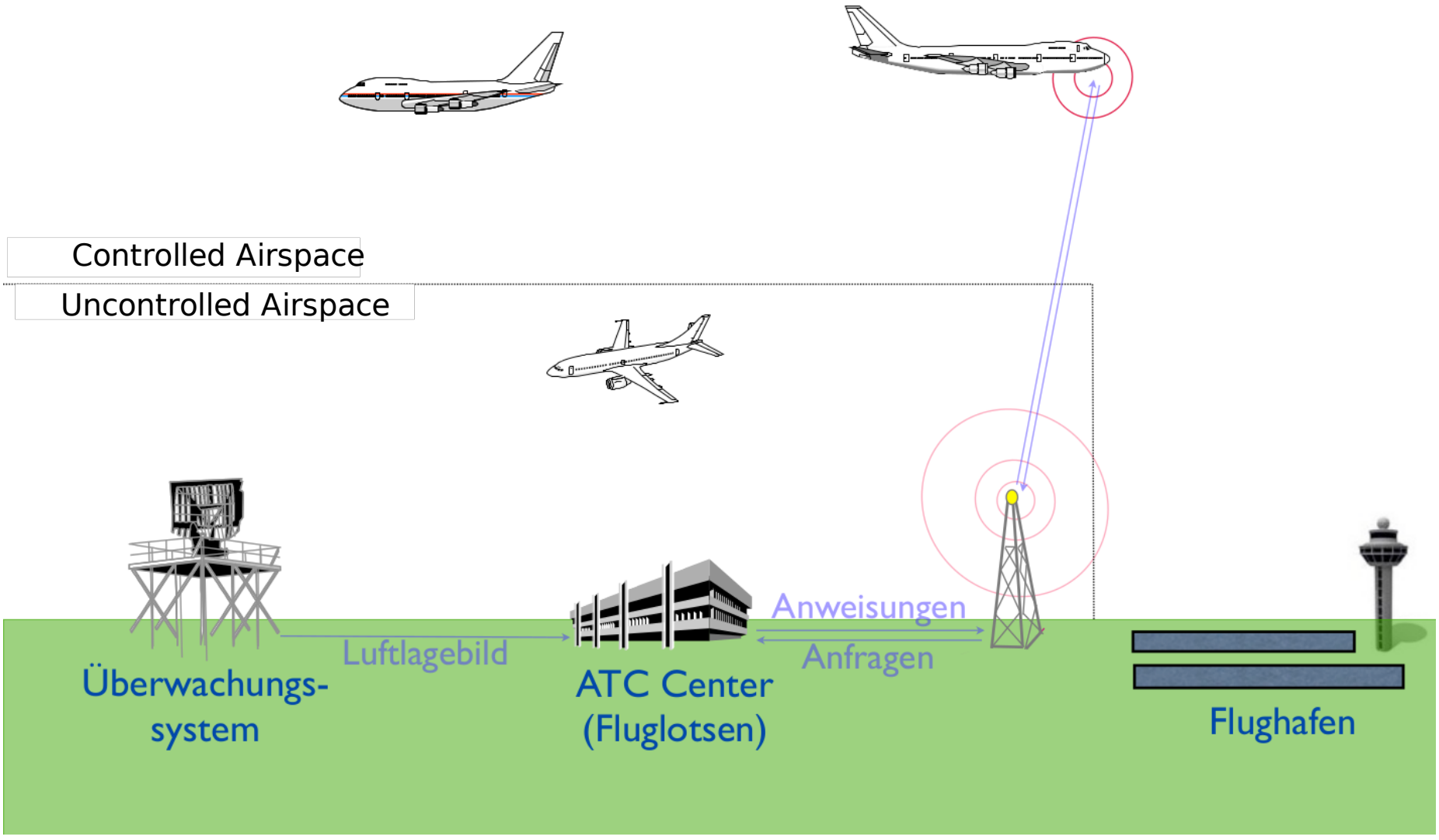


Anwendungsbeispiel: Äquivalenz von Spezifikationen

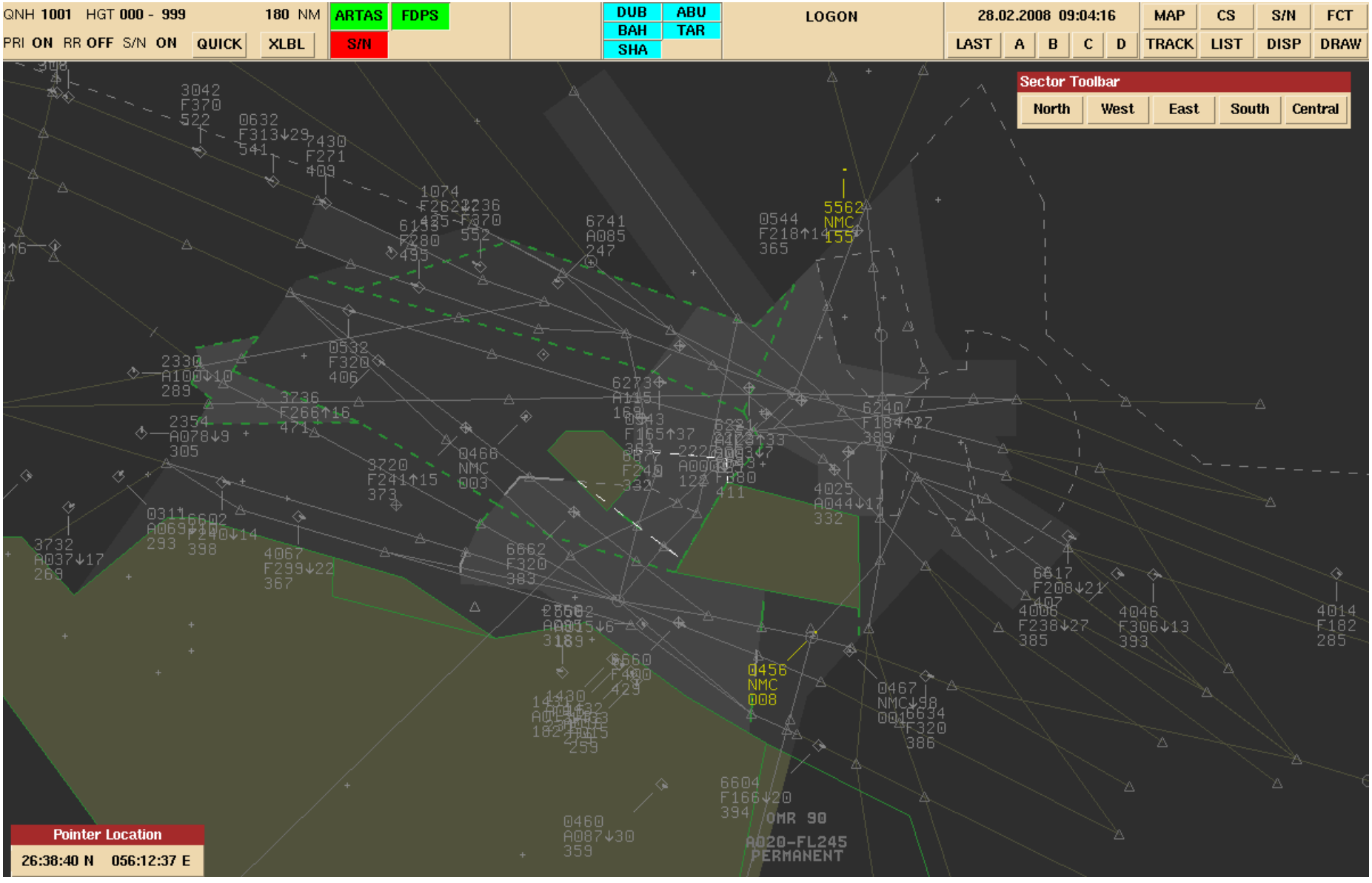
- Entwicklungsprozess (Auszug):
  - Kundenspezifikation
  - Systemspezifikation
  - Software-Design
  - Implementierung
- Problem: Äquivalenz der verschiedenen Ebenen
  - Manuelle Überprüfung aufwendig und fehleranfällig
  - Äquivalenz nicht immer offensichtlich
  - Ausgangsspezifikation nicht immer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

Geht das auch konkreter?

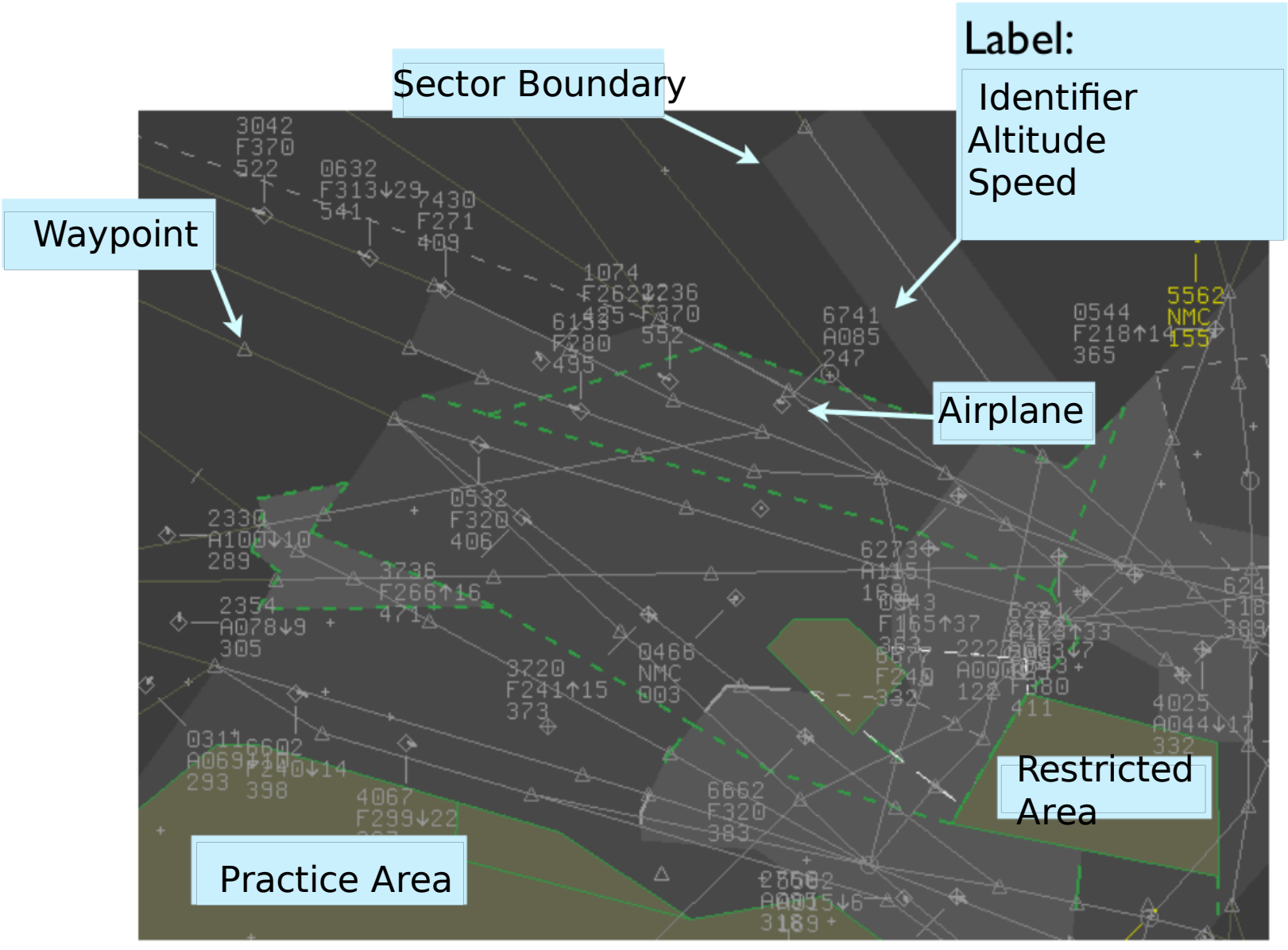
Beispiel: Flugsicherung



Luftlagebild



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

$\Longrightarrow$

Hart-kodierte Lösungen ungeeignet

Lösung: Symbolische Logik

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

Luftraum UAE (II)

Filterlösung: Stelle Flugzeug X genau dann dar, wenn

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

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

Optimierung

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

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

Äquivalenz?

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

gegen

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

Formalisierung in TPTP-Syntax

```
fof(filter_equiv, conjecture, (
% Naive version: Display aircraft in the Abu Dhabi Approach are
% 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))))
  <=>
% Optimized version: Display all aircraft in either Approach, d
% aircraft in upper airspace, except military aircraft in the m
% training region
    (![X]:((uppairspace(X) | dub_app(X) | a_d_app(X)) &
    (~military(X) | ~milregion(X))))).
```

Verifikation

- 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)
  - Analyse: lowairspace(X) oder 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 are
% 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))))
    <=>
% Optimized version: Display all aircraft in either Approach, d
% aircraft in upper airspace, except military aircraft in the m
% training region
    (![X]:((uppairspace(X) | dub_app(X) | a_d_app(X)) &
    (~military(X) | ~milregion(X)))))).
```

Verifikation

- 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)

Fazit

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

Ende Vorlesung 1

Mengenlehre

Definitionen

Definition (Definition)

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

Mathematische Beweise

Definition (Beweis)

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

Mengen

Definition (Menge, Element)

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

- Elemente können beliebige Objekte sein:
  - Zahlen
  - 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:
  - $A = \{2, 3, 5, 7, 11, 13\}$
  - $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$
- Beschreibung („Deskriptive Form“):
  - $A = \{x \mid x \text{ ist Primzahl und } x \leq 13\}$
  - (Allgemein: $\{\text{Ausdruck} \mid \text{ Bedingungen}\}$)
- 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 (Teilmenge)

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 (Mengengleichheit)

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 (Echte Teilmenge)

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.

Notationsalarm: 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
- $\mathbb{N} = \{0, 1, 2, 3 \ldots\}$ (die natürlichen Zahlen)
  - Informatiker (und moderne Mathematiker) fangen bei 0 an zu zählen!
- $\mathbb{N}^+ = \{1, 2, 3, \ldots\}$ (die positiven ganzen Zahlen)
- $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ (die ganzen Zahlen)
- $\mathbb{Q} = \{\frac{p}{q} | p \in \mathbb{Z}, q \in \mathbb{N}^+\}$ (die rationalen Zahlen)
- $\mathbb{R}$, die reellen Zahlen

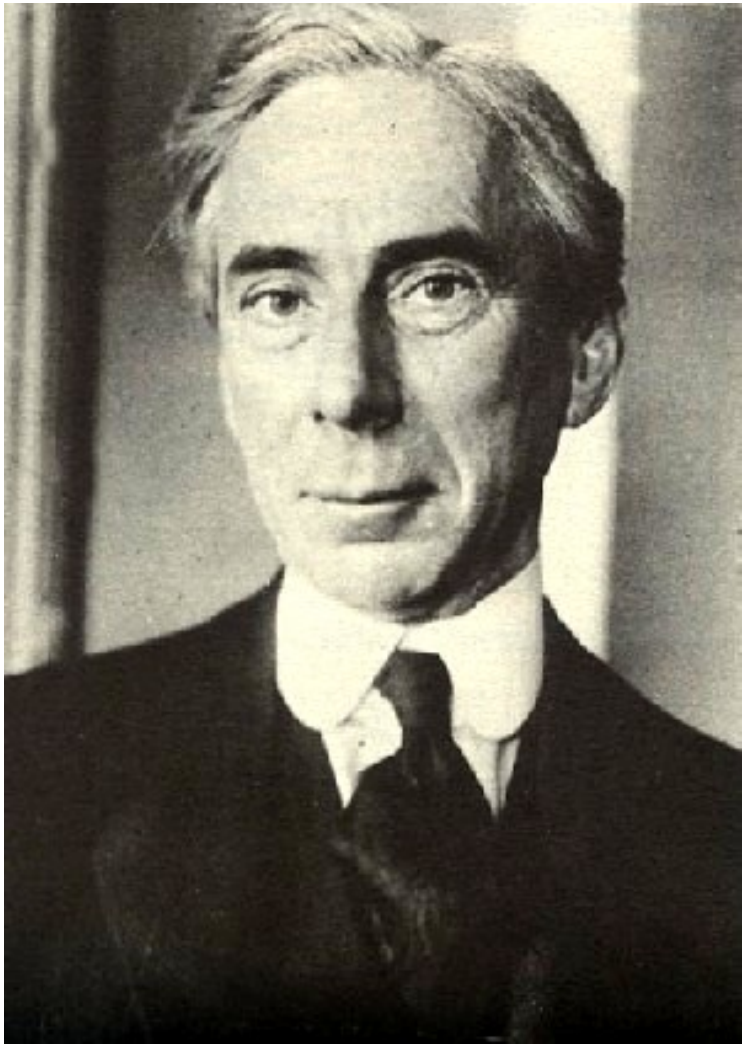
Die ganzen Zahlen hat der liebe Gott gemacht, alles andere ist Menschenwerk.
Leopold Kronecker (1823–1891)

Übung: Mengenbeschreibungen

- Geben Sie formale Beschreibungen *(descriptive Form)* 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*Bertrand Russell (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.
$\ldots$	$\ldots$	$\ldots$
$M_{i+1} =$	$\{M_i\}$	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:

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

Mengenlehre ist die Grundlage der Mathematik (3)

Und bequemere Schreibweise:

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

- Wir führen ein neues Symbol a ein und geben folgende Regeln an:
  - $a(x, 0) = x$
  - $a(x, s(y)) = s(a(x, y))$
- Beispielrechnung:

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

$\ldots$ oder auch $1 + 2 = 3$

Übung: Multiplikation rekursiv

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

Kommentare zur Zahlkonstruktion

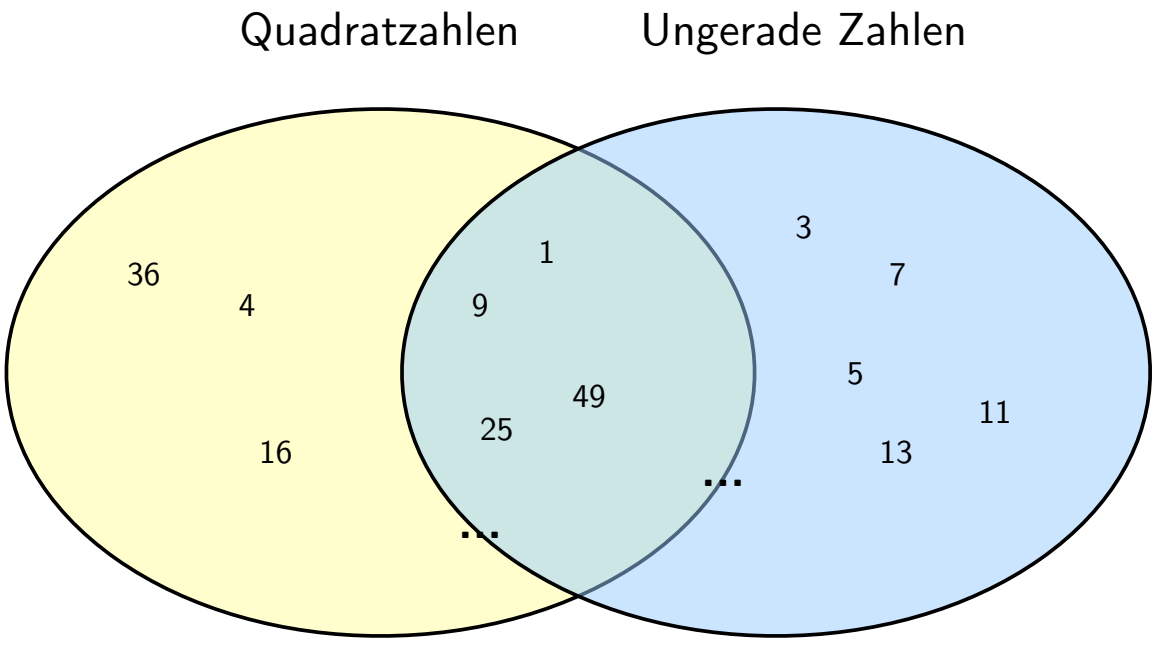
- Wir können mit dieser Zahlenkonstruktion und unseren Regeln rechnen, ohne ein Verständnis von Zahlen zu haben
  - Wir verwenden nur einfachste Definitionen auf Basis des Mengenbegriffs
  - Die Rechnungen sind rein mechanisch
- Diese Konstruktion der natürlichen Zahlen ist induktiv:
  - 0 (oder die leere Menge) ist eine natürliche Zahl
  - Wenn n (d.h. M_n) eine Zahl ist, so auch $s(n)$ (=$\{n\}$ oder $\{M_n\}$)
- Die Definitionen von a und m sind rekursiv
  - Für den Basisfall (in diesem Fall die 0 im zweiten Argument) wird die Lösung direkt angegeben
  - Für den rekursiven Fall (zweites Argument > 0) wird das Problem auf ein kleineres reduziert

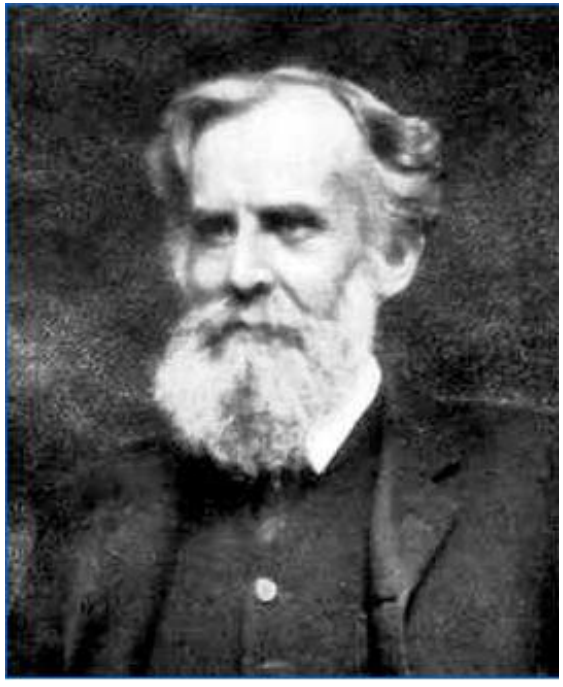
Rekursion (die Lösung eines Problems durch Reduktion auf kleinere Teilprobleme) ist eines der wichtigsten Konzepte der Informatik!

Mengenoperationen

Venn-Diagramme

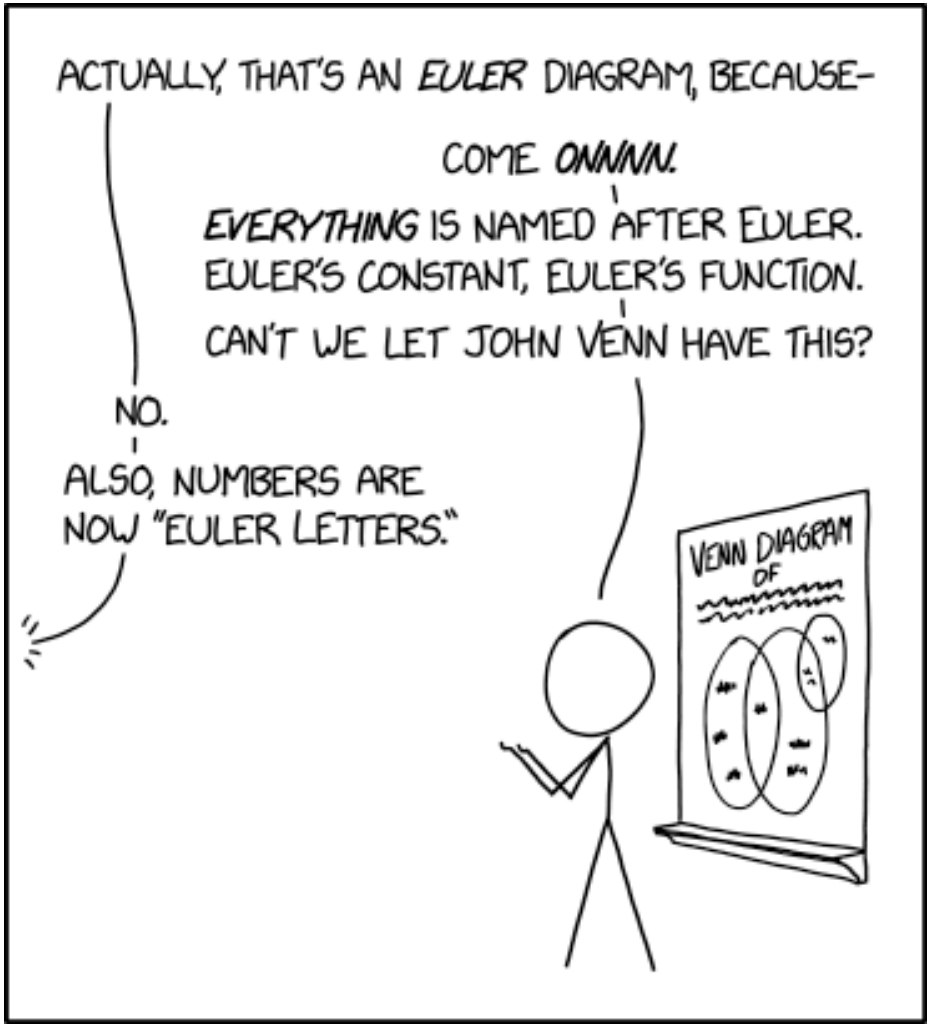
- Graphische Mengendarstellung
  - Mengen sind zusammenhängende Flächen
  - Überlappungen visualisieren gemeinsame Elemente
- Zeigen alle möglichen Beziehungen





John Venn
(1834–1923)

Exkurs: Nicht-Venn-Diagramme



https://xkcd.com/2721/

Leonard Euler (1707–1783)

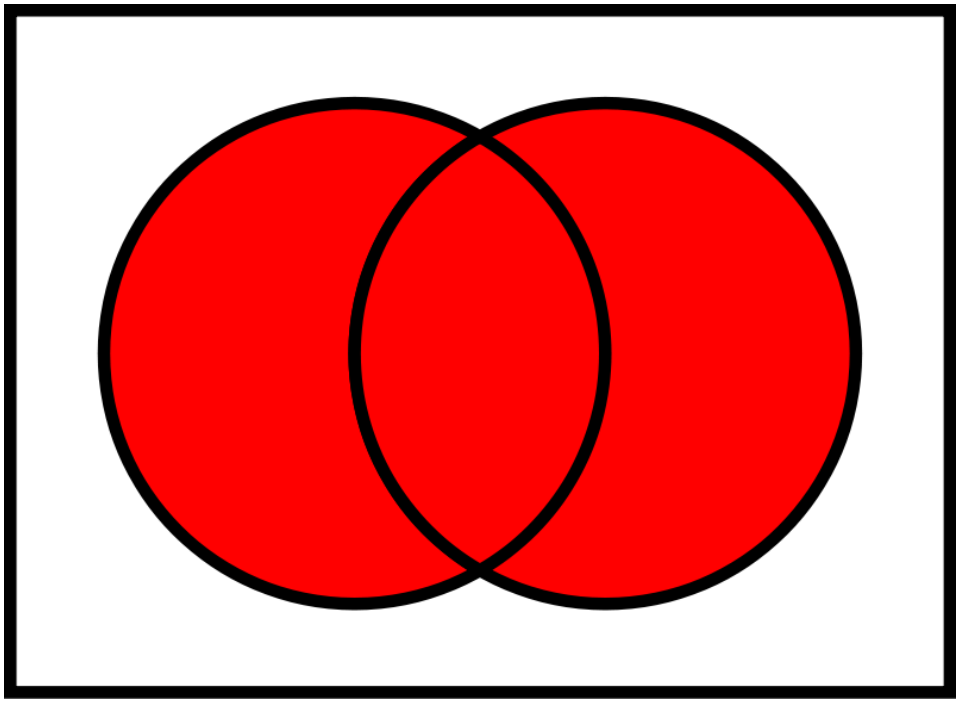
- Venn-Diagramme zeigen *alle* Schnittmengen
- Euler-Diagramme zeigen nur nicht-leere Schnittmengen
  - Oft einfacher zu zeichnen
  - Oft weniger informativ
  - Use-case z.B.: Inklusionen

Mengenoperationen

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

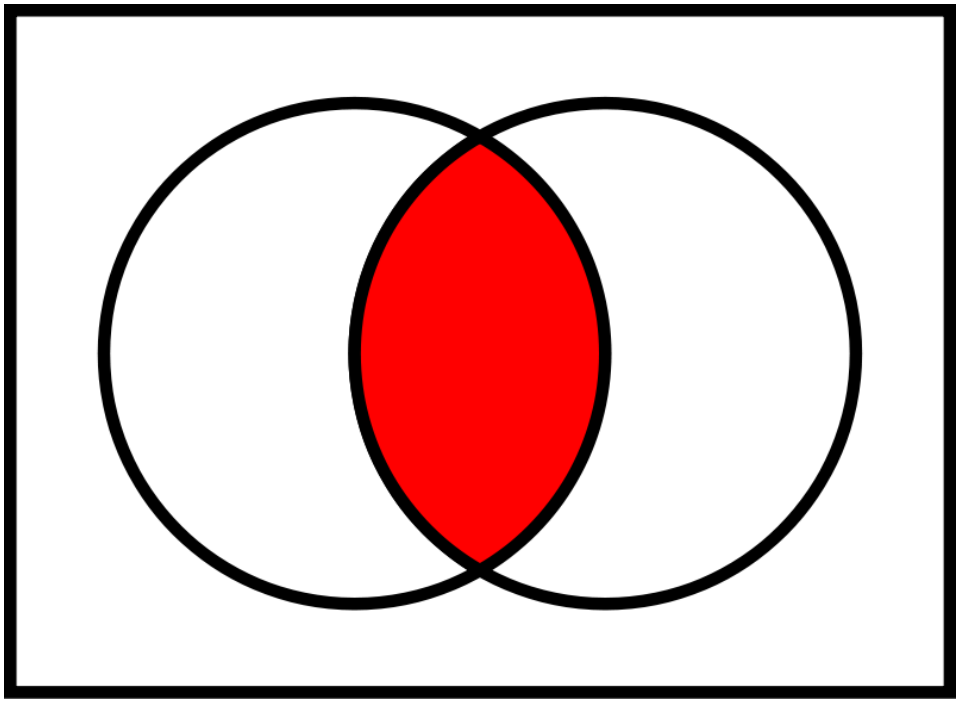
- Vereinigung
- Schnitt
- Differenz
- Symmetrische Differenz
- Komplement

Vereinigungsmenge



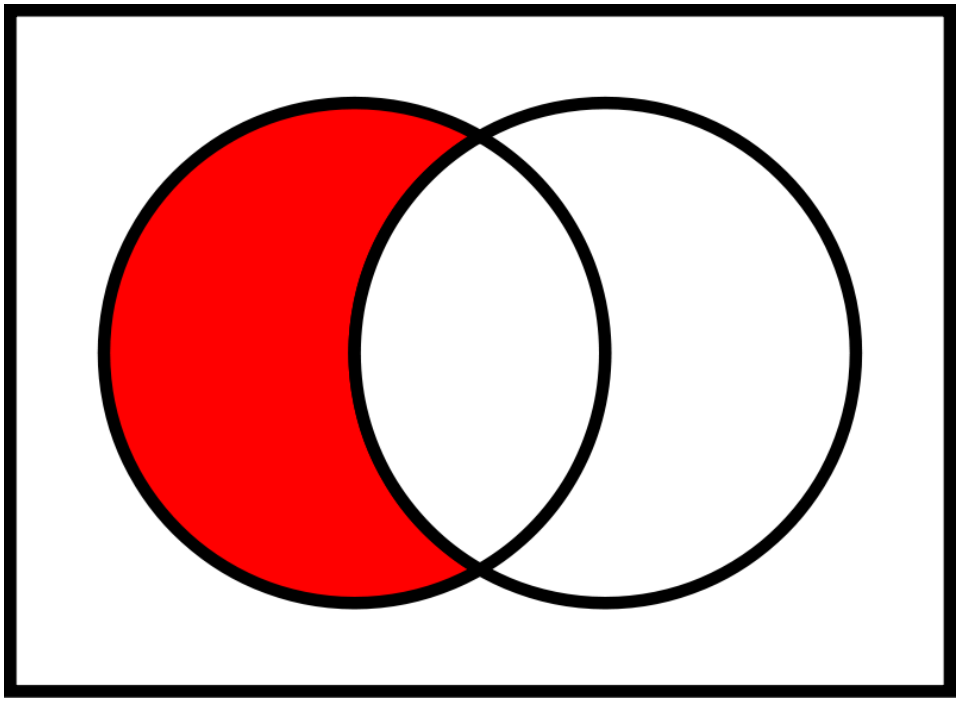
- $M_1 \cup M_2 = \{x | x \in M_1 \text{ oder } x \in M_2\}$
- $x \in M_1 \cup M_2$ gdw. $x \in M_1$ oder $x \in M_2$

Schnittmenge



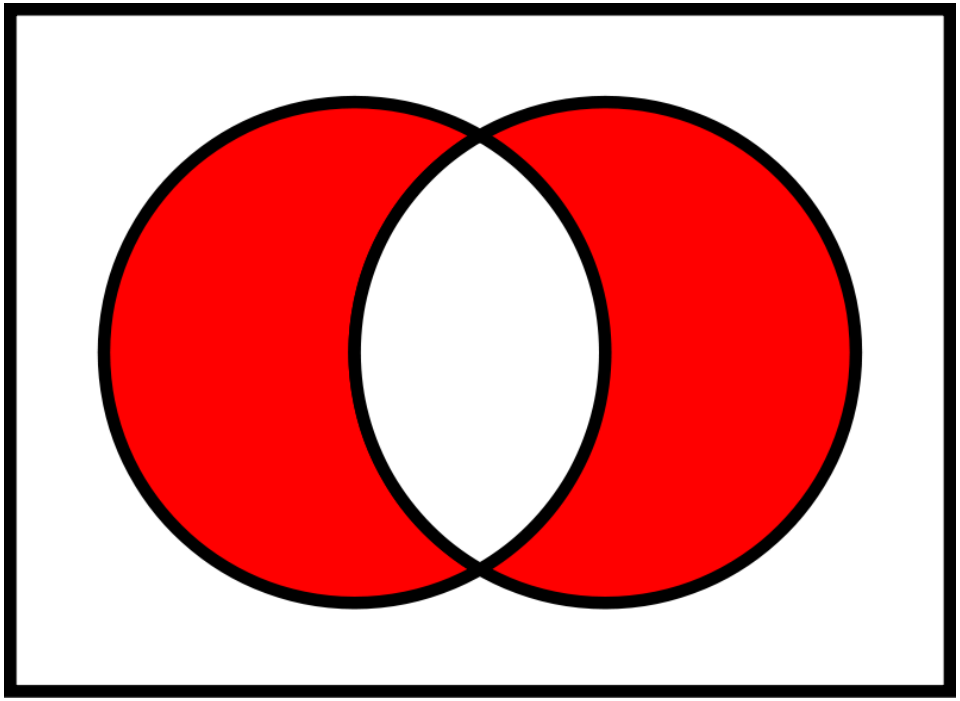
- $M_1 \cap M_2 = \{x | x \in M_1 \text{ und } x \in M_2\}$
- $x \in M_1 \cap M_2$ gdw. $x \in M_1$ und $x \in M_2$

Differenz



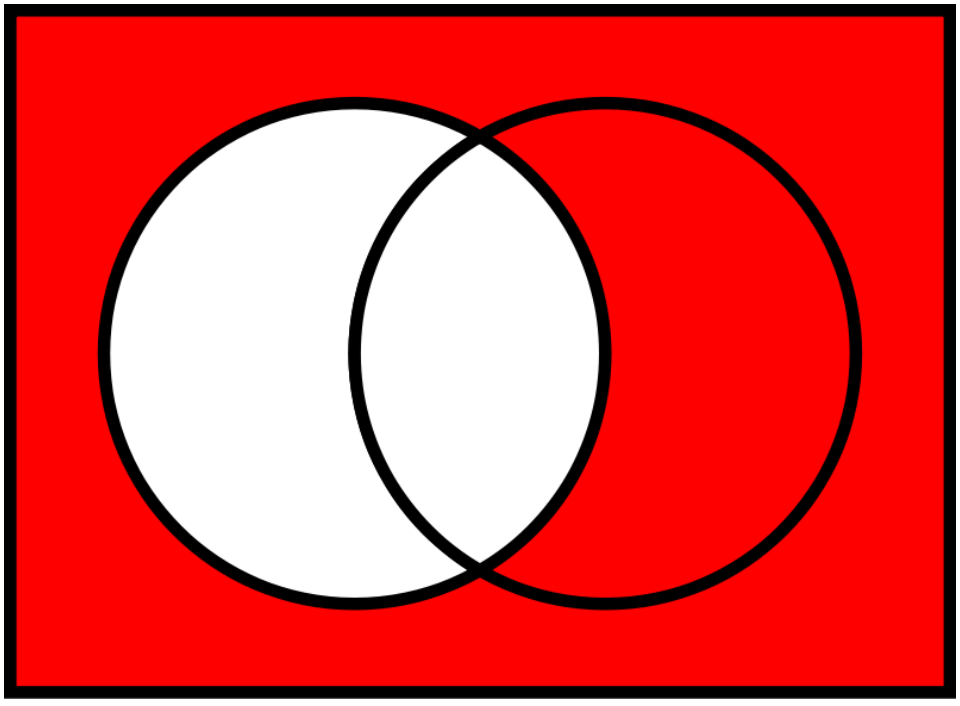
- $M_1 \setminus M_2 = \{x | x \in M_1 \text{ und } x \notin M_2\}$
- $x \in M_1 \setminus M_2$ gdw. $x \in M_1$ und $x \notin M_2$
- Sprechweise: "M_1 ohne M_2"

Symmetrische Differenz



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

Komplement



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

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

Übung Mengenoperationen

- Sei $T = \{1, 2, \ldots, 12\}$, $M_1 = \{1, 2, 3, 4, 5, 6, 7, 8\}$, $M_2 = \{2, 4, 6, 8, 10, 12\}$. Berechnen Sie die folgenden Mengen und visualisieren Sie diese.
  - $M_1 \cup M_2$
  - $M_1 \cap M_2$
  - $M_1 \backslash M_2$
  - $M_1 \triangle M_2$
  - $\overline{M_1}$ 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 (*deskriptive Form*) und eine umgangssprachliche Charakterisierung des Ergebnisses an.
  - $M_1 \cup M_2$
  - $M_1 \cap M_2$
  - $M_1 \backslash \overline{M_2}$
  - $M_1 \backslash \overline{M_2}$
  - $M_1 \triangle M_2$

Diskussion: Übung Mengenoperationen

- Sei $T = \mathbb{N}$, $M_1 = \{3i \mid i \in \mathbb{N}\}$, $M_2 = \{2i + 1 \mid i \in \mathbb{N}\}$. Berechnen Sie die folgenden Mengen. Geben Sie jeweils eine mathematische und eine umgangssprachliche Charakterisierung des Ergebnisses an.
  - $M_1 \cup M_2$
    - 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
    - $M_1 \triangle M_2 = \{6i \mid i \in \mathbb{N}\} \cup \{6i + k \mid i \in \mathbb{N}, k \in \{1, 5\}\}$ $= \{6i + k \mid i \in \mathbb{N}, k \in \{0, 1, 5\}\}$

Ende Vorlesung 2

Kartesisches Produkt

Definition (Kartesisches Produkt)

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

- $M_1 \times M_2$ ist eine Menge von Paaren oder 2-Tupeln
- Verallgemeinerung: $M_1 \times M_2 \ldots \times M_n = \{(x_1, x_2, \ldots, x_n) \mid x_i \in M_i\}$ 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

Kartesisches Produkt: Spezialfälle

Definition (n-Tupel über einer Menge)

Sei M eine beliebige Menge. Dann ist $M^n = \underbrace{M \times \ldots \times M}_{n \text{ Mal}}$ die Menge der

n-Tupel über M.

- Spezialfall: $n = 1$: $M^1 = \{(x) \mid x \in M\}$
  - M^1 ist die Menge der 1-Tupel über M
  - M^1 enthält genau ein Element für jedes Element aus M
  - Oft werden M^1 und M miteinander identifiziert (obwohl sie strikt gesehen verschieden sind)
- Spezialfall: $n = 0$: $M^0 = \{()\}$
  - M^0 enthält genau ein Element: Das leere Tupel ()

Potenzmengen

Definition (Potenzmenge)

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: Kartesisches Produkt und Potenzmenge

- Sei $M_1 = \{1, 2, 3, 4, 5, 6, 7\}$, $M_2 = \{2, 4, 6, 8, 10\}$.
  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 $2^{M_1 \cup M_2}$ berechnen?

Lösung: $M_1 \times M_1$

$M_1 \times M_1 =$
$\{(1,1),(1,2),(1,3),(1,4),(1,5),(1,6),(1,7),$
$(2,1),(2,2),(2,3),(2,4),(2,5),(2,6),(2,7),$
$(3,1),(3,2),(3,3),(3,4),(3,5),(3,6),(3,7),$
$(4,1),(4,2),(4,3),(4,4),(4,5),(4,6),(4,7),$
$(5,1),(5,2),(5,3),(5,4),(5,5),(5,6),(5,7),$
$(6,1),(6,2),(6,3),(6,4),(6,5),(6,6),(6,7),$
$(7,1),(7,2),(7,3),(7,4),(7,5),(7,6),(7,7)\}$

Lösung: $M_1 \times M_2$

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

Lösung: $M_2 \times M_1$

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

Lösung: Potenzmenge

Definition (Potenzmenge)

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

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

Übung: M^3

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

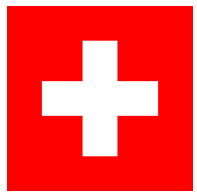
Mengenalgebra

- 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
  - $(\{0, s(0), s(s(0)), \ldots, \}, a, m)$ ist eine Termalgebra
- Mengenalgebra:
  - Die Trägermenge der Algebra ist die Menge der Mengen über einem gemeinsamen Universum
  - Die Operatoren sind $\cup, \cap, \bar{}, \ldots$

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)

Augustus De Morgan
(1806-1871)

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

- Distributivgesetze
  - $M_1 \cup (M_2 \cap M_3) = (M_1 \cup M_2) \cap (M_1 \cup M_3)$
  - $M_1 \cap (M_2 \cup M_3) = (M_1 \cap M_2) \cup (M_1 \cap M_3)$
- Komplementäre 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
  - $\overline{\overline{M_1}} = M_1$

Relationen

Allgemeine Relationen

Definition (Relationen)

Seien $M_1, M_2, \ldots M_n$ Mengen. Eine (n-stellige) Relation R über $M_1, M_2, \ldots M_n$ ist eine Teilmenge des kartesischen Produkts der Mengen, also $R \subseteq M_1 \times M_2 \times \ldots \times M_n$ (oder äquivalent $R \in 2^{M_1 \times M_2 \times \ldots \times M_n}$).

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

Übung

- 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?

Homogene Relationen, Binäre Relationen

Definition (homogene Relation, binäre Relation)

Sei R eine Relation über $M_1, M_2, \ldots M_n$.

- R heißt homogen, falls $M_i = M_j$ für alle $i, j \in \{1, \ldots, n\}$.
- R heißt binär, falls $n = 2$.
- R heißt homogene binäre Relation, falls 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 Relationen

- Beispiele für homogene binäre Relationen:
  - $=$ über $\mathbb{N}$
    - $=\ =\{(0,0),(1,1),(2,2),\ldots\}$
  - $<$ über $\mathbb{Z}$
    - $<\ =\{(i,i+j) \mid i \in \mathbb{Z}, j \in \mathbb{N}^+\}$
    - $(1,2) \in <, (7,42) \in <, (0,666) \in <$
    - $\{(0,1),(0,2),(0,3)\} \subseteq <$
  - $\neq$ über $\{w \mid w$ 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 (linkstotal, rechtseindeutig)

Sei R eine binäre Relation über A, 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 (reflexiv, symmetrisch, transitiv, Äquivalenzrelation)

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

- Ordnungen sind Beispiele für transitive Relationen
- Äquivalenzrelationen teilen Mengen in Klassen von "gleichen" (oder zumindest, wörtlich, "gleichwertigen") Elementen ein

Übung: Eigenschaften von Relationen

- 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

Ende Vorlesung 3

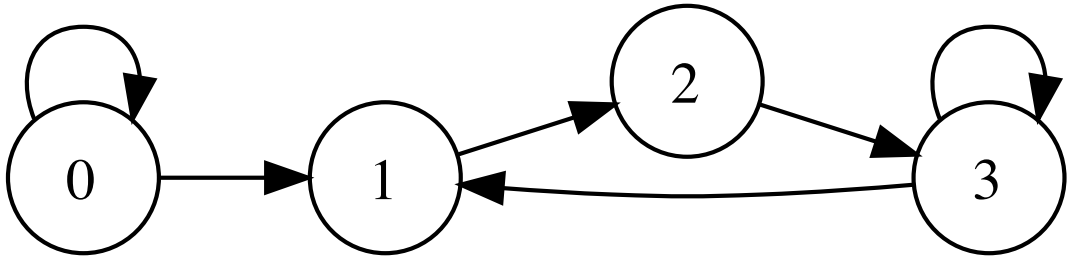
Darstellung von Relationen: Mengendarstellung

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)

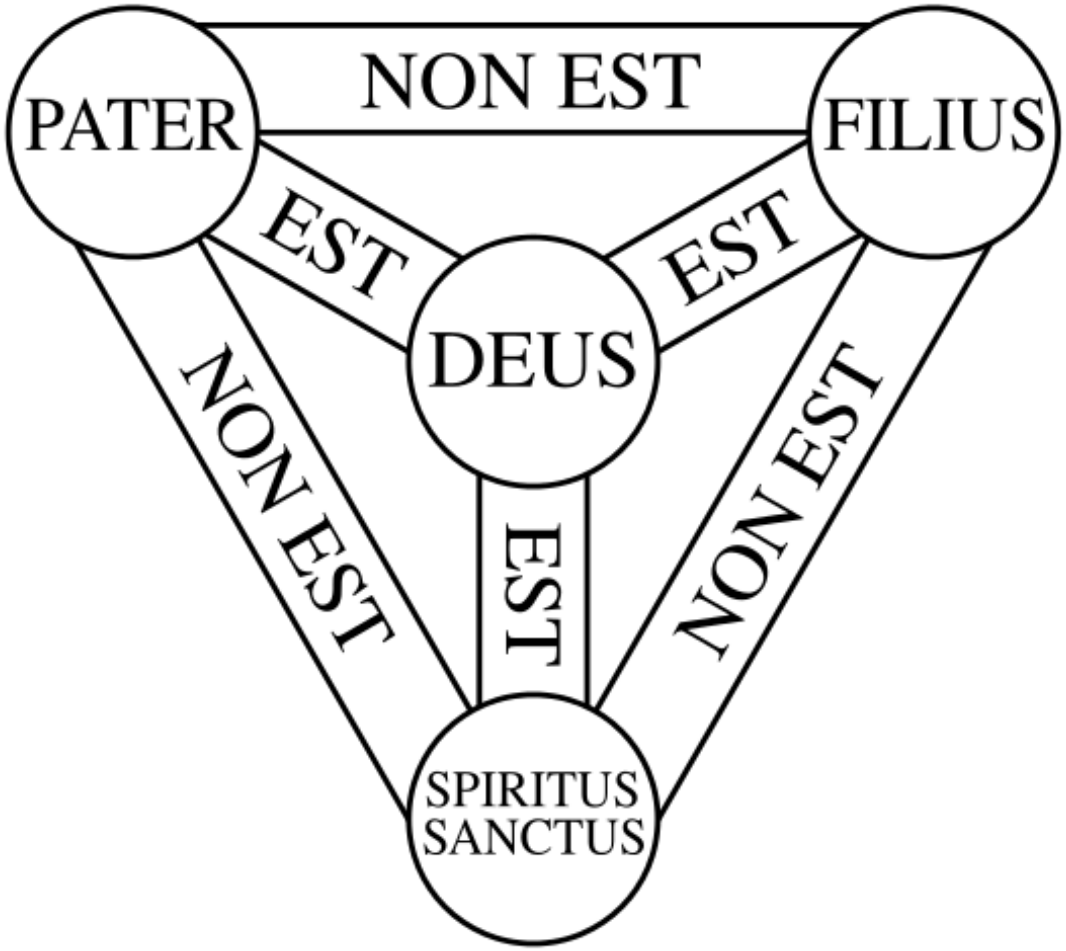
Darstellung von (endlichen) Relationen: Graphdarstellung

- 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



- Diskutieren 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	1	0	1

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

- 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 ;-)

Die inverse Relation

Definition (Inverse Relation)

Sei 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 Gleicheitsrelation gilt: $=^{-1} = =$ (und das ist kein Tippfehler - lies: „Die inverse Relation der Gleichheitsrelation ist wieder die Gleichheitsrelation“)

Verknüpfung von Relationen

Definition (Relationsprodukt)

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

- $R_2 \circ R_1$ spricht sich „R_2 nach R_1“
- Es gilt nicht immer $R_1 \subseteq R_2 \circ R_1$ oder $R_2 \subseteq R_2 \circ R_1$
- Schreibweise gelegentlich auch $R_1 R_2$ oder $R_1; R_2$ statt $R_2 \circ R_1$
  - Bei dieser Schreibweise wird die zuerst anzuwendendende Relation auch als erste geschrieben
  - Die Schreibweise mit $\circ$ ist bei Verknüpfung von Funktionen intuitiver
- Beachte:
  - Damit $R_1 \subseteq A \times B$ und $R_2 \subseteq C \times D$ sinnvoll verknüpft werden können, sollte $B = C$ sein - mindestens aber $B \cap C \neq \emptyset$ (sonst ist das Ergebnis automatisch leer).
  - Bei homogenen Relationen ist das immer gegeben

Identitätsrelation, Relationenalgebra

Definition (Identitätsrelation)

Sei M eine Menge. Dann ist $\mathrm{id}_M = \{(x, x) \mid x \in M\}$ (kurz: id) die Identitätsrelation über M.

- Es gilt $R \circ \mathrm{id} = \mathrm{id} \circ R = R$ für beliebige Relationen R (über M)
- id ist ein neutrales Element in Bezug auf die Relationenverknüpfung

Die Menge aller homogenen binären Relationen über M ist $2^{M \times M}$. $(2^{M \times M}, \circ, \mathrm{id}_M)$ ist eine algebraische Struktur. Konkret: $(2^{M \times M}, \circ, \mathrm{id}_M)$ ist ein Monoid mit der assoziativen Verknüpfung $\circ$ und dem neutralen Element id.

Beispiel zur Relationenalgebra

- Sei H die Menge aller Menschen, die jemals gelebt haben
- Sei $V \subseteq H \times H$ die Vater-Relation (also $V = \{(x, y) \mid y \text{ ist Vater von } x\}$
- Sei analog $M \subseteq H \times H$ die Mutter-Relation
- Dann können wir die Großmutter-Relation G wie folgt beschreiben:

$$G = M \circ (M \cup V)$$

Mehrfachanwendung von Relationen

Definition (Potenzierung von Relationen)

Sei R eine Relation über M. Wir definieren:

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

- Beispiel: Betrachte $M = \{a, b, c\}$ und $R = \{(a, b), (a, c), (b, c)\}$
  - $R^0 = \{(a, a), (b, b), (c, c)\}$
  - $R^1 = R$
  - $R^2 = \{(a, c)\}$
  - $R^3 = \{\}$
- Beispiel: Betrachte $S = \{(x, x + 1) \mid x \in \mathbb{Z}\}$
  - $S^0 = \{(x, x) \mid x \in \mathbb{Z}\} (= =)$
  - $S^2 = \{(x, x + 2) \mid x \in \mathbb{Z}\}$
  - ...
  - $S^n = \{(x, x + n) \mid x \in \mathbb{Z}\}$

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
- $S \circ S$
- $S \circ R$

Beachte:

- Die Potenzierung von Relationen R^n ist nur für $n \in \mathbb{N}$ definiert.
- R^{-1} ist die inverse Relation und von uns unabhängig definiert.

Erweiterung von Relationen - “Hüllenbildung”

- Manchmal haben wir Relationen, die “notwendige Paare” enthalten, aber nicht alle gewünschten Eigenschaften haben
- In dem Fall können wir die Relation erweitern, bis sie die gewünschten Eigenschaften hat
- Die minimale Erweiterung einer Relation R mit Eigenschaft X heißt dann die *“X”- Hülle von R*
- Für eine solche Hülle H gilt:
  - $R \subseteq H$
  - Wir können kein Element aus H entfernen, ohne entweder $R \subseteq H$ oder Eigenschaft X zu verletzen - die Hülle ist die kleinste Erweiterung von R mit Eigenschaft X.

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

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

Reflexive, symmetrische, transitive Hüllen

Definition (Reflexive, symmetrische, transitive Hüllen)

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.

Übung: Hüllenbildung

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.

Lösung | Ende Vorlesung 4

Funktionen

Definition

Seien M, N Mengen.

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

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

Bild einer Menge

Definition (Bild, Urbild)

Sei M, N Mengen und $f : M \to N$ eine Funktion. Sei $M_0 \subseteq M$ und $N_0 \subseteq N$

- $f(M_0) = \{y \in N \mid \exists x \in M_0 : f(x) = y\}$ ist das Bild von M_0 unter f.
- $\{x \in M \mid \exists y \in N_0 : f(x) = y\}$ ist das Urbild von N_0 unter f.

- Für das Bild erweitern wir f zu einer Funktion $2^M \to 2^N$
- Das Urbild einer Menge umfasst alle Werte, die auf diese abgebildet werden
  - Gelegentlich wird als Schreibweise für das Urbild von N_0 unter f auch $f^{-1}(N_0)$ verwendet.

Eigenschaften von Funktionen

Definition (Injektiv, surjektiv, bijektiv)

Sei $f : M \to N$ eine (totale) Funktion.

- f heißt surjektiv, wenn $\forall y \in N \exists x \in M : f(x) = y$
- f heißt injektiv, wenn $\forall x, z \in M : f(x) = y$ und $f(z) = y \rightsquigarrow x = z$
- f heißt bijektiv (oder „1-zu-1“), wenn f injektiv und surjektiv ist

- Anmerkungen:
  - Wenn f surjektiv ist, so gilt $f(M) = N$.
  - Wenn f injektiv ist, so ist f^{-1} rechtseindeutig (also eine (partielle) Funktion).
  - Im Prinzip kann man die Begriffe *injektiv* und *surjektiv* so auch auf partielle Funktionen übertragen. Für Bijektivität wird dann zusätzlich (Links-)Totalität gefordert.

Übung

- Betrachten Sie die folgenden Funktionen:
  - $f_1 : \mathbb{Z} \to \mathbb{N}, x \mapsto |x|$
  - $f_2 : \mathbb{N} \to \mathbb{N}, x \mapsto |x|$
  - $f_3 : \mathbb{N} \to \mathbb{N}, x \mapsto 2x$
  - $f_4 : \mathbb{N} \times \mathbb{N} \to \mathbb{N}, (x, y) \mapsto x + y$
- Welche der Funktionen sind surjektiv ("jeder wird getroffen"), injektiv ("niemand wird mehrfach getroffen"), 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 $\{6, 8\}$?

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

Kardinalität

- 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 $f : M \to N$ existiert
- Für endliche Mengen ist $|M|$ die Anzahl der Elemente in M
- Eine unendliche Menge M heißt abzählbar, wenn Sie die selbe Kardinalität wie $\mathbb{N}$ hat
  - Das läßt sich zum Beispiel zeigen, indem man ein Verfahren angibt, das alle Elemente von M in einer klaren Reihenfolge aufzählt

Kardinalität: Beispiele

- Abzählbar (gleichmächtig zu $\mathbb{N}$) sind:
  - $\mathbb{Z}$ (siehe Übung)
  - $\{3i \mid i \in \mathbb{N}\}$
  - $\{p \mid p \text{ ist Primzahl}\}$
  - Menge der Worte über einem gegebenen endlichen Alphabet
    - Sortiere nach Länge, dann alphabetisch, und numeriere durch
  - Menge aller syntaktisch korrekten Programme
    - Sortiere nach Länge, dann alphabetisch, und numeriere durch
  - $\mathbb{N} \times \mathbb{N}$ (*"Schwalbenflug"*)
  - $\mathbb{Q}$ (Ditto)
- Echt mächtiger als $\mathbb{N}$ sind z.B.:
  - $\mathbb{R}$ (Cantors Diagonalisierung)
  - $2^{\mathbb{N}}$ (die Menge aller Teilmengen von $\mathbb{N}$)
  - $2^{\mathbb{N}\times\mathbb{N}}$ (die Menge aller binären Relationen über $\mathbb{N}$)
  - Die Menge aller Funktionen von $\mathbb{N}$ nach $\mathbb{N}$

Übung: Kardinalität

- Zeigen Sie: $\mathbb{Z}$ ist abzählbar
  - Verfahren: Geben Sie eine totale bijektive Funktion $\mathbb{N} \to \mathbb{Z}$ an
- Für endliche Mengen M gilt: $|2^M| = 2^{|M|}$
  - Verfahren: Vollständige Induktion über $|M|$

Ende Vorlesung 5

Funktionales Programmieren mit Scheme

Einstimmung

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/Scheme

- LISP: **Lis**t **P**rocessing
  - 1958 von John McCarthy entworfen
  - Realisiert Church's λ-Kalkül
  - Implementierung durch Steve Russell
- Wichtige Dialekte:
  - Scheme (seit 1975, aktueller Standard R^7RS, 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)

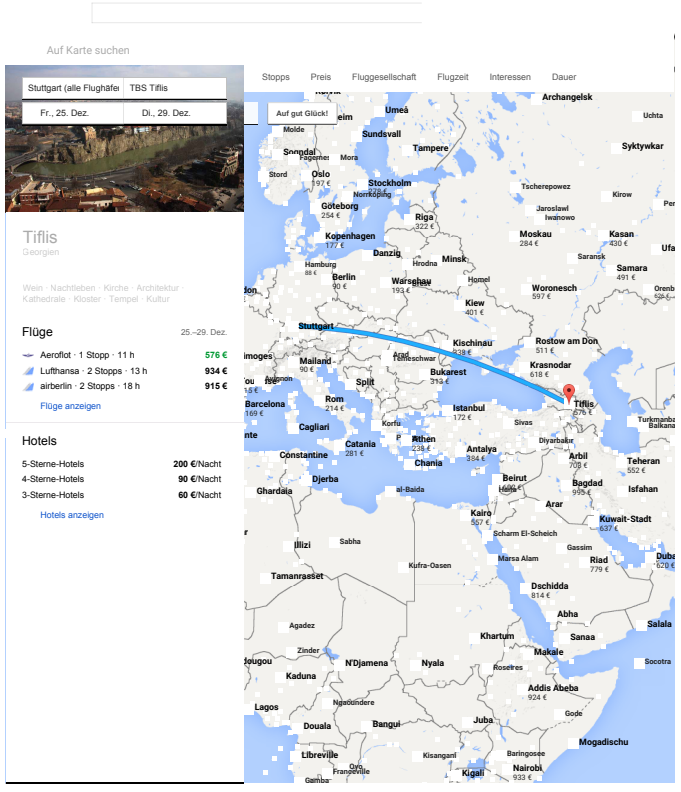
Steve Russell (1937–)

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/DCTP/Gandalf/ACL2
  - AM-Reasoner/Heurisco/Cyc
  - Viele Expertensysteme
- Scripting
  - Emacs (ELisp)
  - AutoCAD (AutoLISP)
  - GIMP (SIOD/TinyScheme)
  - LilyPond/gdb/GnuCash (Guile)
  - Audacity (Nyquist)
- Sonstiges
  - (Yahoo Stores, Reddit)
  - ITA Software (Google Flights)
  - Grammarly



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, ..., e_n s-expressions sind, dann auch `(`e_1 e_2 `...`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)))))`

Ein funktionales Beispiel

- Die Fakultät einer natürlichen Zahl n ist das Produkt der Zahlen von 1 bis n: $fak(n) = \prod_{i=1}^{n} i$
  - $fak(3) = 6, fak(5) = 120, \ldots$
- 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))))
  )
)
```

Scheme in top-secret places

Through some clever security hole manipulation, I have been able to break into the NSA computers and acquire the Scheme code of the PRISM project. Here is the last page (tail -10) of it to prove that I actually have the code:

```
))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))
))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))
))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))
))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))
))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))
))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))
))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))
))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))
))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))
))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))
```

Updated from a rec.houmor.funny joke, `http://www.netfunny.com/rhf/jokes/90q2/lispcode.html`

Read-Eval-Print

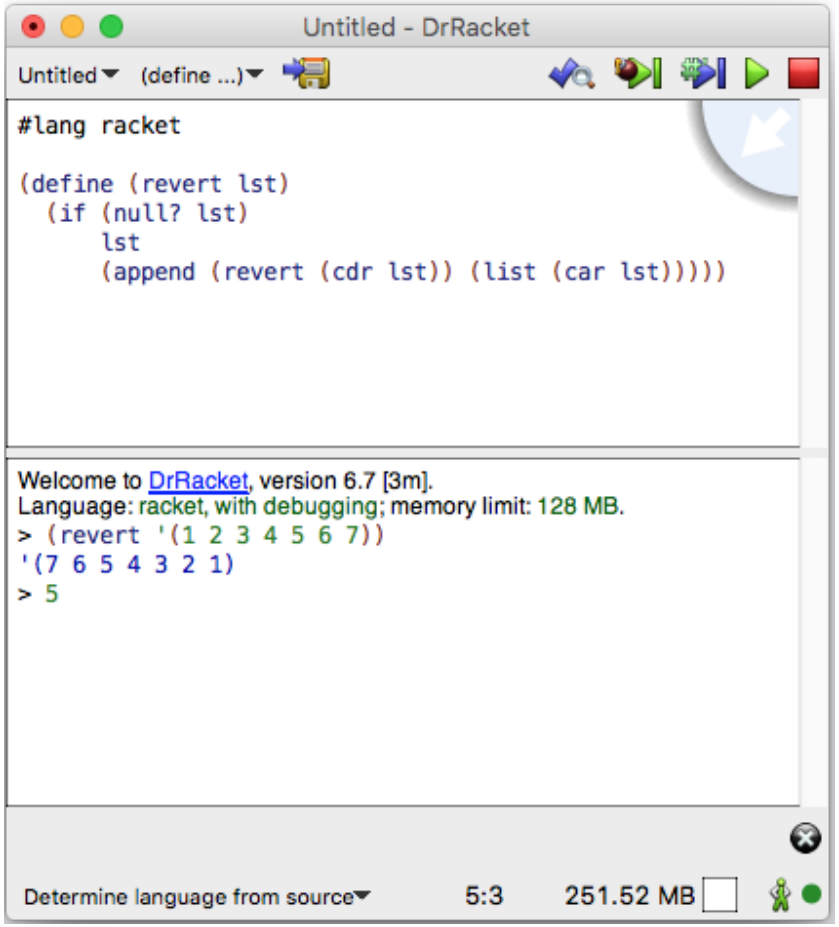
LISP programmers know the value of everything and the cost of nothing.

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)
```

Read-Eval-Print in DrRacket



- Zwei zentrale Bereiche:
  - *Definitions window* - Programmeditor
    - Für permanente Definitionen ("Das Programm")
    - Wird erst durch `run` ▶ in den Interpreter geladen und ausgewertet
    - Inhalt kann abgespeichert werden
  - *Interactions window* - Interaktiver Interpreter
    - Ausdrücke werden sofort ausgewertet

Übung: Hello World

```
;; Funktion hello, die "Hello World ausgibt
(define (hello)
  (display "Hello-World")
  (newline)
  )

;; Aufruf der Funktion
(hello)
```

- Führen Sie das Programm in DrRacket aus
  - Selektieren Sie Racket als Sprache
  - Bringen Sie das Programm in den Definitionsbereich
  - `run`▶
- Definieren Sie die Fakultätsfunktion und berechnen Sie `(fak 10)` und `(fak 30)`
  - Fällt Ihnen etwas auf?

Scheme-Semantik: 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
. . hallo: undefined;
 cannot reference undefined identifier
```

  - Auswerten von (normalen) Listen:
    - Zuerst werden (in beliebiger Reihenfolge) alle Listenelemente ausgewertet
    - Dann wird das erste Ergebnis als Funktion betrachtet und diese mit den Werten der anderen Elemente als Argument aufgerufen

Beispiel

Auswerten von (normalen) Listen:

- Zuerst werden (in beliebiger Reihenfolge) alle Listenelemente ausgewertet
- Dann wird das erste Ergebnis als Funktion betrachtet und diese mit den Werten der anderen Elemente als Argument aufgerufen

```
> +
==> #<procedure:+>
> 17
==> 17
> (* 3 7)
==> 21
> (+ 17 (* 3 7))
==> 38
```

Besonderheiten

- Problem:

  ```
  ( i f  (= 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* *expr*$_1$ *expr*$_2$`)`
  - *tst* wird auf jeden Fall ausgewertet
  - Ist der Wert von *tst* nicht `#f`, so wird (nur) *expr*$_1$ ausgewertet, das Ergebnis ist der Wert des gesamten `if`-Ausdrucks
  - Sonst wird (nur) *expr*$_2$ 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 *formalen Parametern*, 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`)

Gleichheit, Grundrechenarten

- `(=` $number_1$ `...`$number_n$`)`
  - `#t`, falls alle Werte gleich sind, `#f` sonst
  - `> (= 1 1) ==> #t`
- `(equal?` obj_1 `...`obj_n`)`
  - `#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)

Lisp Jedi

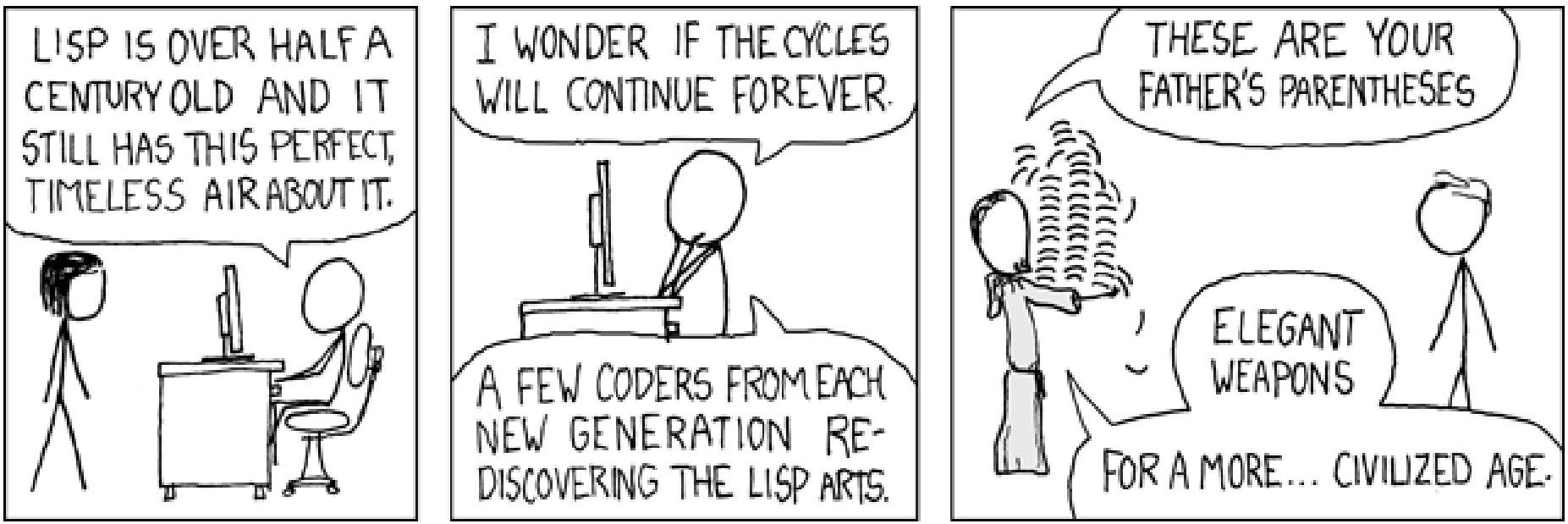


Image credit: `http://xkcd.com/297/`

Ende Vorlesung 6

LISt Processing

- (Verschachtelte) Listen sind das definierende Element von LISP
  - ...als Syntax für Programme
  - ...als Standard-Datenstruktur für Mengen/Sammlungen o.ä.
  - ...zum Kodieren von rekursiven Datenstrukturen
- Typische Fallunterscheidung für Listen:
  - Fall 1: Die *leere Liste* - in Scheme `'()`
  - Fall 2: Eine nicht-leere Liste (bestehend aus einem *ersten Element* und einer *Restliste*)

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` $list_1$ $list_2$`)`
  - Hänge $list_1$ und $list_2$ 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_1 `...`obj_n`)`
  - Gib die Liste `(`obj_1 `...`obj_n`)` zurück

car und cdr

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 + Länge der Liste ohne das erste Element

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

- `(len '(1 2 3)) = (+ 1 (len '(2 3)))`
  `= (+ 1 (+ 1 (len '(3)))) = (+ 1 (+ 1 (+ 1 (len '()))))`
  `= (+ 1 (+ 1 (+ 1 0)) = 3`

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

Funktionale Programmierung?

- Ein Programm besteht aus Funktionen
  - Die Ausführung entspricht der Berechnung eine 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

Rekursion

- 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? element lst)
  (if (null? lst)
      #f
      (if (equal? element (car lst))
          #t
          (is-element? element (cdr lst)))))
```

- 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? element lst)
  (if (null? lst)
      #f
      (if (equal? element (car lst))
          #t
          (is-element? element (cdr lst)))))
```

- Standard-Muster: Bearbeite alle Elemente einer Liste
  - Bearbeite erstes Element: `(equal? element (car lst))`
  - Bearbeite Rest: `(is-element? element (cdr lst))`
- 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)
==> ()
```

Übung: Mengenlehre in Scheme

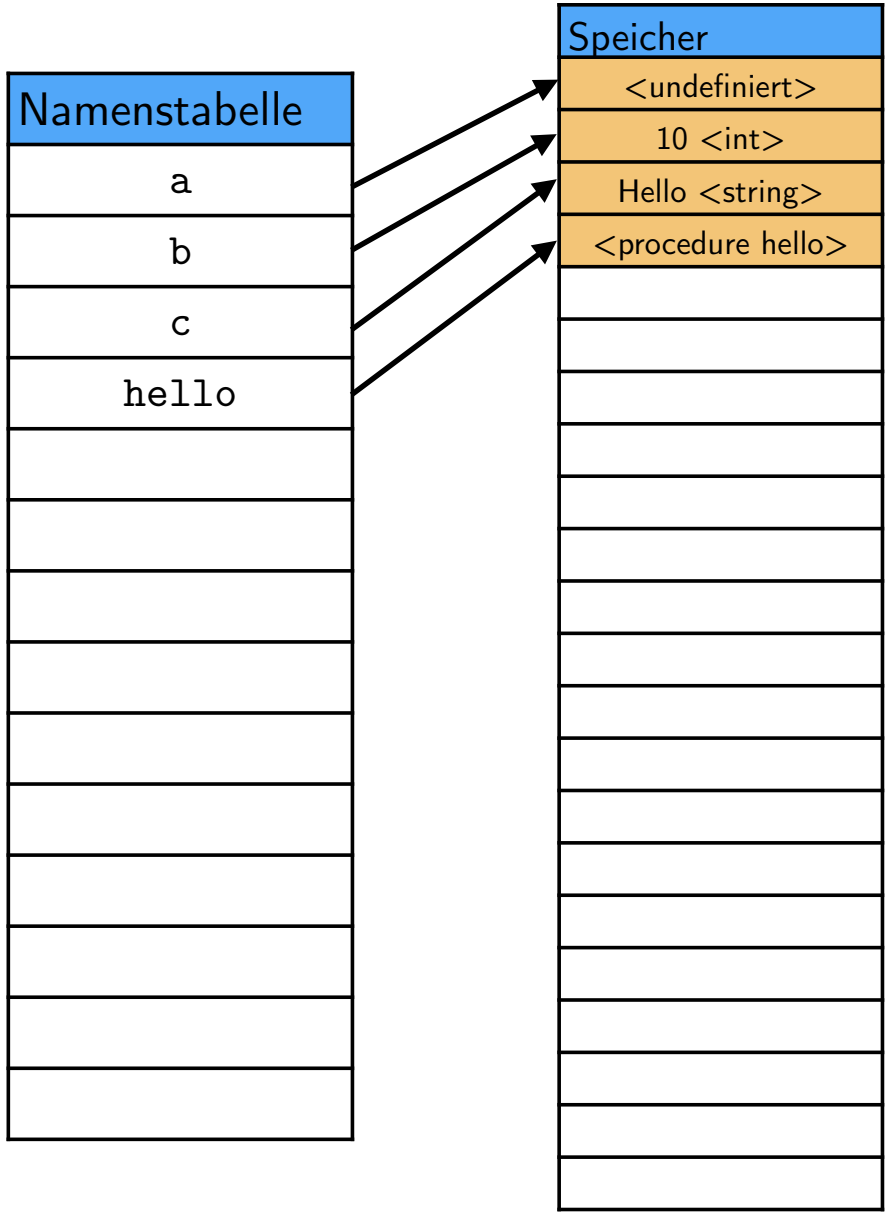
- Repräsentieren Sie im folgenden Mengen als Listen
- Erstellen Sie Scheme-Funktionen für die folgenden Mengen-Operationen:
  - Einfügen:
    - `(insert 4 '(1 2 3)) ==> (1 2 3 4)` (Reihenfolge egal)
    - `(insert 2 '(1 2 3)) ==> (1 2 3)`
  - Vereinigung:
    - `(union '(1 2 3) '(3 4 5)) ==> (1 2 3 4 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 binären Relationen realisiert!

Speichermodell 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 eine neue Variable ensteht, wird für diese Speicher reserviert und als benutzt markiert

Speichermodell und Variablen (2)

```
(define a)

(define b 10)

(define c "Hello")

(define (hello)
   (display c)
    (newline))
```



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)))

Hallo
==> 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"))

==> "Zehn"

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

Sonstwas
==> 15
```

Special Form: `and`

- Syntax: `(and` $expr_1$ `...)`
  - 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` $expr_1$ `...)`
  - 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
```

Noch einmal Rekursion...

- Erinnerung: `is-element?`, komplexer Fall
  - Erfolg: Das gesuche Element ist das erste der Liste
  - Erfolg: Das gesuchte Element is im Rest der Liste

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

- Mit boolscher Logik:
  - Erfolg: Das gesuche Element ist das erste der Liste oder es ist im Rest der Liste

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

Definierte Funktion: `not`

- `not` ist in der initialen Umgebung als eine normale Funktion definiert
- Synax: (`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
```

Ende Vorlesung 7

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 einzelne 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 Initalisierer 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

> (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))

==> 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 $n > 1$
- Finden Sie für möglichst viele der Mengenfunktionen elegantere oder effizientere Implementierung.
  - Bearbeiten Sie dabei mindestens die Potenzmengenbildung.

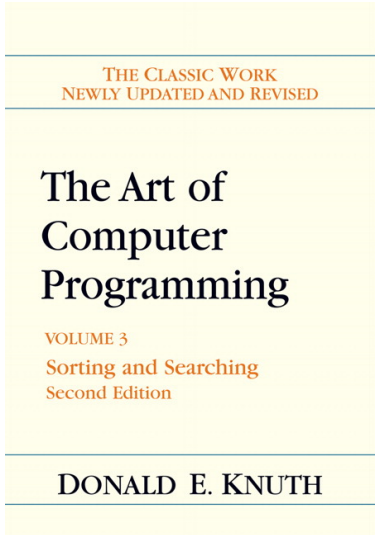
Sortieren durch Einfügen

- 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

Abzuarbeiten	**(Zwischen-)ergebnis**
`'(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!

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 ensteht, 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

Sortieren allgemeiner

- Problem: Unser `isort` sortiert
  1. Nur Zahlen (genauer: Objekte, die mit < verglichen werden können)
  2. Nur aufsteigend
- Lösung?
  - Scheme ist funktional
  - Wir können die „Kleiner-Funktion“ als Parameter übergeben!

```
> (isort '(2 5 3 1))
==> (1 2 3 5)
> (isort2 '(2 5 3 1) <)
==> (1 2 3 5)
> (isort2 '(2 5 3 1) >)
==> (5 3 2 1)
> (isort2 '("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 $1 < 11 < 111 < 1111 < \ldots < 2 < 22 < 222 \ldots$)
  - Die Funktion `number->string` könnte hilfreich sein!

Analyse InsertSort

```
(define (insert k lst)
   (if (null? lst)
       (list k)
       (if (< k (car lst))
           (cons k lst)
           (cons (car lst) (insert k (cdr lst))))))

(define (isort lst)
        (if (null? lst)
            lst
            (insert (car lst) (isort (cdr lst)))))
```

- Wie viele Elemente müssen beim Einfügen in eine Liste der Länge n (im Mittel, ungefähr) verglichen werden?
- Wie viele Elemente müssen eingefügt werden?
- Wie lange ist die Liste, in die eingefügt wird (im Mittel)?

Ende Vorlesung 8

Mergesort

- 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)	()
(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 $\Longrightarrow$ 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
((((2) (5)) (1)) ((3) (7)))	4. Split
(((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 eine sortierten List zusammenfügt.

Lösung

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 und Dateien

- 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 ließt
    - Schließen der Datei: `(close-input-port` *port*`)`
  - Output: `(open-output-file` *name*`)`
    - Schließen der Datei: `(close-output-port` *port*`)`

Beispiel

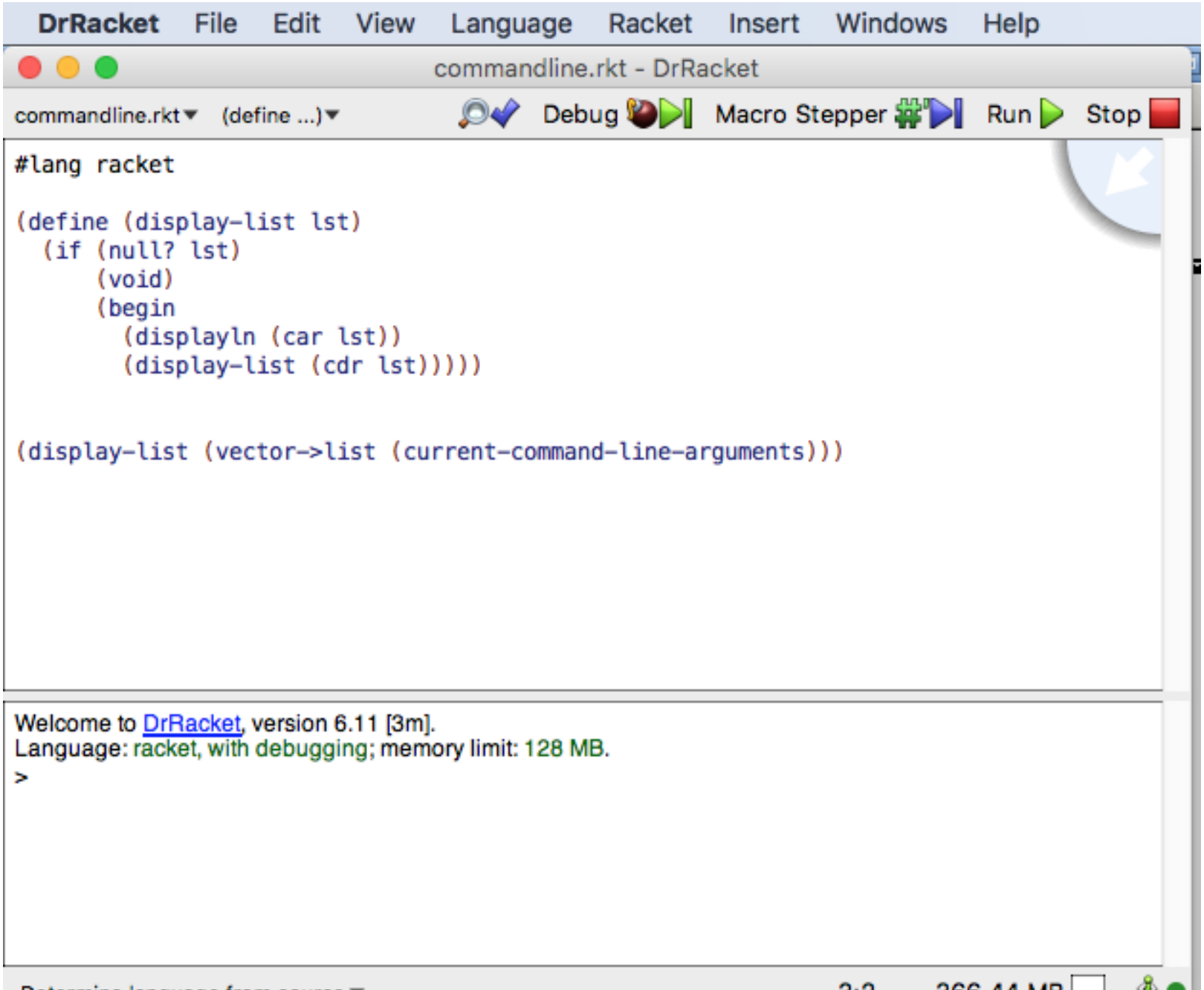
```
> (define (print-file inp-port)
    (if (not (eof-object? (peek-char inp-port)))
        (begin
          (write-char (read-char inp-port))
          (print-file inp-port))
        (close-input-port inp-port)))

> (let ((inprt (open-input-file "test.txt")))
      (print-file inprt))
Testfile
Was passiert?
Ende!
>
```

Erweiterungen: Kommandozeilen und Programmende

Diese Erweiterungen stehen bei Racket zur Verfügung

- Zugiff auf die Kommandozeilenargumente (bei Stand-Alone Programmen): `(current-command-line-arguments)`
  - Ergebnis: Vektor von Strings, die den Kommandozeilenargumenten entspricht
  - Als Liste: `(vector->list (current-command-line-arguments))`
- Beenden des Programs: `(exit` *`obj`*`)`
  - *obj* ist optional
  - Ohne *obj*: Normales Ende (Erfolg)
  - Ansonsten: Implementierungsdefiniert (aber in erster Näherung: Kleine numerische Werte werden als OS-*exit()*-Werte interpretiert)

Beispiel: Kommandozeilenargumente ausgeben

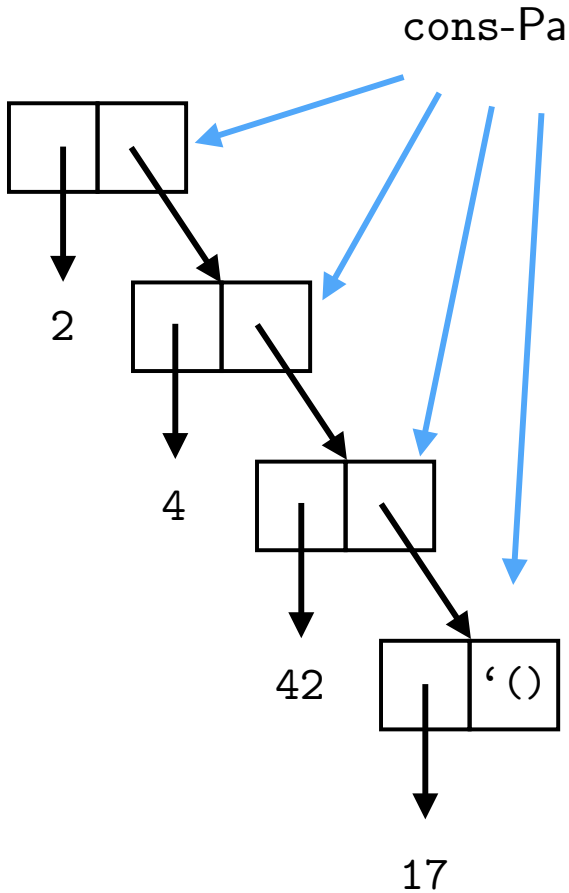


Übung: Komplexität von Sortierverfahren

- Beschaffen Sie sich eine Datei mit 4 Listen von zufällig generierten natürlichen Zahlen mit 1000, 2000, 4000, 8000 und 16000 Elementen (in Scheme-Syntax)
  - Unter `http://wwwlehre.dhbw-stuttgart.de/~sschulz/lgli2025html` können Sie die Datei `sortlists.txt` herunterladen
- Schreiben Sie ein Scheme-Programm, dass 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*)
  - (`cdddr` *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`

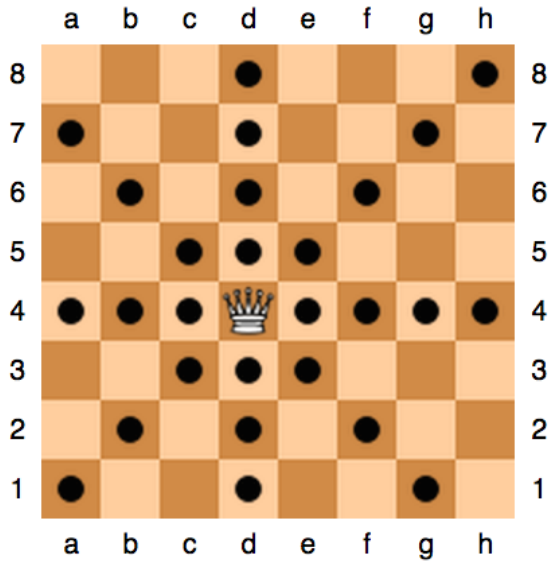
Alists - Schlüssel/Wert Paare

- *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
```

Übung: Höfliche Damen

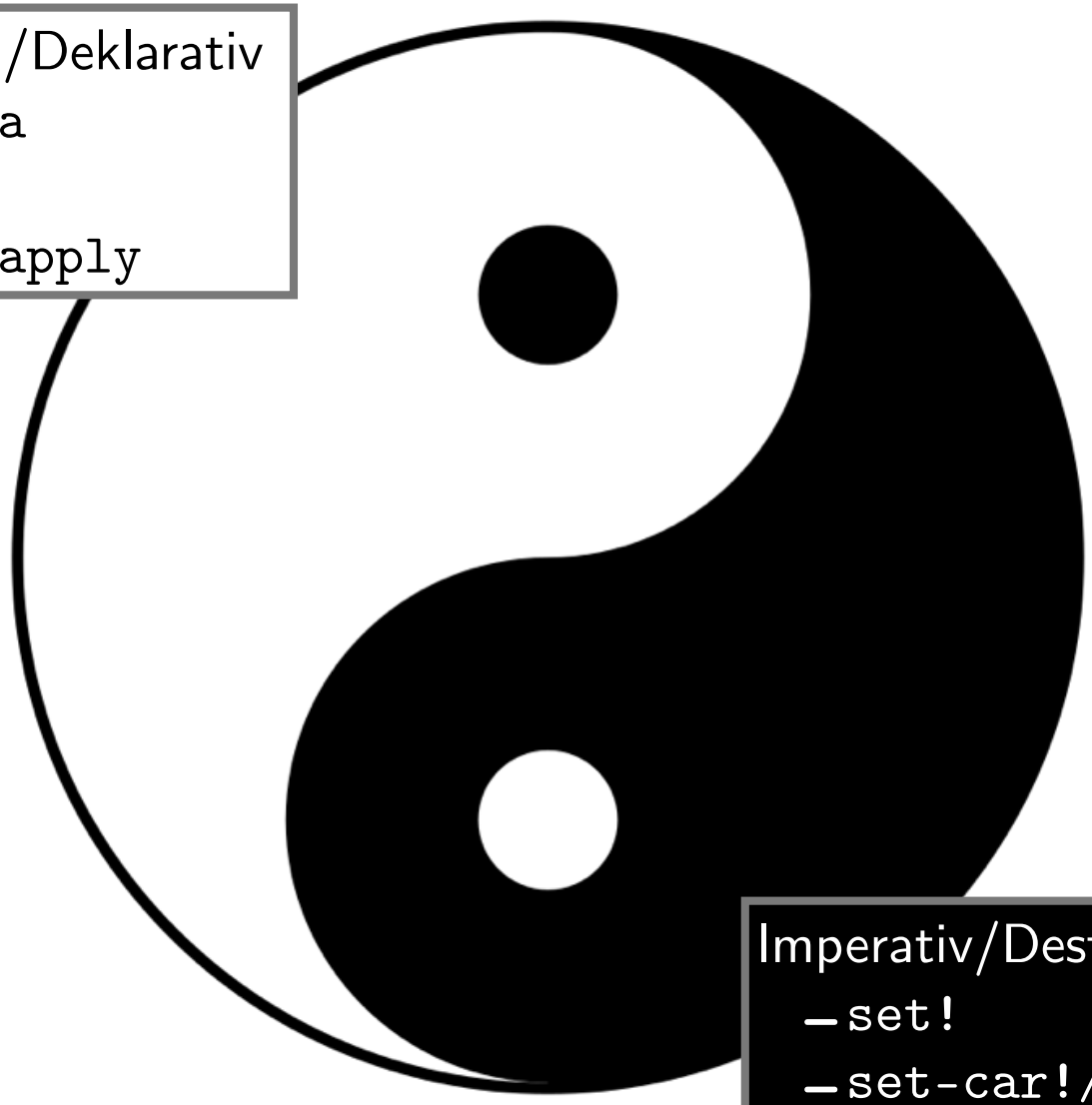
- Eine Dame (im Schach) bedroht alle Felder in ihrer Reihe, in ihrer Spalte, und auf ihren Diagonalen. Das *n-Damen-Problem* besteht darin, n Damen so auf einem $n \times n$-Brett zu platzieren, dass keine Dame eine andere bedroht
  - Aufgabe 1: Suchen Sie nach Lösungen für Bretter mit 1×1, 2×2, 3×3, 4×4 Feldern
  - Aufgabe 2: Erstellen Sie ein Programm, das 8 Damen auf einem Schachbrett so plaziert, 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



Ende Vorlesung 9

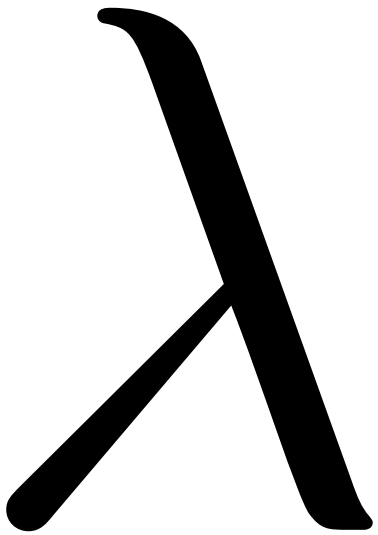
Die Helle und die Dunkle Seite der Macht

Functional/Deklarativ
- `lambda`
- `map`
- `eval/apply`



Imperativ/Destruktiv
- `set!`
- `set-car!/set-cdr!`

Anonyme Funktionen



- Bekannt: Funktionsdefinition mit `define`
  - Z.B. `(define (square x)(* x x))`
- Zwei separate Operationen:
  - Erschaffe 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 angegeben 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`*`))`
- `lambda`s können im lokalen Kontext erzeugt werden
  - Keine Verschmutzung des globalen Namensraums
  - Zugriff auf lokale Variablen (!)
- `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))

==> (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 $(l_1, \ldots, l_n)$
  - Ergebnis: Liste von Ergebnissen
    - Das erste Element des Ergebnises ist $f((\texttt{car}\ l_1), \ldots, (\texttt{car}\ l_n))$
- 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))

==> 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 Befehl: `eval` (Racket)

- `eval` nimmt einen Scheme-Ausdruck und wertet ihn in der aktuellen Umgebung aus
  - Optionales 2. Argument: Environment (R5RS)/Namespace (Racket)
- Beispiele:

```
> (eval '(+ 3 4))
==> 7

> (eval (cons '+ '(3 4 5)))
==> 12
```

Übung: `map` und `apply`

- Schreiben Sie eine Funktion, die das Skalarprodukt von zwei n-elementigen Vektoren (repräsentiert als Listen) berechnet
  - Das Skalarprodukt von $(a_1, a_2, \ldots a_n)$ und $(b_1, b_2, \ldots b_n)$ 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)
==> 13
```

`set-(m)car!/set-(m)cdr!`

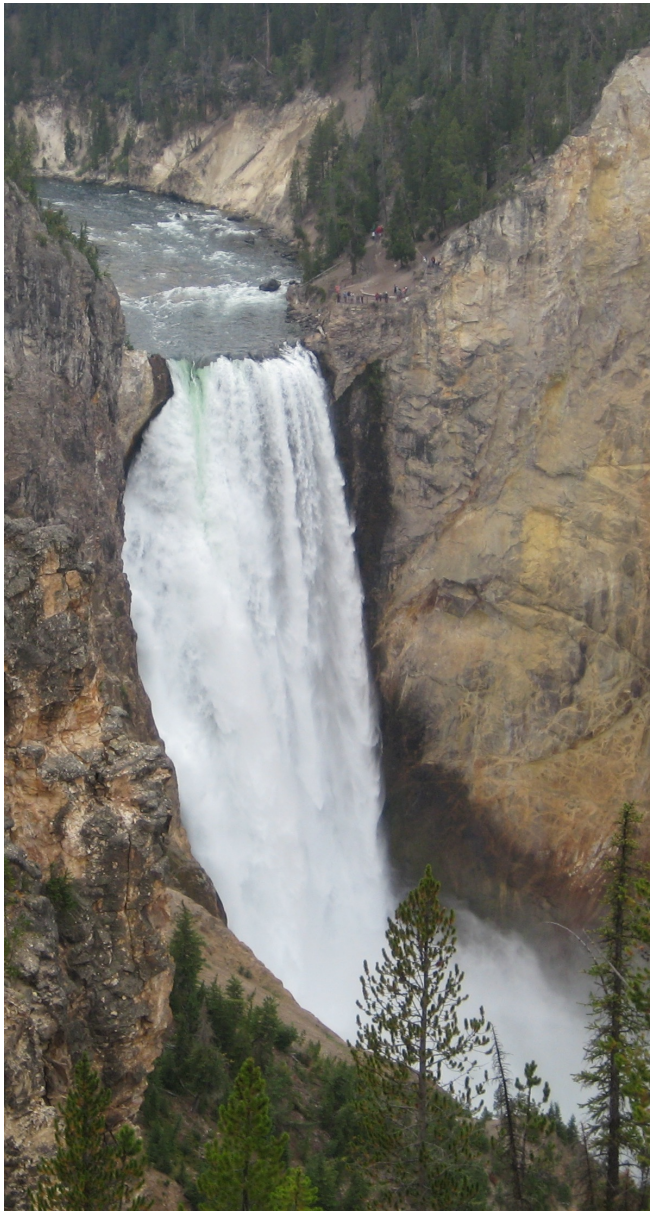
- `set-car!` verändert den ersten Wert eines existierenden `cons`-Paares
- `set-cdr!` verändert den zweiten Wert eines existierenden `cons`-Paares
- Racket verbietet das destruktive Verändern von *normalen* Listen!
- Statt dessen: "Modifizierbare Listen"
  - Gebaut mit `mcons`
  - Verändert mit `set-mcar!`, `set-mcdr!`

"Instead of programming with mutable pairs and mutable lists, data structures such as pairs, lists, and hash tables are practically always better choices." – Racket Manual, 4.10

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: Mastermind (1)

- Klassisches Denkspiel
  - Ziel: Geheimcode ermitteln
  - Hilfe: Hinweise auf Teilerfolge
- Spielbar z.B. auf `http://www.steyrerbrains.at/spiele/Mastermind/index.html`
  - Standard-Version hat Codelänge 4, 6 Farben und erlaubt Wiederholungen!
- Ziel: Programm, dass Mastermind *löst*
  - Also: Initialisierung mit einem Code
  - Programmteil 1 rät Code
  - Programmteil 2 gibt Feedback

Übung: Mastermind (2)

- Konventionen
  - Farben werden als Zahlen 1-6 kodiert
    - Also: `(1 2 2 4)` wäre ein gültiger Code
  - Feedback als 2er-Liste von Zahlen
    - 1. Element: Korrekte Farbe an korrekter Position
    - 2. Element: Korrekte Farbe, aber auf falscher Position
  - Beispiel: Code `(1 2 2 4)`, Tipp `(1 2 4 3)`
    - Bewertung `(2 1)`
- Programmlauf:

```
(master-mind '(4 2 2 1))
Game initialized. Secret code: (4 2 2 1)
Guess: (3 4 5 6) -> (0 1)
Guess: (1 5 2 5) -> (1 1)
Guess: (1 2 4 4) -> (1 2)
Guess: (2 1 2 4) -> (1 3)
Guess: (4 2 2 1) -> (4 0)
'(4 2 2 1)
```

Übung: Mastermind (3)

- Algorithmus:
  1. Erstelle Liste *cand* aller möglichen Codes
  2. Tippe einen beliebigen Wert aus *cand*
     - Bewertung `(4 0)`: Gewonnen
     - Sonst: Entferne alle Werte aus *cand*, die nicht zur aktuellen Bewertung passen
     - Weiter bei Schritt 2

Übung: Mastermind (4)

Partielle Programmstruktur

- `(mm-eval guess soln)` - Bewerte `guess` relativ zu `soln`
  - `(exact-matches l1 l2)` - Zähle gleiche Werte an gleichen Positionen
  - `(count-occ l1 l2)` - Zähle, wie viele Elementen aus `l1` in `l2` vorkommen
    - `(delete-element e lst)` - gib `lst` ohne `e` zurück
- `(make-guesses pins colours)` - generiere alle Codes für `pins` Stifte aus `colours` Farben
  - `(add-to-tuples tuples colours)` - ergänze alle Tuple um alle Farben
    - `(add-to-tuple tuple colours)` - ergänze ein Tupel um alle Farben
- `(define (solve-mm solution candidates))`
  - `(filter-compatible candidates guess ev)`
    - `(is-compatible candidate guess ev)`

Ende Vorlesung 10

Aussagenlogik

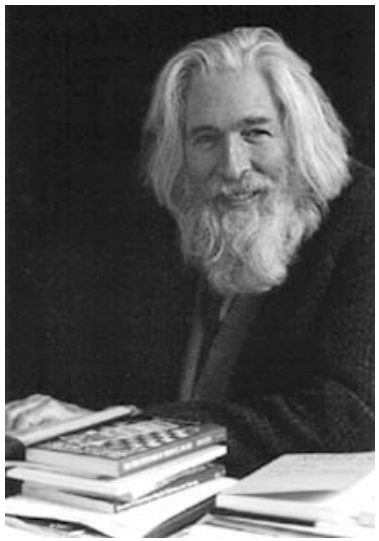
Übung: Verbrechensaufklärung

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



Raymond M. Smullyan (1919-2017)

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.

A	**B**	**C**	**1.**	**2.**	**3.**	**4.**	**1.-4.**

Formale Logik

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

Aussagenlogik - Kernkonzepte

Atomare Aussagen

C ist schuldig

C

Die Straße ist nass

strasseNass

Verknüpft mit logischen Operatoren

und	oder	impliziert	nicht
$\wedge$	$\vee$	$\rightarrow$	$\neg$

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. $A \vee B \vee C$

Syntax der Aussagenlogik: Signatur

Definition (Aussagenlogische Signatur)

Eine aussagenlogische Signatur Σ ist eine (nichtleere) abzählbare Menge von Symbolen, etwa

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

Bezeichnungen für Symbole in Σ

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

Syntax der Aussagenlogik: Logische Zeichen

$\top$ Symbol für den Wahrheitswert „wahr“

$\bot$ 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 Aussagenlogik: Formeln

Definition (Menge $For0_\Sigma$ der Formeln über Σ)

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

- $\top \in For0_\Sigma$
- $\bot \in For0_\Sigma$
- $\Sigma \subseteq For0_\Sigma$ (jedes Atom ist eine Formel)
- Wenn $P, Q \in For0_\Sigma$, dann sind auch
  $(\neg P), \quad (P \wedge Q), \quad (P \vee Q), \quad (P \to Q), \quad (P \leftrightarrow Q)$
  Elemente von $For0_\Sigma$

- Beachte: P und Q 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) $A \to \bot$

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

c) $(A \neg B)$

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

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

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

g) $(A \to A)$

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

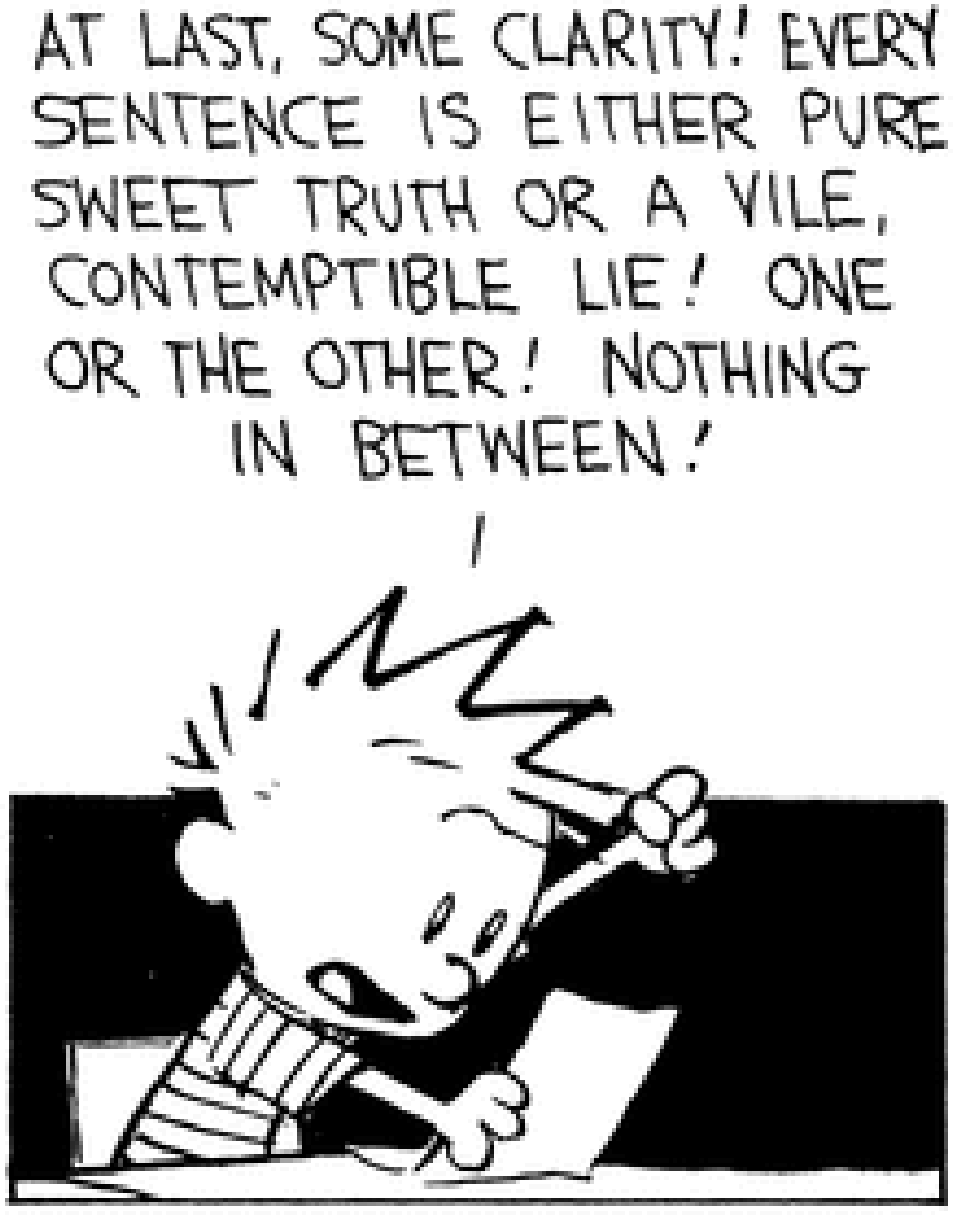
Operator	Präzedenz
$\neg$	1 (stärkste)
$\wedge$	2
$\vee$	3
$\rightarrow$	4
$\leftrightarrow$	5 (schwächste)

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


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

Semantik der Aussagenlogik: Interpretation

Definition (Aussagenlogische Interpretation)

Sei Σ eine aussagenlogische Signatur.

- Eine Interpretation (über Σ) ist eine beliebige Abbildung $I : \Sigma \to \{1, 0\}$.
- Die Menge aller Interpretationen über Σ bezeichen wir als I_Σ.

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.
- Für endliches Σ gilt allgemein $|I_\Sigma| = 2^{|\Sigma|}$.

Semantik der Aussagenlogik (1)

Definition (Auswertung von Formeln unter einer Interpretation (1))

Eine Interpretation I wird fortgesetzt zu einer Auswertungsfunktion

$$\mathsf{val}_I : \mathit{For0}_\Sigma \longrightarrow \{1, 0\}$$

durch:

$$\begin{aligned} \mathsf{val}_I(\top) &= 1 \\ \mathsf{val}_I(\bot) &= 0 \\ \mathsf{val}_I(P) &= I(P) \qquad \text{für } P \in \Sigma \end{aligned}$$

Semantik der Aussagenlogik (2)

Definition (Auswertung von Formeln unter einer Interpretation (2))

und:

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

und:

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

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

Semantik der Aussagenlogik (3)

Definition (Auswertung von Formeln unter einer Interpretation (3))

und:

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

und:

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

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 $\mathrm{val}_I(A) = 1/0$:
  - I macht A wahr/falsch
  - A ist wahr/falsch unter I
  - A ist wahr/falsch in I
- Statt $\mathrm{val}_I(A)$ 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

Materielle Implikation

Ex falso quodlibet

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

Übung: Evaluierung von logischen Formeln

- Betrachten Sie $\Sigma = \{p, q, r, s\}$ und
  - $I_1 = \{p \mapsto 1, q \mapsto 0, r \mapsto 1, s \mapsto 1\}$
  - $I_2 = \{p \mapsto 1, q \mapsto 0, r \mapsto 0, s \mapsto 1\}$
  - $I_3 = \{p \mapsto 0, q \mapsto 0, r \mapsto 1, s \mapsto 1\}$
- Bestimmen Sie den Wert der folgenden Formeln unter den drei Interpretationen
  - $F_1 = (p \wedge q \rightarrow p \vee (q \leftrightarrow \neg r))$
  - $F_2 = (p \wedge q \leftrightarrow p \vee q)$
  - $F_3 = s \wedge (\neg q \vee \neg(p \rightarrow r))$

Ende Vorlesung 11

Modell einer Formel(menge)

Definition (Modell einer Formel)

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

$$\mathrm{val}_I(A) = 1 \qquad \text{(alternative Schreibweise: } I(A) = 1\text{)}$$

Definition (Modell einer Formelmenge)

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

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

- Wir betrachten also eine Formelmenge hier implizit als Konjunktion („ver*und*ung“) 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. $A \vee B \vee C$

A	**B**	**C**	**1.**	**2.**	**3.**	**4.**	**1.-4.**	
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) $A \rightarrow B \leftrightarrow \neg B \rightarrow \neg A$

Können Sie eine Formel angeben, die kein Modell hat?

Modellmengen

Definition (Modellmenge einer Formel)

Mod(F) ist die Menge aller Modelle von F. Formell:

- Sei $A \in For0_\Sigma$ eine Formel. $\text{Mod}(A) = \{I \in I_\Sigma \mid I(A) = 1\}$
- Sei $M \subseteq For0_\Sigma$ eine Formelmenge.
  $\text{Mod}(M) = \{I \in I_\Sigma \mid I \text{ ist Modell von M}\}$

Mengenlehre und logische Verknüpfungen

Wir können die Semantik der Operatoren auch über Mengenoperationen definieren: Seien $p \in \Sigma$, seien $A, B \in For0_\Sigma$. Dann gilt:

$$
\begin{aligned}
\mathrm{Mod}(p) &= \{I \in I_\Sigma \mid I(p) = 1\} \\
\mathrm{Mod}(\top) &= I_\Sigma \\
\mathrm{Mod}(\bot) &= \emptyset \\
\mathrm{Mod}(\neg A) &= \overline{\mathrm{Mod}(A)} \\
\mathrm{Mod}(A \wedge B) &= \mathrm{Mod}(A) \cap \mathrm{Mod}(B) \\
\mathrm{Mod}(A \vee B) &= \mathrm{Mod}(A) \cup \mathrm{Mod}(B) \\
\mathrm{Mod}(A \to B) &= \overline{\mathrm{Mod}(A)} \cup \mathrm{Mod}(B) \\
\mathrm{Mod}(A \leftrightarrow B) &= (\overline{\mathrm{Mod}(A)} \cup \mathrm{Mod}(B)) \cap (\mathrm{Mod}(A) \cup \overline{\mathrm{Mod}(B)})
\end{aligned}
$$

Übung: Modellmengen

- Betrachten Sie $\Sigma = \{p, q, r\}$ und
  - $F = (p \vee q)$
  - $G = (p \rightarrow r)$
- Bestimmen Sie:
  - $\mathrm{Mod}(F)$
  - $\mathrm{Mod}(G)$
  - $\mathrm{Mod}(\neg F)$
  - $\mathrm{Mod}(\neg G)$
  - $\mathrm{Mod}(F \wedge G)$
  - $\mathrm{Mod}(F \rightarrow G)$

Tautologien

Definition (Tautologie)

Eine Formel $F \in For0_\Sigma$ heißt Tautologie oder allgemeingültig, falls $\mathrm{val}_I(F) = 1$ für jede Interpretation I.

Schreibweise: $\models F$

- Sprachregelung: Eine Tautologie ist tautologisch.
- Eine Tautologie F ist unter allen Umständen wahr, unabhängig von der Belegung der Variablen. Es gilt also: $\mathrm{Mod}(F) = I_\Sigma$
- Beispiele:
  - $\top$
  - $A \vee \neg A$
  - $A \to A$
  - $A \to (B \to A)$

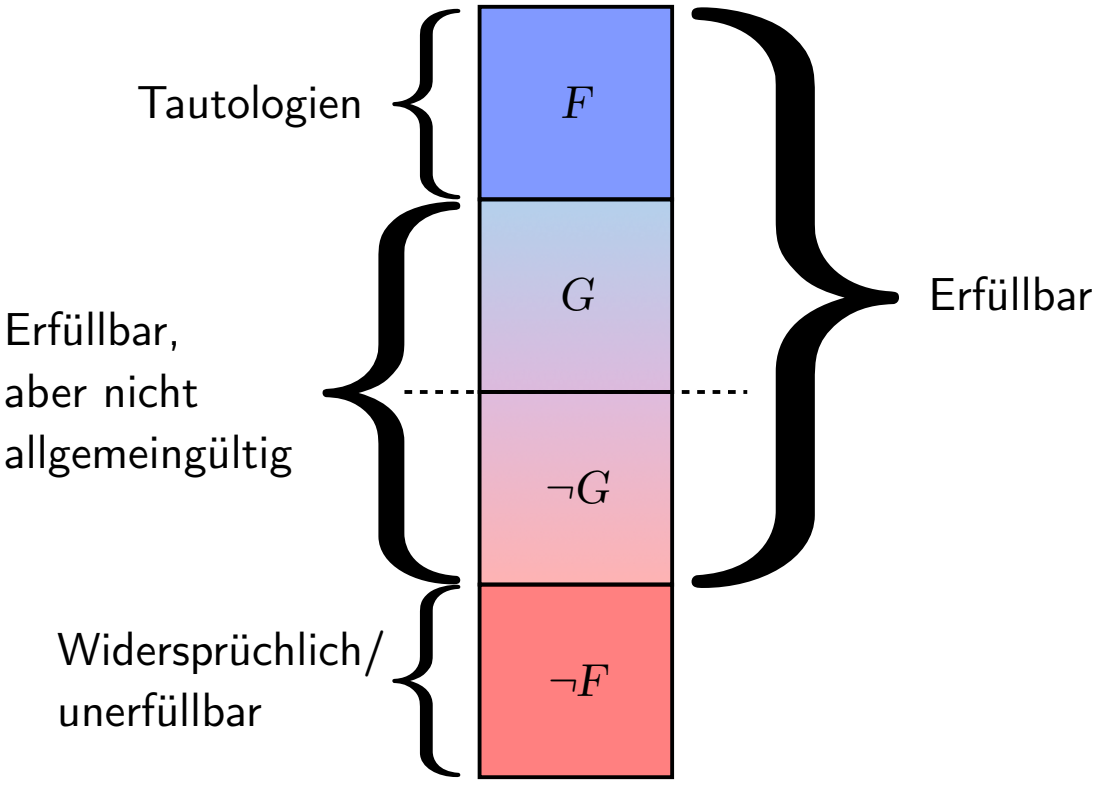
Unerfüllbarkeit

Definition (Unerfüllbarkeit)

- Eine Formel $F \in For0_\Sigma$ heißt unerfüllbar, falls sie kein Modell hat, d.h falls $\mathrm{val}_I(F) = 0$ für jede Interpretation I.
- Eine Formelmenge $A \subseteq For0_\Sigma$ 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?
  - $\bot$
  - $A \wedge \neg A$
  - $(\neg A) \leftrightarrow A$
  - $\{(A \vee B), (A \vee \neg B), (\neg A \vee B), (\neg A \vee \neg B)\}$

(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!
- Speziell: die Negation einer erfüllbaren Formel kann erfüllbar sein!

Unerfüllbarkeit und Allgemeingültigkeit

Satz: (Dualität von Unerfüllbarkeit und Allgemeingültigkeit)

Sei $F \in For0_\Sigma$ eine Formel. Dann gilt: F ist eine Tautologie gdw. $(\neg F)$ unerfüllbar ist.

- Beweis („$\Longrightarrow$“): 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 („$\Longleftarrow$“): 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 $\Longrightarrow$ und $\Longleftarrow$ folgt der Satz. q.e.d.

Logische Folgerung

Definition (Logische Folgerung)

Eine Formel A folgt logisch aus Formelmenge KB
gdw.
alle Modelle von KB sind auch Modell von A
(äquivalent: $\text{Mod}(KB) \subseteq \text{Mod}(A)$)
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 Spezifikation einer Schaltung das gewünschte 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

A	**B**	**C**	**1.**	**2.**	**3.**	**4.**	**KB**	Kommentar	**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) $KB \models (A \vee B)$

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!

Ende Vorlesung 12

Deduktionstheorem

Satz: (Deduktionstheorem)

$F_1, \ldots, F_n \models G$ gdw. $\models (F_1 \wedge \ldots \wedge F_n) \rightarrow G$

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

Wir können das Problem der logischen Folgerung reduzieren auf einen Test der Allgemeingültigkeit einer Formel!

- ... und also genau dann, wenn $\neg((F_1 \wedge \ldots \wedge F_n) \rightarrow G)$ unerfüllbar ist (nach Satz: Unerfüllbarkeit und Allgemeingültigkeit).

Unerfüllbarkeit

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

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

Sind wir mit der Aussagenlogik fertig?

Anwendungen von Aussagenlogik

- Anwendungsbereiche
  - Hardware-Verifikation
  - Software-Verifikation
  - Kryptographie
  - Bio-Informatik
  - Diagnostik
  - Konfigurationsmanagement
- Größe echter Probleme:
  - Quelle: SAT-RACE 2010,
    `https://baldur.iti.kit.edu/sat-race-2010/`
  - Kleinstes CNF-Problem: 1694 aussagenlogische Variable
  - Größtes Problem: 10 950 109 aussagenlogische Variable
  - Beweiser im Wettbewerb haben ≈80% Erfolgsquote!

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

Wider die Wahrheitstafel. . .

Die Wahrheitstafelmethode muss alle Interpretationen betrachten!

- $2^{1694} =$
  880668889606027853447740861991726967406733818923567420943787865915770151391693052437052806123402268334422032472871055154010100030466445931952653881792190414089415517697017224920420724582220579305677036044240064760584902924659218190306462313414149311550791275258264005150281274697251368824854797968791349433872037125384069717927974613689982161836405664786537607103689106339431738947020490620263676821205374146735973617690088882917066588031987458198543014856545747116756031135831699254784181546802859819699011584 $\approx 9 \cdot 10^{511}$
- Zum Vergleich: Das Universum hat ca. 10^{80} Atome

Wir brauchen bessere Kalküle!

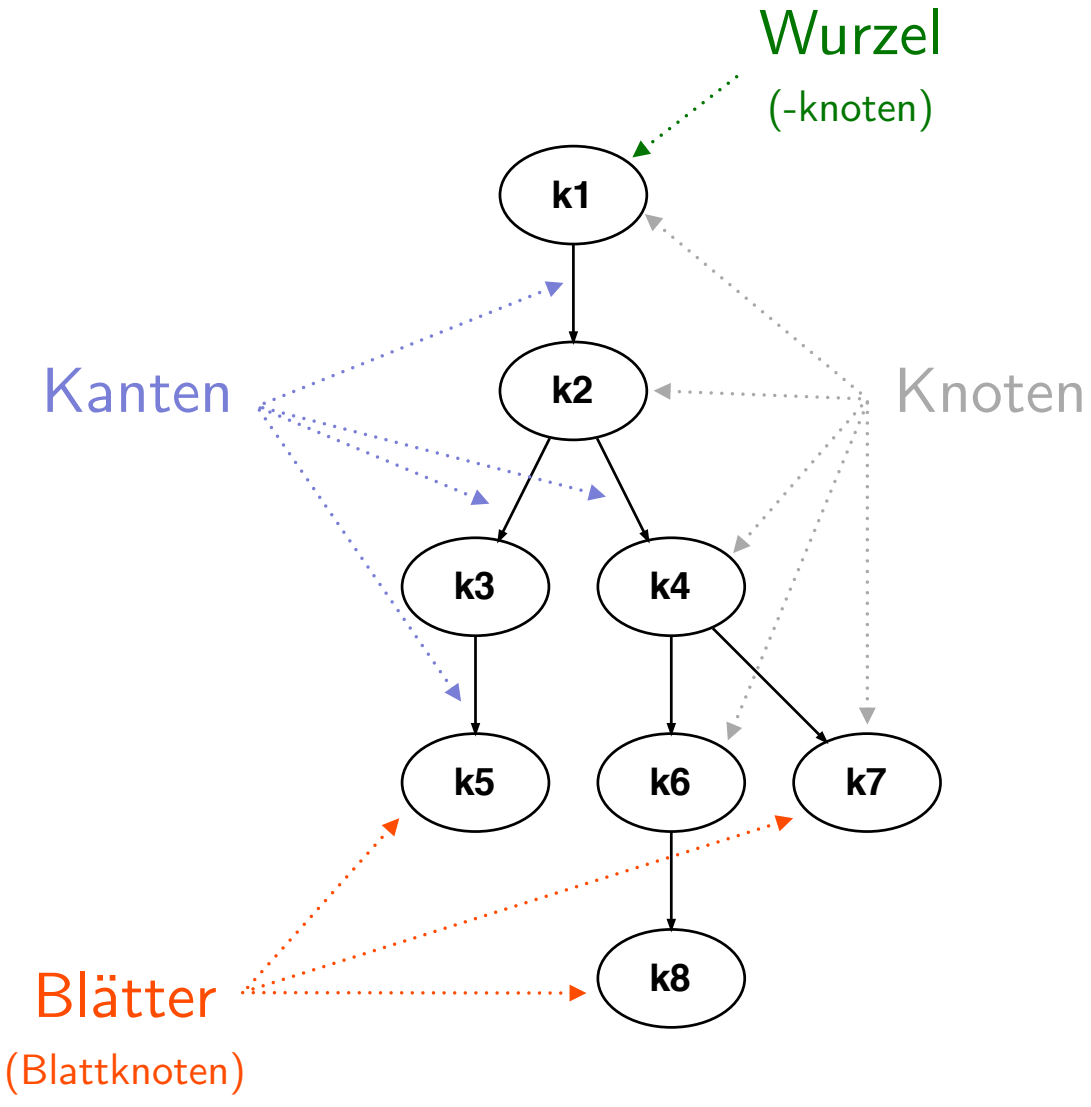
Widerspruchskalküle/Widerlegungskalküle

- Viele praktisch effiziente Kalküle zeigen die Unerfüllbarkeit einer Formel(-Menge):
  - Zu zeigen: Eine Hypothese H folgt aus einer Menge von Formeln $KB = \{F_1, F_2, \ldots, F_n\}$:
    $$KB \models H$$
  - Deduktionstheorem: Das gilt genau dann, wenn die entsprechende Implikation allgemeingültig ist, also:
    $$\models (F_1 \wedge F_2 \wedge \ldots \wedge F_n) \rightarrow H$$
  - Dualitätsprinzip: Das gilt genau dann, wenn die Negation dieser Formel unerfüllbar ist:
    $$\neg((F_1 \wedge F_2 \wedge \ldots \wedge F_n) \rightarrow H) \text{ unerfüllbar}$$
  - Fakt: Äquivalent können wir zeigen:
    $$F_1 \wedge F_2 \wedge \ldots \wedge F_n \wedge \neg H \text{ unerfüllbar}$$
    $$\{F_1, F_2, \ldots, F_n, \neg H\} \text{ unerfüllbar}$$

Übung: Deduktionstheorem/Widerspruchsbeweise

- Betrachten Sie das bekannte Beispiel von Jane:
  1. $KB = \{\neg K \wedge E \rightarrow M, B \rightarrow E, \neg B \rightarrow F, \neg M, \neg K\}$
  2. Vermutung: F
  3. Zu zeigen: $KB \models F$
- Generieren Sie aus KB und F eine Formel, die tautologisch ist, wenn F aus KB folgt.
- Generieren Sie aus KB und F eine Formelmenge, die unerfüllbar ist, wenn F aus KB folgt.

Exkurs: Bäume



Formal:

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

Bäume (1)

Definition (Baum)

- Sei V eine beliebige Menge (die Knoten, Vertices) und $E \subseteq V \times V$ (die Kanten, Edges) eine zweistellige Relation über V.
  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 $(x, r) \in E$ exisitiert.
    „Die Wurzel hat keinen Vorgänger. “
  - Für alle $k \in V \backslash \{r\}$ gilt: Es gibt genau ein $x \in V$ mit $(x, k) \in E$.
    „Jeder Knoten außer der Wurzel hat einen eindeutigen Vorgänger.“

Bäume (2)

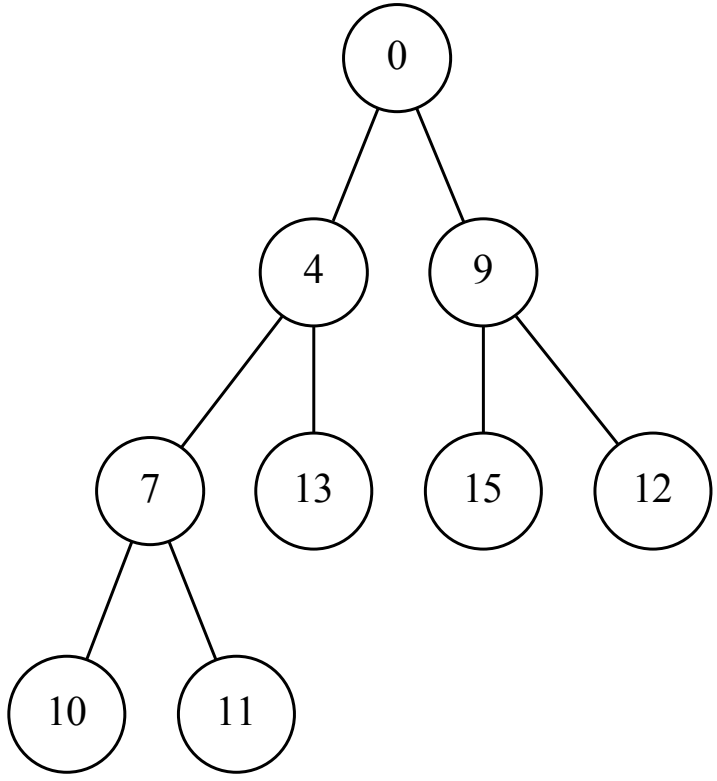
Definition (Pfad, Ast, Binärbaum...)

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 $p = \langle k_0, k_1, \ldots, k_n \rangle$ $(m \in \mathbb{N})$ 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.

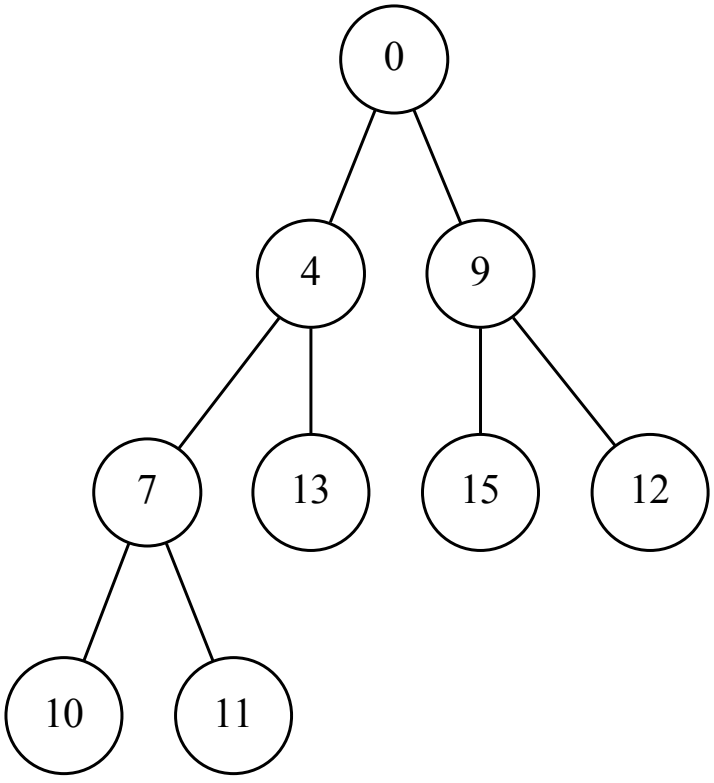
Bäume (3)

- 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.



Min-Heap zu $\{0, 4, 7, 12, 13, 9, 15, 10, 11\}$ in Baumdarstellung

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?

Analytische Tableaux für die Aussagenlogik

Wesentliche Eigenschaften

- Widerlegungskalkül: Versucht die Unerfüllbarkeit einer Formel (oder Formelmenge) zu zeigen
- Beweis durch vollständige Fallunterscheidung
- Sukzessive Zerlegung der Formel in ihre Bestandteile („Analyse“)
  - . . . basierend auf *syntaktischen* Regeln!
- Organisation der zerlegten Formeln als *Baum* (Tableau)

Beobachtung

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

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 erfüllenden 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

E.W. Beth

Jaakko Hintikka

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

Idee: Fallunterscheidung nach Formeltyp

Uniforme Notation: Jede komplexe Formel ist α oder β

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

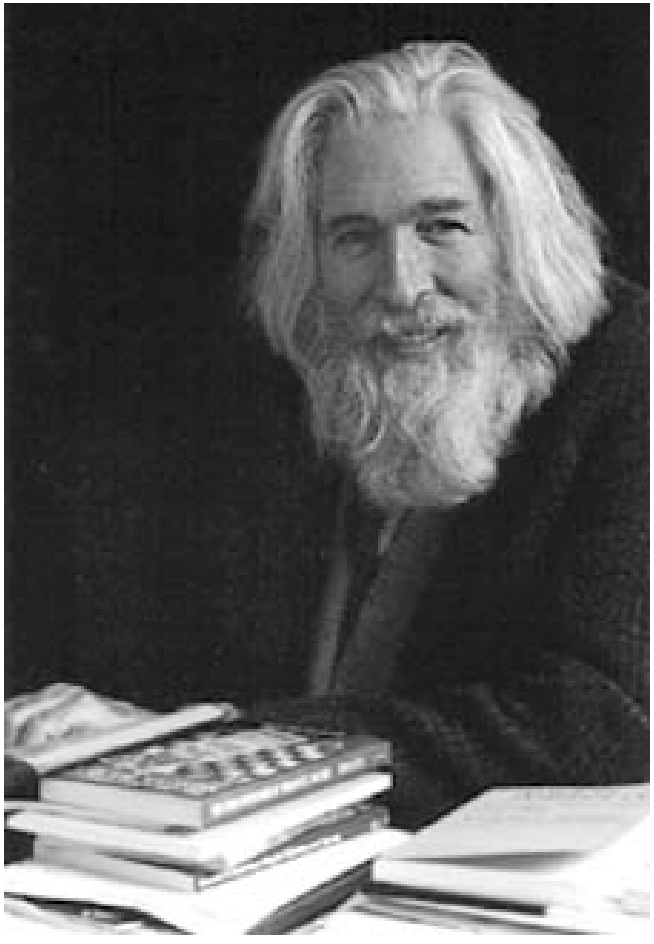
Uniforme Notation: α und β

Konjunktive Formeln: Typ α

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

Disjunktive Formeln: Typ β

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



Uniforme Notation: Zerlegung

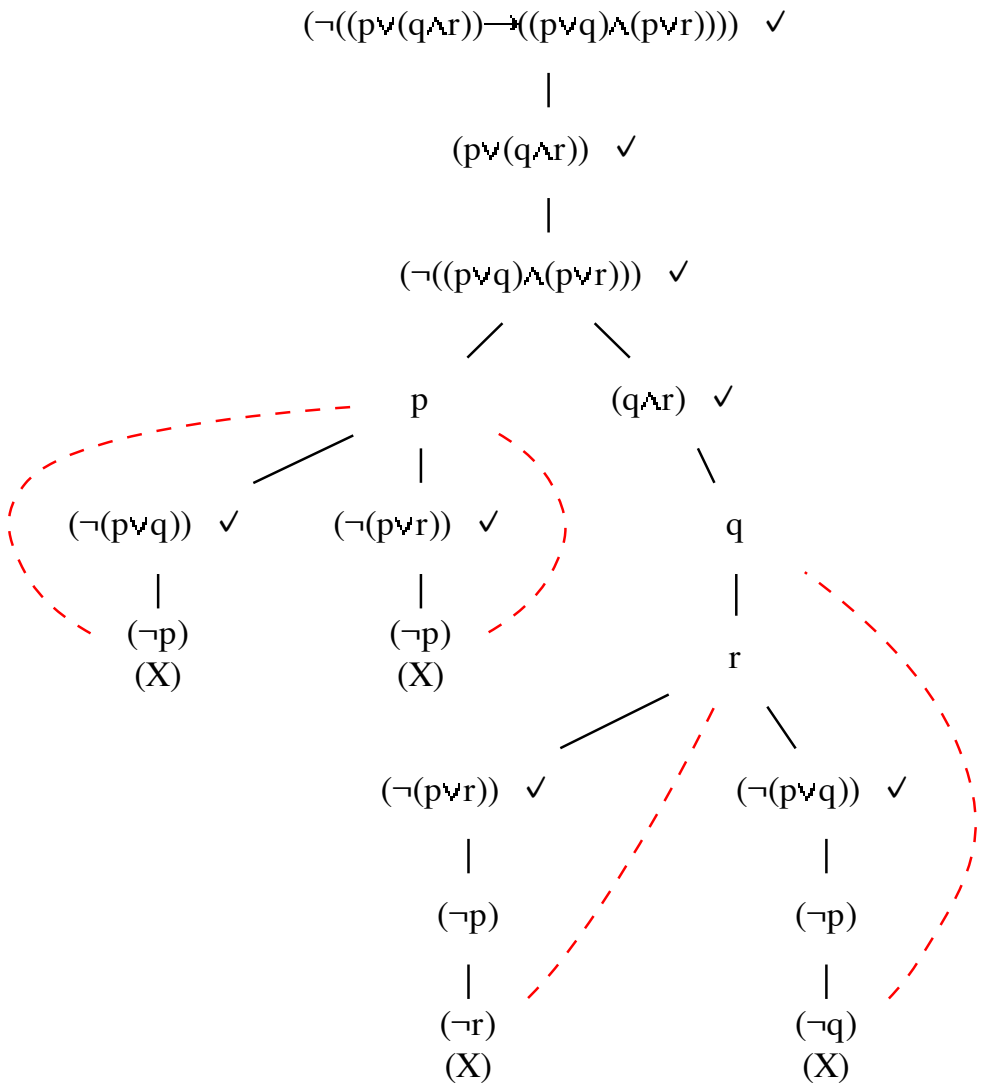
Zuordnungsregeln Formeln / Unterformeln

α	α_1	α_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	

β	β_1	β_2
$\neg(A \wedge B)$	$\neg A$	$\neg B$
$A \vee B$	A	B
$A \rightarrow B$	$\neg A$	B
$\neg(A \leftrightarrow B)$	$A \wedge \neg B$	$\neg A \wedge B$

- α ist wahr, wenn α_1 und α_2 wahr sind
- β ist wahr, wenn β_1 oder β_2 wahr ist

Tableaux-Beispiel



α	α_1	α_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

β	β_1	β_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-Konstruktion

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
  - α-Regel: $\dfrac{\alpha}{\begin{array}{c}\alpha_1\\ \alpha_2\end{array}}$
  - β-Regel: $\dfrac{\beta}{\beta_1 \mid \beta_2}$
  - Abschluss-Regeln: $\dfrac{\begin{array}{c}A\\ \neg A\end{array}}{\times}$ $\qquad$ $\dfrac{\begin{array}{c}\neg A\\ A\end{array}}{\times}$

Erläuterungen zu den Regeln

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

    α_1

    |

    α_2
- β-Regel:
  - Wähle einen beliebigen Knoten mit einer β-Formel A, auf die noch keine Regel angewendet wurde
  - Erweitere jeden offenen Ast, der durch den Knoten A geht, durch Anhängen von

    β_1 $\quad$ β_2
- Abschluss-Regel: Ein Ast, auf dem eine Formel und ihre Negation vorkommt, kann als geschlossen markiert werden.

Zerlegungen sind eindeutig!

- Jede nicht-primitive Formel hat eine eindeutige α- oder β-Zerlegung
  - Wir sind frei in der Wahl des Knotens, denn wir expandieren wollen
  - Wir haben aber keine Wahl dabei, wie wir die entsprechende Formel zerlegen
  - Die Zerlegung folgt zwingend aus dem obersten oder (bei $\neg$-Formeln) den obersten beiden Operatoren der Formel
- Beispiele:
  - $a \wedge b \rightarrow c$: Der oberste Operator ist $\rightarrow$ (wie man sieht, wenn man die Formel vollständig klammert), damit ist das eine β-Formel, $\beta_1 = \neg(a \wedge b)$, $\beta_2 = c$
  - $a \vee b \vee c \wedge a$: Der oberste Operator ist das zweite $\vee$, die Formel ist β.

Falsche (oft "bequeme") Zerlegung von Formeln ist häufige Fehlerursache!

Tableaux und Formeln

Definition (Tableaux für eine Formel)

Sei $F \in For0_\Sigma$ 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 ensteht, dann ist T' ein Tableau für F
- Nichts anderers ist ein Tableau für F

- Eine Ableitung im Tableaux-Kalküle erzeugt also eine Folge von Tableaux
- Ein Nachfolgetableau ist entweder eine Erweiterung des Vorgängers, oder ensteht durch Markierung eines Astes als geschlossen

Tableaux-Eigenschaften

Definition (Offene und geschlossene Tableaux)

Sei T ein Tableaux 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 α- und β-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 Streikbegin 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?

Ende Vorlesung 13

Umsetzung des Tableaux-Kalküls

- Markiere expandierte Knoten
- Heuristik: Bevorzuge α-Regeln
  - Ausnahme: $a \wedge b \wedge c \ldots$ kann (von Hand abweichend vom exakten Kalkül) in einem Schritt expandiert werden
- 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: Jane als Tableau

- Informelles Problem
  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.
- Formell: $\neg K \wedge E \rightarrow M, B \rightarrow E, \neg B \rightarrow F, \neg M, \neg K \models F$
- Deduktionstheorem: Das gilt genau dann, wenn:
  $\models ((\neg K \wedge E \rightarrow M) \wedge (B \rightarrow E) \wedge (\neg B \rightarrow F) \wedge \neg M \wedge \neg K) \rightarrow F$
- . . . oder falls folgende Formel unerfüllbar ist:
  $G = \neg(((\neg K \wedge E \rightarrow M) \wedge (B \rightarrow E) \wedge (\neg B \rightarrow F) \wedge \neg M \wedge \neg K) \rightarrow F)$

Konstruieren Sie ein vollständiges Tableau zu G

Tableaux-Beweise

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?

Interpretationen von Ästen

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 $\langle F_1, F_2, \ldots, F_n \rangle$, wobei die F_i die Formeln an den Knoten des Astes sind.

Lemma: Erfüllbare Tableaux

Lemma: (Existenz erfüllbarer Äste)

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

- Beweisidee: Induktion nach der Länge der Ableitung
  - Sei T ein Tableau für F.
  - Dann existiert nach der Definition "Tableaux für eine Formel" eine Sequenz $T_0, T_1, \ldots, T_n$, so dass gilt:
    - T_0 ist das initiale Tableau für F (d.h. T_0 besteht aus nur einem Knoten, der mit F beschriftet ist)
    - T_{i+1} entsteht aus T_i durch Anwendung einer Tableaux-Regel
    - $T_n = T$ (T steht am Ende der Ableitung)

Lemma (Induktionsanfang)

Also: Wir zeigen per Induktion: Wenn $I(F) = 1$ und $T_0, T_1, \ldots, T_n$ eine Sequenz von Tableaux für F ist, dann gibt es in jedem T_i einen Ast M mit $I(M) = 1$

- 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 $I(\langle F \rangle) = 1$.
  - Also ist der Induktionsanfang gesichert.
- Induktionsvoraussetzung: T_i hat Ast M mit $I(M) = 1$

Lemma (Induktionsschritt)

- Induktionsschritt: T_{i+1} entsteht aus T_i durch Anwendung einer Tableaux-Regel.
- 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) α-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 exisitiert ein Pfad in T_{i+1} der unter I wahr ist.
     b) β-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 (a+b) existiert der gesuchte Ast.

  In beiden Fällen (1+2) 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. q.e.d.

Korrektheit des Tableaux-Kalküls

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: 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

Tableaux und Modelle

Satz: (Tableaux-Modelle)

Sei F eine aussagenlogische Formel, T ein vollständiges Tableau von F, 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 M 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.

Vollständigkeit und Korrektheit

Satz: (Vollständigkeit und Korrektheit des Tableaux-Kalküls)

Sei $F \in For0_\Sigma$ eine aussagenlogische Formel. Dann gilt:

- Der Tableaux-Kalkül generiert zu jeder aussagenlogischen Formel nach endlich vielen Schritten ein vollständiges Tableaux T für F
- Wenn T offen ist, so ist F erfüllbar
- Wenn T geschlossen ist, so ist F unerfüllbar

- Da die Teilformeln bei jeder Zerlegung kleiner werden, wird jede Ableitung nach endliche vielen Schritten ein vollständiges Tableau erzeugen!
- Zusammenfassung der letzten beiden Sätze

Der Tableaux-Kalkül ist ein Entscheidungsverfahren für die Erfüllbarkeit in der Aussagenlogik!

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:

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

- 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.

Übung: Mehr Tableaux

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

Ende Vorlesung 14

Logische Äquivalenz

Definition (Logische Äquivalenz)

Zwei Formeln $F, G \in For0_\Sigma$ 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
- Also $\mathrm{Mod}(F) = \mathrm{Mod}(G)$
- 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!

Übung: Äquivalenz

- Finden Sie äquivalente Paare von verschiedenen Formeln...
  - ...mit 3 gemeinsamen Aussagenvariablen/Atomen
  - ...mit 2 gemeinsamen Aussagenvariablen, wobei beide Formeln keinen Operator gemeinsam haben
  - ...mit Aussagenvariablen, aber ohne gemeinsame Aussagenvariablen
- Wie immer gibt es faule und interessante Lösungen!

Basis der Aussagenlogik

Definition (Basis der Aussagenlogik)

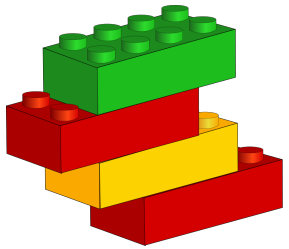
Eine Menge von Operatoren O heißt eine Basis der Aussagenlogik, falls gilt: Zu jeder Formel F gibt es eine Formel G mit $F \equiv G$, und G verwendet nur Operatoren aus O.

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

Satz: (Basen der Aussagenlogik)

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

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 einzelnen Teile bereits die Eigenschaft haben.

      Achtung: Das ist wieder eine bedingte Aussage - erst mit dem Induktionsanfang wird daraus ein Beweis!
  - Konkret für aussagenlogische Formeln:
    - Basisfälle sind die aussagenlogischen Variablen oder $\top$, $\bot$.
    - Zusammengesetzte Formeln haben die Form $(\neg A)$, $(A \vee B)$, $(A \wedge B)\ldots$, 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 eine der folgenden Aussagen:
  1. $\{\wedge, \vee, \neg\}$ ist eine Basis
  2. $\{\rightarrow, \neg\}$ ist eine Basis
  3. $\{\wedge, \neg\}$ ist eine Basis

Beispiellösung Teil 2

Unterformel/Teilformel

Definition (Teilformel)

Seien $A, B \in For0_\Sigma$. 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: $TF(A)$ ist die Menge der Teilformeln von A.

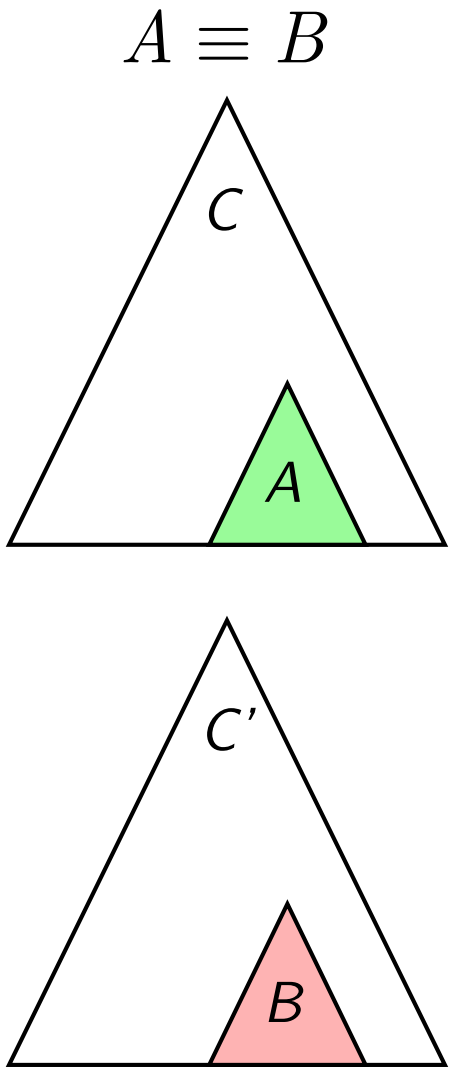
- Beispiel:
  - $TF((a \vee (\neg b \leftrightarrow c))) = \{(a \vee (\neg b \leftrightarrow c)), a, (\neg b \leftrightarrow c), \neg b, b, c\}$
  - $TF(\neg\neg\neg a) = \{\neg\neg\neg a, \neg\neg a, \neg a, a\}$

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: $p \vee q \quad \equiv \quad q \vee p$
  impliziert
  $(r \wedge (p \vee q)) \rightarrow s \quad \equiv \quad (r \wedge (q \vee p)) \rightarrow s$

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 $C \equiv \bot$ 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 und sei $C \neq A$ (sonst wie IA). 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.

- 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 $\text{val}_I(C) = \text{val}_I(C')$. Sei also I eine beliebige Interpretation.
  - Fall 1a): $\text{val}_I(C) = 0 \Longrightarrow \text{val}_I(D) = 1 = \text{val}_I(D') \Longrightarrow \text{val}_I(C') = 0$ per Definition val_I
  - Fall 1b): $\text{val}_I(C) = 1 \Longrightarrow \text{val}_I(D) = 0 = \text{val}_I(D') \Longrightarrow \text{val}_I(C') = 1$ per Definition val_I

Substitutionstheorem (Beweis) - 3

IS: (Fortgesetzt)

- Fall 2: $C = (D \vee E)$. Sei o.B.d.A. A Unterformel von D
  - Fall 2a(i): $\text{val}_I(D) = 1 \Longrightarrow \text{val}_I(D') = 1$ per IV
    $\Longrightarrow \text{val}_I(D' \vee E) = 1 \Longrightarrow \text{val}_I(C') = 1$
  - Fall 2a(ii): $\text{val}_I(E) = 1 \Longrightarrow \text{val}_I(D' \vee E) = 1 \Longrightarrow \text{val}_I(C') = 1$
  - Fall 2b): $\text{val}_I(C) = 0 \Longrightarrow \text{val}_I(D) = 0$ und
    $\text{val}_I(E) = 0 \Longrightarrow \text{val}_I(D') = 0$ (per IV)
    $\Longrightarrow \text{val}_I(D' \vee E) = 0 \Longrightarrow \text{val}_I(C') = 0$
- 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 $\bot$ 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 $\wedge$
- $(A \vee B) \equiv (B \vee A)$ — Kommutativität von $\vee$
- $((A \wedge B) \wedge C) \equiv (A \wedge (B \wedge C))$ — Assoziativität von $\wedge$
- $((A \vee B) \vee C) \equiv (A \vee (B \vee C))$ — Assoziativität von $\vee$
- $(A \wedge A) \equiv A$ — Idempotenz für $\wedge$
- $(A \vee A) \equiv A$ — Idempotenz für $\vee$
- $\neg\neg A \equiv A$ — Doppelnegation
- $(A \to B) \equiv (\neg B \to \neg A)$ — Kontraposition

Wichtige Äquivalenzen (2)

- $(A \to B) \equiv (\neg A \vee B)$ Elimination Implikation
- $(A \leftrightarrow B) \equiv ((A \to B) \wedge (B \to A))$ 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)

1.	$(A \wedge B)$	$\equiv$	$(B \wedge A)$	Kommutativität von $\wedge$
2.	$(A \vee B)$	$\equiv$	$(B \vee A)$	Kommutativität von $\vee$
3.	$((A \wedge B) \wedge C)$	$\equiv$	$(A \wedge (B \wedge C))$	Assoziativität von $\wedge$
4.	$((A \vee B) \vee C)$	$\equiv$	$(A \vee (B \vee C))$	Assoziativität von $\vee$
5.	$(A \wedge A)$	$\equiv$	A	Idempotenz für $\wedge$
6.	$(A \vee A)$	$\equiv$	A	Idempotenz für $\vee$
7.	$\neg\neg A$	$\equiv$	A	Doppelnegation
8.	$(A \rightarrow B)$	$\equiv$	$(\neg B \rightarrow \neg A)$	Kontraposition
9.	$(A \rightarrow B)$	$\equiv$	$(\neg A \vee B)$	Elimination Implikation
10.	$(A \leftrightarrow B)$	$\equiv$	$((A \rightarrow B)) \wedge (B \rightarrow A))$	Elimination Äquivalenz
11.	$\neg(A \wedge B)$	$\equiv$	$(\neg A \vee \neg B)$	De-Morgans Regeln
12.	$\neg(A \vee B)$	$\equiv$	$(\neg A \wedge \neg B)$	De-Morgans Regeln
13.	$(A \wedge (B \vee C))$	$\equiv$	$((A \wedge B) \vee (A \wedge C))$	Distributivität von $\wedge$ über $\vee$
14.	$(A \vee (B \wedge C))$	$\equiv$	$((A \vee B) \wedge (A \vee C))$	Distributivität von $\vee$ über $\wedge$

Wichtige Äquivalenzen mit $\top$ und $\bot$

15. $(A \wedge \neg A) \equiv \bot$
16. $(A \vee \neg A) \equiv \top$ Tertium non datur
17. $(A \wedge \top) \equiv A$
18. $(A \wedge \bot) \equiv \bot$
19. $(A \vee \top) \equiv \top$
20. $(A \vee \bot) \equiv A$
21. $(\neg\top) \equiv \bot$
22. $(\neg\bot) \equiv \top$

Kalkül: Logische Umformung

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: $\vdash_{LU} (p \to (p \vee q))$
- Herleitung:

$$
\begin{array}{lll}
\top & \vdash_{LU} & q \vee \top \\
 & \vdash_{LU} & \top \vee q \qquad\qquad \text{Kommutativität} \\
 & \vdash_{LU} & (p \vee \neg p) \vee q \\
 & \vdash_{LU} & (\neg p \vee p) \vee q \qquad \text{Kommutativität} \\
 & \vdash_{LU} & \neg p \vee (p \vee q) \qquad \text{Assoziativität} \\
 & \vdash_{LU} & p \to (p \vee q) \qquad \text{Elimination Implikation}
\end{array}
$$

Übung: Logische Umformungen

Zeigen Sie:

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

Lösung zu Teil 1

Kalkül: Logische Umformung

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 $A \vdash_{LU} B$
  - Beispiel: $a \wedge b \models a$, aber $a \wedge b \nvdash_{LU} a$

Ende Vorlesung 15

Normalformen

Normalformen

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

Idee: Konvertierung in einfachere Teilsprache

Negations-Normalform

Definition (Negations-Normalform (NNF))

Eine Formel $F \in For0_\Sigma$ ist in Negations-Normalform, wenn folgende Bedingungen gelten:

- Als Operatoren kommen nur $\wedge, \vee, \neg$ vor.
- $\neg$ kommt nur direkt vor Atomen vor.
- $\top$, $\bot$ sind nur erlaubt, wenn $F = \top$ oder $F = \bot$.

- Beispiele:
  - $F = (A \wedge \neg B) \vee C$ ist in NNF
    F ist eine NNF von $(A \to B) \to C$
  - $(A \vee \neg(B \wedge C))$ ist nicht in NNF
  - $(\neg A \vee \neg\neg B)$ ist nicht in NNF
    ... aber $(\neg A \vee B)$ ist in NNF

Transformation in NNF

Drei Schritte

1. Elimination von $\leftrightarrow$
   Verwende $A \leftrightarrow B \quad \equiv \quad (A \rightarrow B) \wedge (B \rightarrow A)$
2. Elimination von $\rightarrow$
   Verwende $A \rightarrow B \quad \equiv \quad \neg A \vee B$
3. „Nach innen schieben“ von $\neg$, Elimination $\top, \bot$
   Verwende de-Morgans Regeln und $\neg\neg A \quad \equiv \quad A$
   Verwende $\top/\bot$-Regeln

Ergebnis: Formel in NNF oder $\top/\bot$

Übung: NNF

Bestimmen Sie je eine NNF zu den folgenden Formeln:

- $\neg(a \to \top) \vee a$
- $\neg(a \vee b) \leftrightarrow (b \wedge (a \to \neg c))$
- $(a \to b) \to ((b \to c) \to (a \to c))$

Literale und Klauseln

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): $\bot$ oder $\square$
    - Die leere Klausel ist unerfüllbar

Konjunktive Normalform

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)

Transformation in KNF

ZweiVier Schritte

1. Transformation in NNF
   - 1.1 Elimination von $\leftrightarrow$
     Verwende $A \leftrightarrow B \quad \equiv \quad (A \rightarrow B) \wedge (B \rightarrow A)$
   - 1.2 Elimination von $\rightarrow$
     Verwende $A \rightarrow B \quad \equiv \quad \neg A \vee B$
   - 1.3 „Nach innen schieben“ von $\neg$, elimination $\top, \bot$
     Verwende de-Morgans Regeln und $\neg\neg A \quad \equiv \quad A$
     Verwende $\top/\bot$-Regeln

2. „Nach innen schieben“ von $\vee$
   Verwende Distributivität von $\vee$ über $\wedge$ (und Vereinfachungen)

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 $\neg$
  $(\neg p \vee q \vee r) \ \wedge \ ((\neg q \wedge \neg r) \vee p)$ (NNF)
- 4. Nach innen schieben von $\vee$
  $(\neg p \vee q \vee r) \ \wedge \ (\neg q \vee p) \ \wedge \ (\neg r \vee p)$ (KNF)

Exkurs: Disjunktive Normalform

- Anmerkung: Analog zur konjunktiven Normalform gibt es auch eine *disjunktive Normalform*
- Eine Formel in disjunktiver Normalform ist eine Disjunktion (*Veroderung*) von Konjunktionen (*und*-verknüpften Literalen)
- Beispiel: Betrachte $F = a \wedge (b \rightarrow \neg c)$
  - $KNF(F) = a \wedge (\neg b \vee \neg c)$
  - $DNF(F) = (a \wedge \neg b) \vee (a \wedge \neg c)$
- An der DNF einer aussagenlogischen Formel kann man die Erfüllbarkeit direkt ablesen!
  - Jede Konjunktion entspricht einer Menge von Modellen, ähnlich wie bei bei offenen Tableaux-Ästen
- Damit: DNF-Berechnung ist ein Entscheidungsverfahren für Aussagenlogik!
  - Aber: Konvertierung in DNF ist potentiell teuer (und in der Regel viel teurer als andere Entscheidungsverfahren)!
  - Deswegen: DNF ist praktisch wesentlich weniger relevant

Übung: KNF Transformation

Transformieren Sie die folgenden Formeln in konjunktive Normalform:

- $\neg(a \vee b) \leftrightarrow (b \wedge (a \rightarrow \neg c))$
- $(A \vee B \vee C) \wedge (A \rightarrow ((B \wedge \neg C) \vee (\neg B \wedge C))) \wedge (\neg B \rightarrow \neg C) \wedge \neg(B \wedge C \wedge \neg A) \wedge (\neg C \rightarrow \neg B)$

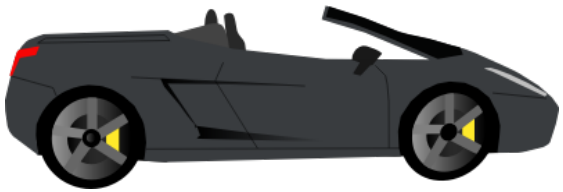
Tipps:

- Wenn die Formel auf der obersten Ebene bereits eine Konjunktion („und“-verknüpft) ist, kann man die entsprechenden 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.

Lösung zu Teil 2 Ende Vorlesung 16

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

Modellierung

- Elementare Aussagen:
  - c: Cabrio
  - s: Schiebedach
  - w: Subwoofer
  - 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 Klauselmenge

Klauseln mal anders

Definition (Klauseln als Mengen)

Eine Klausel ist eine Menge von Literalen.

- Beispiele
  - $C_1 = \{\neg c, \neg s\}$
  - $C_3 = \{\neg d, q, c\}$
  - $C_4 = \{\neg q, v\}$
- Literale einer Klausel sind (implizit) oder-verknüpft
- Schreibweise
  - $C_1 = \neg c \vee \neg s$
  - $C_3 = \neg d \vee q \vee c$
  - $C_4 = \neg q \vee v$

Es besteht eine einfache Beziehung zwischen Klauseln als expliziten Disjunktionen und Klauseln als Mengen - wir unterscheiden die beiden Sichten nur, wenn notwendig!

Interpretationen mal anders

Definition (Interpretationen als Literalmengen)

- Eine Interpretation ist eine Menge von Literalen mit:
  - Für jedes $p \in \Sigma$ ist entweder $p \in I$ oder $\neg p \in I$
  - Für kein $p \in \Sigma$ 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:

$$I(a) = \begin{cases} 0 & \text{falls} \quad \neg a \in I \\ 1 & \text{falls} \quad a \in I \end{cases}$$

Semantik: Beispiele

- Atome: $\Sigma = \{c, s, w, d, q, v\}$
- Klauseln:
  - $C_1 = \neg c \vee \neg s$
  - $C_3 = \neg d \vee q \vee c$
  - $C_4 = \neg q \vee v$
- Interpretationen:
  - $I_1 = \{c, s, w, d, q, v\}$
  - $I_2 = \{\neg c, s, \neg w, d, \neg q, v\}$

- $I_1(C_1) = 0$
- $I_1(C_3) = 1$
- $I_1(C_4) = 1$
- $I_2(C_1) = 1$
- $I_2(C_3) = 0$
- $I_2(C_4) = 1$

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 entprechend angepasst.

Damit entspricht eine Klauselmenge einer Formel in konjunktiver Normalform. Für die Mengenschreibweise hat sich der Begriff Klauselnormalform eingebürgert. Beide Begriffe werden oft synonym verwendt.

Die leere Klausel

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.

Resolution: Grundidee

- 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	**Mengenschreibweise**
$\dfrac{C \vee p \qquad D \vee \neg p}{C \vee D}$	$\dfrac{C \uplus \{p\} \qquad D \uplus \{\neg p\}}{C \cup D}$

- 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
  - Es gilt: $\{C \vee p, D \vee \neg p\} \models C \vee D$

Automobile in Klauselform

- Elementare Aussagen:
  - c: Cabrio
  - s: Schiebedach
  - w: Subwoofer
  - 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.

Korrektheit und Vollständigkeit

Satz: (Resolution ist korrekt)

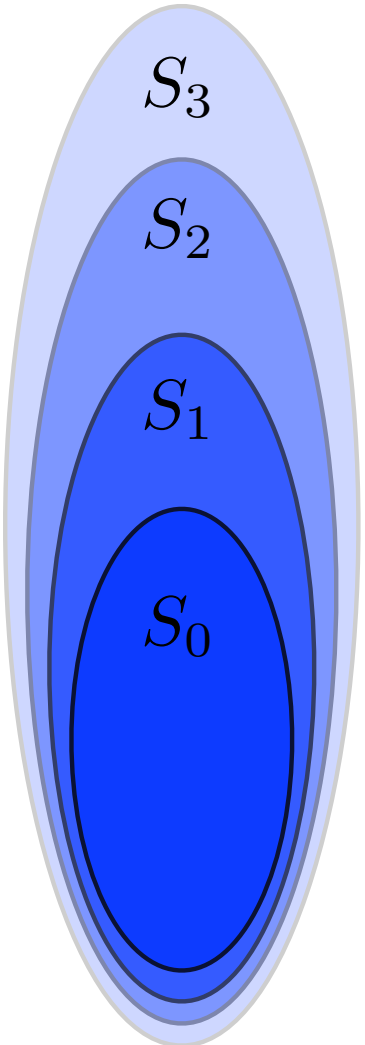
- Wenn aus S die leere Klausel ableitbar ist, so ist S unerfüllbar

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

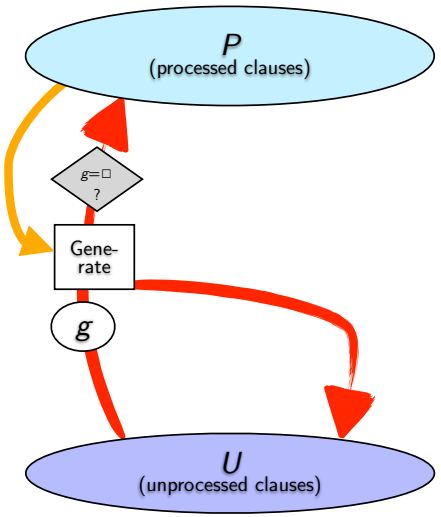
Saturierungs-Algorithmen (1)

- Ziel: Systematische Saturierung von S
- Algorithmus 1: Level-Saturation
  1. Berechne die Menge aller Resolventen R mit zwei Prämissen aus S
  2. Prüfe, ob $\square \in R$. Ja: S ist unerfüllbar
  3. Prüfe, ob $R \subseteq S$. Ja: S ist saturiert und erfüllbar
  4. Setze $S := S \cup R$, weiter bei 1
- Vorteile:
  - Einfach zu erklären
  - Garantiert fair und vollständig
- Nachteil:
  - S wächst sehr schnell
  - Keine Kontrolle über die Suche
  - Konsequenz: Nur sehr kurze Beweise können gefunden werden!



Saturierungs-Algorithmen (2)

- Algorithmus 2: *Given-Clause-Algorithmus*
  1. $(P, U) = (\emptyset, S)$
  2. Prüfe, ob $U = \emptyset$. Falls ja: P ist saturierte Version von S und erfüllbar
  3. Wähle (fair und clever) g (die *given clause*) aus U
  4. Ist $g = \square$? Falls ja: S is unerfüllbar
  5. Berechne alle Resolventen R bei denen g eine Prämisse ist, die andere aus P kommt
  6. $U := (U \backslash \{g\}) \cup R$; $P := P \cup \{g\}$
  7. Weiter bei 2
- Vorteile:
  - Vollständig, falls Wahl von g fair ist
  - Einfach und effizient zu implementieren
  - Gut zu steuern (wähle g geschickt)
  - Redundanzvermeidung ist leicht zu integrieren
- Nachteil:
  - Komplizierter zu erklären



Spickzettel Widerspruchskalküle

1. Fragestellung: Ist Formel F (oder Formelmenge) unerfüllbar?
   - Methode 1: Wahrheitstafel
   - Methode 2: Tableaux-Methode - geschlossenes Tableau impliziert Unerfüllbarkeit von F
   - Methode 3: KNF-Transformation, Resolution. Leere Klausel herleitbar: F ist unerfüllbar
2. Fragestellung: Ist F allgemeingültig?
   - ...das gilt gdw. $\neg F$ unerfüllbar ist. Weiter bei 1.
3. Fragestellung: Gilt $F_1, \ldots, F_n \models G$?
   - ...das gilt gdw. $F = F_1 \wedge \ldots \wedge F_n \rightarrow G$ allgemeingültig ist. Weiter bei 2.
   - Abkürzung: das gilt gdw. $F = F_1 \wedge \ldots \wedge F_n \wedge \neg G$ unerfüllbar ist. Weiter bei 1.
4. Fragestellung: Gilt $G \equiv G'$?
   - ...das gilt gdw. $G \leftrightarrow G'$ allgemeingültig ist. Weiter bei 2.

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!

Ende Vorlesung 17

Prädikatenlogik

Grenzen der Aussagenlogik

Alle Menschen sind sterblich
Sokrates ist ein Mensch

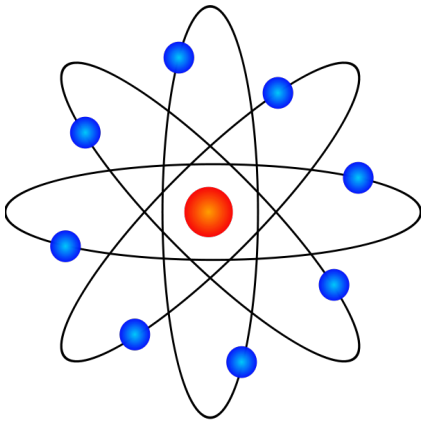
Also ist Sokrates sterblich

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
- Prädikatssymbole repräsentieren Relationen
- Funktionssymbole repräsentieren Funktionen
- Variablen stehen für beliebige Objekte (Universelle und existentielle Quantifizierung)



Atomspaltung!

- Damit:
$$\frac{\forall X(\mathit{mensch}(X) \rightarrow \mathit{sterblich}(X)) \quad \mathit{mensch}(\mathit{sokrates})}{\mathit{sterblich}(\mathit{sokrates})}$$

Prädikatenlogik: Reichere Struktur

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, ...

Syntax der Prädikatenlogik: Signatur

Definition (Prädikatenlogische Signatur)

Eine Signatur Σ ist ein 3-Tupel (P, F, V). Dabei sind P, F, V paarweise disjunkte Mengen von Symbolen. Insbesondere:

- P ist eine endliche oder abzählbare Menge von Prädikatssymbolen mit assozierter Stelligkeit
- F ist eine endliche oder abzählbare Menge von Funktionssymbolen mit assoziierter Stelligkeit
- V ist eine abzählbar unendliche Menge von Variablen

- Wir schreiben $p/n \in P$, um auszudrücken, dass p ein n-stelliges Prädikatssymbol ist
- Analog schreiben wir $f/n \in F$ für ein n-stelliges Funktionssymbol f.

Prädikatenlogische Signatur - Erläuterungen

Wir betrachten Signatur $\Sigma = (P, F, V)$

- P: Menge der Prädikatssymbole mit Stelligkeit
  - Z.B. $P = \{\mathit{mensch}/1, \mathit{sterblich}/1, \mathit{lauter}/2\}$
  - n-stellige Prädikatssymbole repräsentieren n-stellige Relationen
  - Relationen bilden Werte aus dem Universum (genauer: Tupel) auf die Wahrheitswerte ab
- F: Menge der Funktionssymbole mit Stelligkeit
  - Z.B. $F = \{\mathit{gruppe}/2, \mathit{lehrer}/1, \mathit{sokrates}/0, \mathit{aristoteles}/0\}$
    - Konstanten (z.B. *sokrates*) sind 0-stellige Funktionssymbole!
  - Funktionen bilden Werte des Universums auf Werte des Universums ab
- V: Abzählbar unendliche Menge von Variablen
  - Z.B. $V = \{X, Y, Z, U, X_1, X_2, \ldots\}$
    - In Prolog und TPTP: Variablen fangen mit Großbuchstaben an
    - Literatur: Oft $\{x, y, z, u, v, x_0, x_1, \ldots\}$
  - Variablen stehen potentiell für beliebige Werte des Universums.

Terme

Definition (Terme)

Sei $\Sigma = (P, F, V)$ eine prädikatenlogische Signatur. Die Menge T_Σ der Terme über F, V ist definiert wie folgt:

- Sei $X \in V$. Dann $X \in T_\Sigma$
- Sei $f/n \in F$ und seien $t_1, \ldots, t_n \in T_\Sigma$. Dann ist $f(t_1, \ldots, t_n) \in T_\Sigma$
- T_Σ ist die kleinste Menge mit diesen Eigenschaften

- Beispiele:
  - $F = \{gruppe/2, lehrer/1, sokrates/0, aristoteles/0, \ldots\}$
  - Terme:
    - X
    - $sokrates()$ (normalerweise ohne Klammern geschrieben)
    - $gruppe(X, sokrates)$
    - $gruppe(lehrer(sokrates), gruppe(sokrates, Y))$

Terme

Bemerkungen

- Insbesondere sind alle Konstanten in T_Σ
  - 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

(Prädikatenlogische) Atome

Definition (Atome)

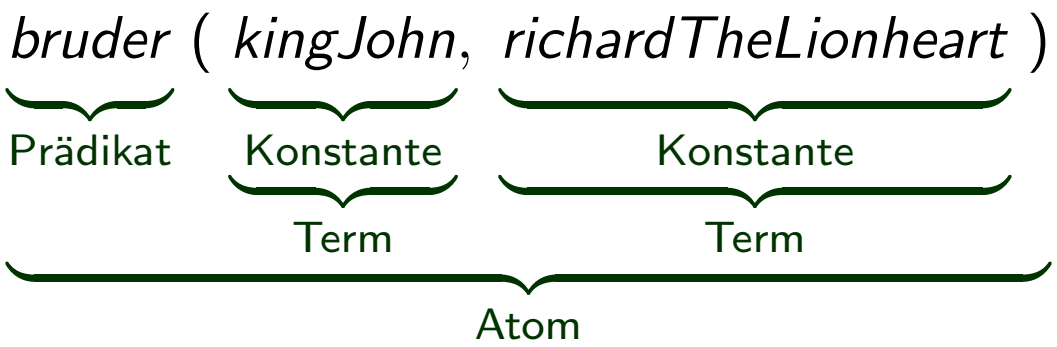
Sei $\Sigma = (P, F, V)$ eine prädikatenlogische Signatur. Die Menge A_Σ der Atome über P, F, V ist definiert wie folgt:

- Sei $p/n \in P$ und seien $t_1, \ldots, t_n \in T_\Sigma$. Dann ist $p(t_1, \ldots, t_n) \in A_\Sigma$
- A_Σ ist die kleinste Menge mit diesen Eigenschaften

- Beispiele:
  - $P = \{mensch/1, sterblich/1, lauter/2\}$,
    $F = \{gruppe/2, lehrer/1, sokrates/0, aristoteles/0\}$
- Atome:
  - $mensch(X)$
  - $lauter(sokrates, X)$
  - $lauter(sokrates, sokrates)$
  - $lauter(sokrates, lehrer(lehrer(lehrer(X))))$

Atome

Beispiel



Syntax der Prädikatenlogik: Atomare Formeln

Beispiel

$$\underbrace{>}_{\text{Prädikat}}\ (\ \underbrace{\underbrace{\mathit{laenge}}_{\text{Funktion}}(\underbrace{\mathit{linkesBein}}_{\text{Funktion}}(\underbrace{\mathit{richard}}_{\text{Konstante}}))}_{\text{Term}},\ \underbrace{\underbrace{\mathit{laenge}}_{\text{Funktion}}(\underbrace{\mathit{linkesBein}}_{\text{Funktion}}(\underbrace{\mathit{kingJohn}}_{\text{Konstante}}))}_{\text{Term}}\ )}_{\text{Atom}}$$

Übung

Sei $\Sigma = (\{gt/2, eq/2\}, \{s/1, 0/0\}, \{x, y, z, \ldots\})$

- 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: Logische Zeichen

Wie in der Aussagenlogik:

- $\top$ Symbol für den Wahrheitswert „wahr“
- $\bot$ 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“)

Syntax der Prädikatenlogik: Komplexe Formeln

Definition (Formeln der Prädikatenlogik 1. Stufe)

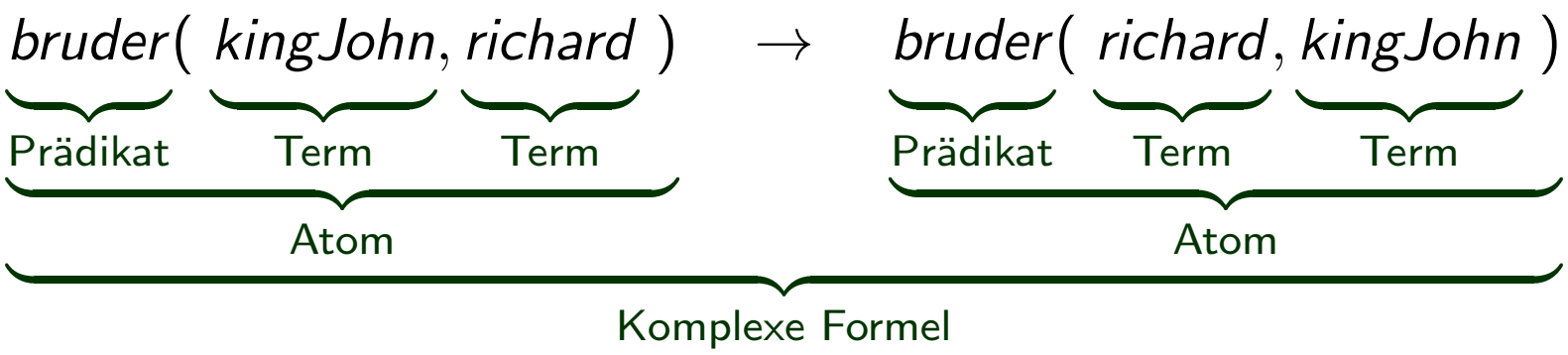
Sei $\Sigma = (P, F, V)$ eine prädikatenlogische Signatur.

For_Σ ist die kleinste Menge mit:

- $A_\Sigma \subseteq For_\Sigma$ (alle Atome sind Formeln)
- $\top \in For_\Sigma$ und $\bot \in For_\Sigma$
- Wenn $A, B \in For_\Sigma$, dann auch
  $(\neg A)$, $(A \wedge B)$, $(A \vee B)$, $(A \to B)$, $(A \leftrightarrow B)$ $\in For_\Sigma$
- Wenn $A \in For_\Sigma$ und $x \in V$, dann
  $\forall x A$, $\exists x A$ $\in For_\Sigma$

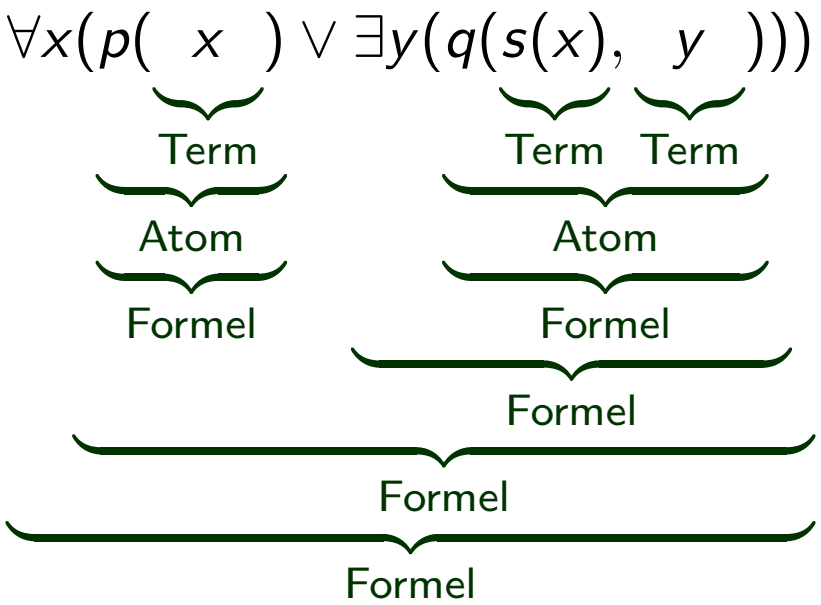
Syntax der Prädikatenlogik: Komplexe Formeln

Beispiel



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“

Lösung

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

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!)

Syntax der Prädikatenlogik: Bemerkungen

- Nach unserer Definition werden alle Funktions- und Prädikatssymbole in Prefix-Notation verwendet:
  - $=(1,2), \mathit{even}(2), +(3,5), \mathit{multiply}(2,3)$
- Konvention: Zweistellige Symbole mit bekannter Semantik werden gelegentlich Infix geschrieben
  - Insbesondere das Gleichheitsprädikat $=$
  - Im Bereich SMT auch $>, +, *, \leq, \ldots$
- Die Quantoren $\exists, \forall$ haben die höchste Priorität
  - Ein Quantor *bindet* immer die kleinstmögliche vollständige Formel
  - Folgt automatisch aus strikter Syntax
  - Relevant, wenn wir wieder Klammern weglassen

Mehr Beispiele

- $\forall X((X < \text{alexander}) \vee (X = \text{alexander}))$
  - Keiner (genauer: kein anderer) ist so groß wie Alexander
  - Alternativ $\neg\exists\, X(\neg(X = \mathit{alexander}) \wedge \neg(\text{alexander} > X))$
- "Es kann nur einen geben" (wörtlicher: "Es gibt nur einen")
  - $\exists Y \forall X(X = Y)$
  - $\forall X(X = \text{connormcloud})$
- "Die Garde stirbt, doch sie ergibt sich nicht!"
  - $\text{stirbt(garde)} \wedge \neg\, \text{kapituliert(garde)}$

Semantik?

Semantik der Prädikatenlogik

Definition (Prädikatenlogische Interpretation)

Eine (prädikatenlogische) Interpretation ist ein Paar $\mathfrak{J} = \langle U, I \rangle$, wobei gilt:

- U ist eine nicht-leere Menge (das Universum)
- I ist eine Interpretationsfunktion – sie interpretiert
  - (freie) Variablen durch ein Element des Universums
  - Prädikatssymbole durch eine Relation auf dem Universum (mit passender Stelligkeit)
  - Funktionssymbole durch eine Funktion auf dem Universum (mit passender Stelligkeit)

Semantik der Prädikatenlogik

Anmerkungen

Sei $\mathfrak{J} = \langle U, I \rangle$ eine prädikatenlogische Interpretation und $\Sigma = (P, F, V)$. Dann gilt:

- Für $f/n \in F$ ist $I(f)$ eine Funktion $U^n \to U$.
- Für $p/n \in P$ ist $I(p)$ eine Relation $\subseteq U^n$
  - Alternative Sichtweise: $I(p)$ ist Funktion $U^n \to \{0, 1\}$
  - ... mit $I(p)(e_1, ..., e_n) = 1$ gdw $(e_1, ..., e_n) \in I(p)$
  - Stichwort: *Charakteristische Funktion*
- Für $x \in V$ gilt: $I(x) \in U$

Semantik der Prädikatenlogik

Bemerkungen

- Im allgemeinen ist das Universum U einen prädikatenlogischen Modells unendlich
  - In dem Fall ist auch die Menge aller Interpretationsfunktionen unendlich!
- Auch schon für ein endliches U gibt es eine riesige Zahl verschiedener Interpretationsfunktionen

Die Wahrheitstafelmethode funktioniert für Prädikatenlogik sicher nicht!

Semantik der Prädikatenlogik (0)

Definition (Semantik eines Terms t)

Sei eine prädikatenlogische Interpretation $\mathfrak{I} = \langle U, I \rangle$ gegeben.

Die Semantik von $t \in \mathit{Term}_\Sigma$ ist das Element $I(t)$ aus U, das rekursiv definiert ist durch

- Ist $t = x$ eine Variable: $I(t) = I(x)$
- Ist $t = c$ eine Konstante: $I(t) = I(c)$
- Ist $t = f(t_1, \ldots, t_n)$: $I(t) = I(f)(I(t_1), \ldots, I(t_n))$

Beispiel: Semantik von Termen

- Sei $F = \{f/2, g/1, a/0, b/0\}$ und $V = \{x, y, \ldots\}$
- Sei $\mathfrak{J} = \langle \mathbb{N}, I \rangle$ mit:
  - $I(f) = (x, y) \mapsto x + y$
  - $I(g) = x \mapsto 2x$
  - $I(a) = 3$
  - $I(b) = 12$
- $I(g(a)) = I(g)(I(a)) = I(g)(3) = 6$
- $I(f(g(a), b)) = I(f)(I(g(a)), I(b))$
  $= I(f)(I(g)(I(a)), I(b))$
  $= I(f)(I(g)(3)), 12)$
  $= I(f)(6, 12)$
  $= 18$

Semantik der Prädikatenlogik (1)

Definition (Semantik einer Formel (1))

Sei eine prädikatenlogische Interpretation $\mathfrak{J} = \langle U, I \rangle$ gegeben.

Die Semantik $I(F)$ einer Formel F unter I ist einer der Wahrheitswerte 1 oder 0 und definiert wie folgt:

$$\begin{aligned} I(\top) &= 1 \\ I(\bot) &= 0 \\ I(p(t_1, \ldots, t_n)) &= I(p)(I(t_1), \ldots, I(t_n)) \end{aligned}$$

...

Semantik der Prädikatenlogik (2)

Definition (Semantik einer Formel (2))

und (wie in der Aussagenlogik):

$$I(\neg F) = \begin{cases} 0 & \text{falls} \quad I(F) = 1 \\ 1 & \text{falls} \quad I(F) = 0 \end{cases}$$

...

Semantik der Prädikatenlogik (3)

Definition (Semantik einer Formel (3))

und (wie in der Aussagenlogik):

$$I(F \wedge G) = \begin{cases} 1 & \text{falls} \quad I(F) = 1 \text{ und } I(G) = 1 \\ 0 & \text{sonst} \end{cases}$$

$$I(F \vee G) = \begin{cases} 1 & \text{falls} \quad I(F) = 1 \text{ oder } I(G) = 1 \\ 0 & \text{sonst} \end{cases}$$

...

Semantik der Prädikatenlogik (4)

Definition (Semantik einer Formel (4))

und (wie in der Aussagenlogik):

$$I(F \to G) = \begin{cases} 1 & \text{falls} \quad I(F) = 0 \text{ oder } I(G) = 1 \\ 0 & \text{sonst} \end{cases}$$

$$I(F \leftrightarrow G) = \begin{cases} 1 & \text{falls} \quad I(F) = I(G) \\ 0 & \text{sonst} \end{cases}$$

...

Semantik der Prädikatenlogik (5)

Definition (Semantik einer Formel (5))

und zusätzlich:

$$I(\forall x F) = \begin{cases} 1 & \text{falls} \quad I_{x/d}(F) = 1 \text{ für alle } d \in U \\ 0 & \text{sonst} \end{cases}$$

$$I(\exists x F) = \begin{cases} 1 & \text{falls} \quad I_{x/d}(F) = 1 \text{ für mindestens ein } d \in U \\ 0 & \text{sonst} \end{cases}$$

wobei

$I_{x/d}$ identisch zu I mit der Ausnahme, dass $I_{x/d}(x) = d$,

d.h. I wird so abgewandelt, dass x den Wert d bekommt.

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 $I(gt) =<, I(plus) = +, I(1) = 1$. Bestimmen Sie $I(F)$.

Arbeitsblatt | Ende Vorlesung 18

Beispiele: Familienverhältnisse

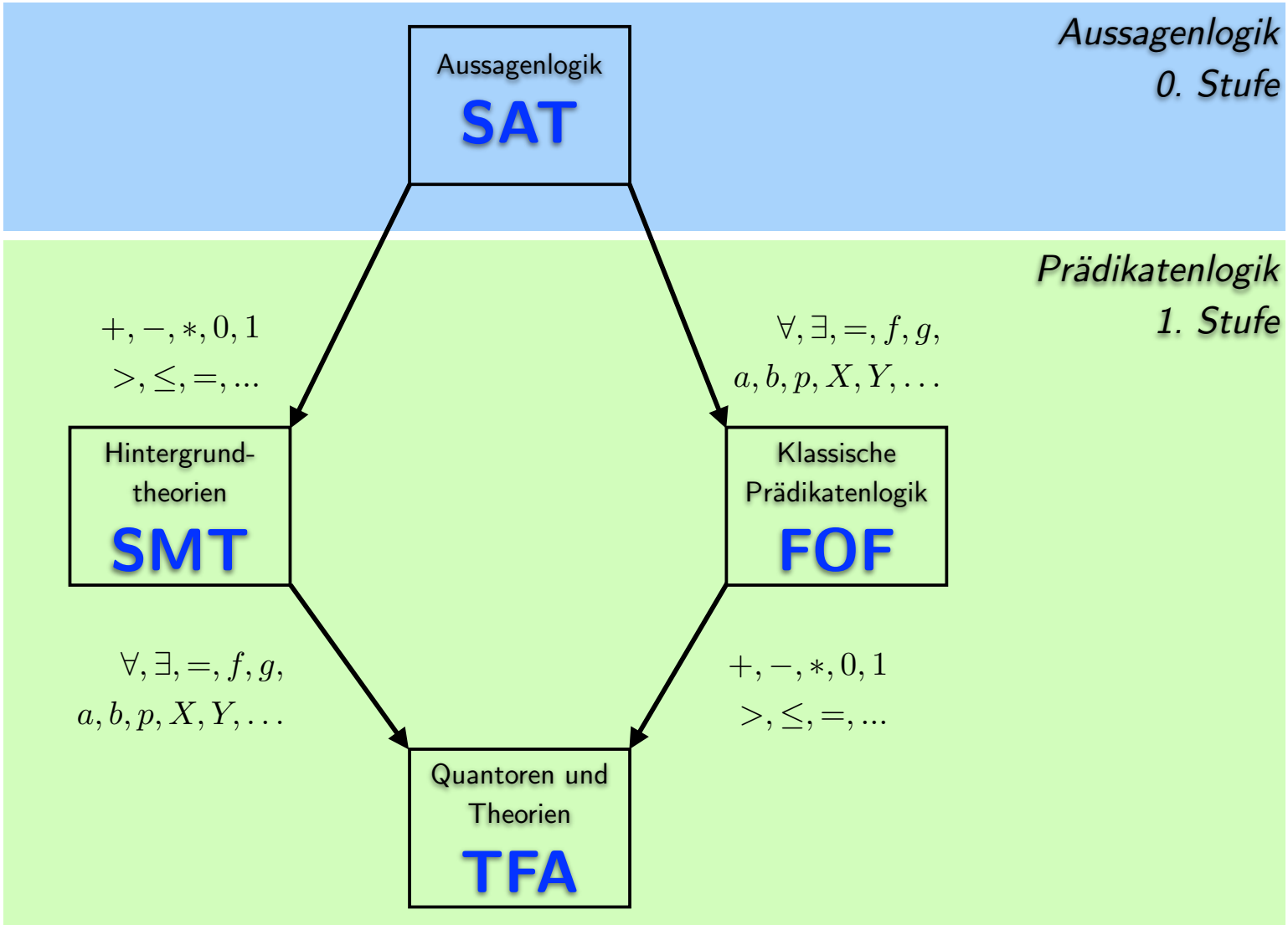
- „Brüder sind Geschwister“
  $\forall x \forall y\ (\mathit{bruder}(x, y) \rightarrow \mathit{geschwister}(x, y))$
- „geschwister“ ist symmetrisch
  $\forall x \forall y\ (\mathit{geschwister}(x, y) \leftrightarrow \mathit{geschwister}(y, x))$
- „Mütter sind weibliche Elternteile“
  $\forall x \forall y\ (\mathit{mutter}(x, y) \leftrightarrow (\mathit{weiblich}(x) \wedge \mathit{elter}(x, y)))$
- „Ein Cousin ersten Grades ist
  das Kind eines Geschwisters eines Elternteils“
  $\forall x \forall y\ (\mathit{cousin}1(x, y) \leftrightarrow$
  $\exists p \exists ps\ (\mathit{elter}(p, x) \wedge \mathit{geschwister}(ps, p) \wedge \mathit{elter}(ps, y)))$

Übung: Primzahlen

- Formalisieren Sie das Prädikat *Primzahl* und die Aussage „Zu jeder Primzahl gibt es eine größere Primzahl“.
  - Sie können die Prädikatssymbole $= /2$, $> /2$ und die Funktionssymbole $mult/2$ und $1/0$ als gegeben betrachten.
- 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.

Lösung

Prädikatenlogik im Kontext



Wichtige Begriffe für Prädikatenlogik

Definition

- Modell
- Allgemeingültigkeit
- Erfüllbarkeit
- Unerfüllbarkeit
- Logische Folgerung
- Logische Äquivalenz

sind für klassische Prädikatenlogik genauso definiert wie für Aussagenlogik
(nur mit prädikatenlogischen Interpretationen/Modellen)

Variablenmengen von Objekten

Definition (Variablenmengen)

Sei $t \in T_\Sigma$, $a \in A_\Sigma$, $F \in For_\Sigma$.

- $Vars(t)$ ist die Menge der Variablen, die in t vorkommen
- $Vars(a)$ ist die Menge der Variablen, die in a vorkommen
- $Vars(F)$ ist die Menge der Variablen, die in F vorkommen
- $FVars(F)$ ist die Menge der *freien* Variablen, die in F vorkommen

- Beispiel:
  - $t = f(X, g(f(Y, Z)))$. $Vars(t) = \{X, Y, Z\}$
  - $F = \forall X(p(X, Y) \rightarrow \exists Z(p(X, Z)))$
    - $Vars(F) = \{X, Y, Z\}$
    - $FVars(F) = \{Y\}$

Grund- und geschlossene Formeln

Definition (Grundterme, Atome, Formeln)

- Ein Term t heißt grund, wenn $\mathit{Vars}(t) = \emptyset$.
- Ein Atom a heißt grund, wenn $\mathit{Vars}(a) = \emptyset$.
- Eine Formel F heißt grund, wenn $\mathit{Vars}(F) = \emptyset$.

Definition (Geschlossene Formeln)

Eine Formel $F \in \mathit{For}_\Sigma$ heißt geschlossen, falls $\mathit{FVars}(F) = \emptyset$

Prädikatenlogik: Folgerung und Unerfüllbarkeit

- Wir betrachten nur geschlossene Formeln (bei denen alle Variablen gebunden sind)

Satz: (Deduktionstheorem für die Prädikatenlogik)

Seien $F_1, \ldots, F_n, G \in For_\Sigma$ geschlossene Formeln. Dann gilt:

$$F_1, \ldots, F_n \models G$$
gdw.
$$\models (F_1 \wedge \ldots \wedge F_n) \rightarrow G$$
gdw.
$$F_1 \wedge \ldots \wedge F_n \wedge \neg G \text{ ist unerfüllbar}$$

Wie können wir die Unerfüllbarkeit in der Prädikatenlogik zeigen?

Plan: Resolution für Prädikatenlogik

- Konvertierung in Negationsnormalform
- Konvertierung in Prenex-Normalform
- Skolemisierung: Entfernung von existenzquantifizierten Variablen
- Konvertierung in konjunktive Normalform/Klauselnormalform
- Resolutionskalkül für prädikatenlogische Klauseln
  - Unifikation
  - Faktorisierung
  - Resolution

Eigenschaften von Quantoren

Quantoren gleicher Art kommutieren

$$\begin{array}{ccc} \forall x \forall y & \text{ist äquivalent zu} & \forall y \forall x \\ \exists x \exists y & \text{ist das gleiche wie} & \exists y \exists x \end{array}$$

Beispiel

$$\forall x \forall y \; equal(mult(0, x), mult(0, y))$$

$$\equiv$$

$$\forall y \forall x \; equal(mult(0, x), mult(0, y))$$

Eigenschaften von Quantoren

Verschiedene Quantoren kommutieren NICHT

$\exists x \forall y$ ist nicht äquivalent zu $\forall y \exists x$

Beispiel

$\exists x \forall y \; \mathit{loves}(x, y)$
Es gibt eine Person, die jeden Menschen in der Welt liebt
(einschließlich sich selbst)

$\forall y \exists x \; \mathit{loves}(x, y)$
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

$\forall x \exists y \; \mathit{mutter}(y, x)$
Jeder hat eine Mutter (richtig)

$\exists y \forall x \; \mathit{mutter}(y, x)$
Es gibt eine Person, die die Mutter von jedem ist (falsch)

Eigenschaften von Quantoren

Dualität der Quantoren

$\forall x \ldots$ ist äquivalent zu $\neg\exists x \neg \ldots$

$\exists x \ldots$ ist äquivalent zu $\neg\forall x \neg \ldots$

Beispiel

$\forall x\ mag(x, eiskrem)$ ist äquivalent zu $\neg\exists x\ \neg mag(x, eiskrem)$

$\exists x\ mag(x, broccoli)$ ist äquivalent zu $\neg\forall x\ \neg mag(x, broccoli)$

Eigenschaften von Quantoren

$\forall$ distributiert über $\wedge$

$\forall x\,(\ldots \wedge \ldots)$ ist äquivalent zu $(\forall x \ldots) \wedge (\forall x \ldots)$

Beispiel

$\forall x\,(\mathit{studiert}(x) \wedge \mathit{arbeitet}(x))$ ist äquivalent zu
$(\forall x\ \mathit{studiert}(x)) \wedge (\forall x\ \mathit{arbeitet}(x))$

Eigenschaften von Quantoren

$\exists$ distributiert über $\vee$

$\exists x\,(\ldots \vee \ldots)$ ist äquivalent zu $(\exists x \ldots) \vee (\exists x \ldots)$

Beispiel

$\exists x\,(\mathit{eiskrem}(x) \vee \mathit{broccoli}(x))$ ist äquivalent zu
$(\exists x\ \mathit{eiskrem}(x)) \vee (\exists x\ \mathit{broccoli}(x))$

Eigenschaften von Quantoren

$\forall$ distributiert NICHT über $\vee$

$\forall x\,(\ldots \vee \ldots)$ ist NICHT äquivalent zu $(\forall x \ldots) \vee (\forall x \ldots)$

Beispiel

$\forall x\,(\mathit{eiskrem}(x) \vee \mathit{broccoli}(x))$ ist NICHT äquivalent zu
$(\forall x\ \mathit{eiskrem}(x)) \vee (\forall x\ \mathit{broccoli}(x))$

Übung: Substitutionen

Sei $\sigma = \{X \leftarrow a, Y \leftarrow f(X, Y), Z \leftarrow g(a)\}$. Berechnen Sie:

- $\sigma(h(X, X, X))$
- $\sigma(f(g(X), Z))$
- $\sigma(f(Z, g(X)))$
- $\sigma(g(Y))$
- $\sigma \circ \sigma(g(Y))$ $(= \sigma(\sigma(g(Y))))$
- $\sigma(p(X) \vee \neg q(X, f(Y, Z)) \vee p(a))$

Fällt Ihnen etwas auf?

Operationen auf Substitutionen

- Substitutionen sind Funktionen (und Relationen). Als solche können sie verknüpft werden. Seien σ und τ zwei Substitutionen. Dann ist $\sigma \circ \tau$ ebenfalls eine Substitution
  - Es gilt $\sigma \circ \tau(X) = \sigma(\tau(X))$ für alle $X \in V$
  - Allgemeiner: $\sigma \circ \tau(t) = \sigma(\tau(t))$ für $t \in T_\Sigma(F, V)$
  - Man wendet erst die "innere" Substitution an, dann die "äußere" auf das Ergebnis
- $\sigma \circ \tau$ kann aber auch explizit berechnet werden
  - $\sigma \circ \tau = \{X \leftarrow \sigma(\tau(X)) \mid X \in \mathrm{Dom}(\tau)\} \cup \{X \leftarrow \sigma(X) \mid X \in \mathrm{Dom}(\sigma), X \notin \mathrm{Dom}(\tau)\}$
- Beispiele:
  - $\sigma = \{X \leftarrow f(X, Y), Y \leftarrow g(X)\}$
  - $\tau = \{Y \leftarrow f(X, b), Z \leftarrow a\}$
  - $\sigma \circ \tau = \{Y \leftarrow f(f(X, Y), b), Z \leftarrow a, X \leftarrow f(X, Y)\}$
  - $\tau \circ \sigma = \{X \leftarrow f(X, f(X, b)), Y \leftarrow g(X), Z \leftarrow a\}$

Instanzen

Definition (Instanzen)

Sei t ein Term, a ein Atom, l ein Literal, c eine Klausel und σ 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 eine Instanz keine Variablen (mehr) enthält, so heißt sie auch eine Grundinstanz.

Satz von Herbrand (2)

Satz: (Korrolar zum Satz von Herbrand)

- Eine Menge von Klauseln ist genau dann unerfüllbar, wenn die Menge aller ihrer Grundinstanzen (aussagenlogisch) unerfüllbar ist.
- Per *Kompaktheitstheorem*: Dann gibt es eine endliche Menge von Grundinstanzen, die (aussagenlogisch) unerfüllbar ist.

- Dabei betrachten wir die Grundatome als aussagenlogische Variablen
- Problem: Die Menge *aller* 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
  - 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.

Problem: Gleichheit der Atome reicht nicht mehr
Lösung: Finde geeignete Ersetzungen für Variablen

Wie finden wir eine solche Substitution σ automatisch?

Unifikation

Definition (Unifikator)

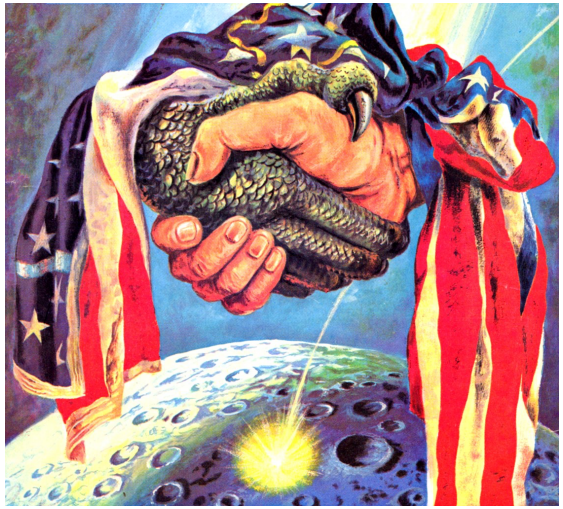
- Seien $s, t \in T_\Sigma$ zwei Terme
- Seien $a, b \in A_\Sigma$ zwei Atome

Ein Unifikator für s, t bzw. a, b ist eine Substitution σ mit $\sigma(s) = \sigma(t)$ bzw. $\sigma(a) = \sigma(b)$.
Der allgemeinste Unifikator für s, t bzw. a, b wird mit $mgu(s, t)$ bzw. $mgu(a, b)$ bezeichnet

- Fakt: Wenn ein Unifikator existiert, so gibt es einen (bis auf Variablenumbenennungen) eindeutigen MGU (*most general unifier*)
- Fakt: (Allgemeinste) Unfikatoren können systematisch gefunden werden

Beobachtungen zum Finden von Unifikatoren

1. $sterblich(X), mensch(X)$ können nie unifiziert werden
   - Das erste Symbol unterscheidet sich immer
2. $X, lehrer(X)$ können nie unifiziert werden
   - Egal, was für X eingesetzt wird, ein *lehrer* bleibt immer über
   - "Occurs-Check"
3. $X, lehrer(sokrates)$ werden mit $\sigma = \{X \leftarrow lehrer(sokrates)\}$ unifiziert
   - 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: $R = \{s_1 = t_1, s_2 = t_2, \ldots, s_n = t_n\}$
  - Suche gemeinsamen Unifikator σ mit
    - $\sigma(s_1) = \sigma(t_1)$
    - $\sigma(s_2) = \sigma(t_2)$
    - $\ldots$
    - $\sigma(s_n) = \sigma(t_n)$
- Verwende Transformationssystem
  - Zustand: R, σ
    - R: Menge von Termpaaren
    - σ: Kandidat des Unifikators
  - Anfangszustand zum Finden von mgu(s, t): $\{s = t\}, \{\}$
  - Termination: $\{\}, \sigma$

Unifikation: Transformationssystem

Löschen: $\dfrac{\{t = t\} \cup R, \sigma}{R, \sigma}$

Binden: $\dfrac{\{x = t\} \cup R, \sigma}{\{x \leftarrow t\}(R), \{x \leftarrow t\} \circ \sigma}$ falls $x \notin \mathrm{Vars}(t)$

Orientieren: $\dfrac{\{t = x\} \cup R, \sigma}{\{x = t\} \cup R, \sigma}$ falls t keine Variable ist

Zerlegen: $\dfrac{\{f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)\} \cup R, \sigma}{\{s_1 = t_1, \ldots, s_n = t_n\} \cup R, \sigma}$

Occurs: $\dfrac{\{x = t\} \cup R, \sigma}{\mathit{FAIL}}$ falls $x \in \mathrm{Vars}(t), t \neq x$

Konflikt: $\dfrac{\{f(s_1, \ldots, s_n) = g(t_1, \ldots, t_m)\} \cup R, \sigma}{\mathit{FAIL}}$ falls $f \neq g$

Beispiel

R	σ	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
$\{U = Z, g(g(Y))) = Z\}$	$\{X \leftarrow U\}$	Binden (U)
$\{g(g(Y)) = Z\}$	$\{X \leftarrow Z,$ $U \leftarrow Z\}$	Orientieren
$\{Z = g(g(Y))\}$	$\{X \leftarrow Z,$ $U \leftarrow Z\}$	Binden (Z)
$\{\}$	$\{X \leftarrow g(g(Y)),$ $U \leftarrow g(g(Y)),$ $Z \leftarrow g(g(Y))\}$	

Übung: Unifikation

Bestimmen Sie (falls möglich) die allgemeinsten Unifikatoren für die folgenden Termpaare. Falls es nicht möglich ist, geben Sie den Grund an:

- $f(X, a)$ und $f(g(a), Y)$
- $f(X, a)$ und $f(g(a), X)$
- $f(X, f(Y, X))$ und $f(g(g(a)), f(a, g(Z)))$

Lösung | Ende Vorlesung 20

Resolutionskalkül für die Prädikatenlogik

Der Resolutionskalkül für die Prädikatenlogik besteht aus zwei Inferenzregeln:

- Resolution: $\dfrac{C \vee p \qquad D \vee \neg q}{\sigma(C \vee D)}$ falls $\sigma = \mathrm{mgu}(p, q)$
- Faktorisieren: $\dfrac{C \vee p \vee q}{\sigma(C \vee p)}$ falls $\sigma = \mathrm{mgu}(p, q)$

Anmerkungen:

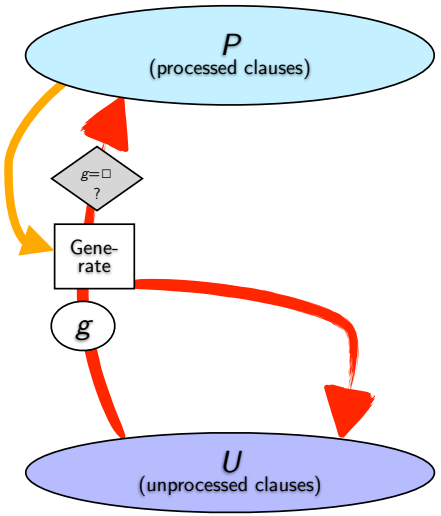
- Unterschiede zur Aussagenlogik:
  - Anwendung des Substitution, um p und q zu unifizieren
  - Faktorisierung ist in der Aussagenlogik implizit durch die Mengendarstellung
- Beachte:
  - Alle Klauseln sind implizit variablendisjunkt!

John Alan Robinson (1930–2016)
"A machine-oriented logic based on the resolution principle" (1965)

Anwendung des Resolutionskalküls

Anwendung: Leite systematisch und fair neue Klauseln aus einer Formelmenge ab.

- Wenn $\square$ (die leere Klausel) hergegleitet 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
  - $C_1 = \neg mensch(X) \vee sterblich(X)$
  - $C_2 = mensch(sokrates)$
  - $C_3 = \neg sterblich(sokrates)$
- Saturierung
  - Unifiziere $sterblich(sokrates)$ (C_3) und $sterblich(X)$ (aus C_1)
    - $\sigma = \{X \leftarrow sokrates\}$
    - $\sigma(C_1) = \neg mensch(sokrates) \vee sterblich(sokrates)$
    - $\sigma(C_3) = C_3$
    - Resolution zwischen $\sigma(C_1)$ und $\sigma(C_3)$: $C_4 = \neg mensch(sokrates)$
  - Resolution zwischen C_2 und C_4: $\square$
- Also: $\{C_1, C_2, C_3\}$ ist unerfüllbar - es gibt keine Möglichkeit, das Sokrates gleichzeitig menschlich und unsterblich ist

Übung: Tierische Resolution

- Axiome:
  - Alle Hunde heulen Nachts
    - $\forall X(h(X) \rightarrow l(X))$
  - Wer Katzen hat, hat keine Mäuse
    - $\forall X, Y(k(X) \wedge hat(Y, X) \rightarrow \neg\exists Z(hat(Y, Z) \wedge m(Z)))$
  - Empfindliche Menschen haben keine Tiere, die Nachts heulen
    - $\forall X(e(X) \rightarrow \neg\exists Y(hat(X, Y) \wedge l(Y)))$
  - John hat eine Katze oder einen Hund
    - $\exists X(hat(john, X) \wedge (h(X) \vee k(X)))$
  - John ist empfindlich
    - $e(john)$
- Behauptung:
  - John hat keine Mäuse
    - $\neg\exists X(hat(john, X) \wedge m(X))$
- 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
    - $\neg k(X) \vee \neg hat(Y, X) \vee \neg hat(Y, Z) \vee \neg m(Z)$
  - Empfindlichen Menschen haben keine Tiere, die Nachts heulen
    - $\neg e(X) \vee \neg hat(X, Y) \vee \neg l(Y)$
  - John hat eine Katze oder einen Hund (skolemisiert)
    - $hat(john, tier) \wedge (h(tier) \vee k(tier))$
  - John ist empfindlich
    - $e(john)$
- Negierte skolemisierte Behauptung
  - John hat Mäuse
    - $hat(john, maus) \wedge m(maus)$
- Formalisieren Sie das Problem und zeigen Sie die Behauptung per Resolution

Übung: Tierische Resolution

- Klauselmenge:
  1. $\neg h(X) \vee l(X)$
  2. $\neg k(X) \vee \neg hat(Y, X) \vee \neg hat(Y, Z) \vee \neg m(Z)$
  3. $\neg e(X) \vee \neg hat(X, Y) \vee \neg l(Y)$
  4. $hat(john, tier)$
  5. $h(tier) \vee k(tier)$
  6. $e(john)$
  7. $hat(john, maus)$
  8. $m(maus)$

Zeigen Sie per Resolution, dass dieses System unerfüllbar ist.

Lösung

Warum Faktorisieren?

- Betrachte folgende Klauselmenge M:
  1. $C_1 = p(X_0) \vee p(Y_0)$
  2. $C_2 = \neg p(X_1) \vee \neg p(Y_1)$
- Dann ist M unerfüllbar: Betrachte
  $\sigma = \{X_0 \leftarrow a, X_1 \leftarrow a, Y_0 \leftarrow a, Y_1 \leftarrow a\}$
  - $\sigma(C_1) = p(a) \vee p(a) = p(a)$, $\sigma(C_2) = \neg p(a) \vee \neg p(a) = \neg p(a)$
- Aber Resolution alleine liefert (wenn wir die Ergebnisse immer mit Variablen X, Y schreiben):
  3. $p(X) \vee \neg p(Y)$ (aus 1 und 2)
  4. $p(X) \vee p(Y)$ (aus 1 und 3 - das selbe wie 1)
  5. $\neg p(X) \vee \neg p(Y)$ (aus 2 und 3 - das selbe wie 2)
- Faktorisierung liefert
  6. $p(X)$
  7. $\neg p(Y)$

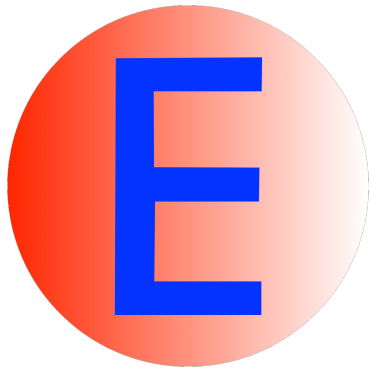
Übung: Resolution

Zeigen Sie per Resolution:

1. $p(0) \wedge \forall X(p(X) \rightarrow p(s(X))) \models p(s(s(0)))$
2. $p(0) \wedge \forall X(p(X) \rightarrow p(s(X))) \models p(s(s(s(0))))$
3. $p(0) \wedge \forall X(p(X) \rightarrow p(s(X))) \models p(s(s(s(s(0)))))$
4. ...

Abschluss

- 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
  - ...



SPASS

Vampire

Ende Vorlesung 21

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* $expr_1$ $expr_2$`)`	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?`	Gleichheit von beliebigen Objekten
`let`, `let*`	Einführung lokaler Variablen. Wert ist Wert des Rumpfes
`begin`	Sequenz. Wert ist Wert des letzten Ausdrucks der Sequenz

Scheme-Übersicht (3 - Listen 1)

`list`	Liste der Argumente
`'()`	Konstante für die leere Liste
`cons`	Listen-Konstruktor: Gibt ein `cons`-Paar mit seinen beiden Argumenten zurück. Normalfall: Hängt neues Element vor Liste, gibt Ergebnis zurück.
`car`	Gibt erstes Element eines `cons`-Paares zurück. Normalfall: Erstes Element einer Liste
`cdr`	Gibt zweites Element eines `cons`-Paares zurück. Normalfall: Liste ohne das erste Element.
`caar`, `cadr`, `cddr` ...	Verschachtelungen von `car` und `cdr`
`append`	Hängt zwei (oder mehr) Listen zusammen

Scheme-Übersicht (4 - Listen 2)

`pair?`	Prüft, ob das Argument ein `cons`-Paar ist
`list?`	Prüft, ob das Argument eine Liste ist
`null?`	Prüft, ob das Argument die leere Liste ist
`list-ref`	Gibt das k-te Element einer Liste zurück
`list-tail`	Gibt Liste ohne die ersten k Elemente zurück
`member`	Sucht Objekt in Liste
`assoc`	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)

`port?`	Test, ob ein Objekt ein IO-Port ist.
`display`	Freundliche Ausgabe eines Objekts (optional: Port)
`newline`	Zeilenumbruch in der Ausgabe
`read`	Liest ein Scheme-Objekt
`write`	Schreibe ein Scheme-Objekt
`read-char`	Liest ein Zeichen
`write-char`	Schreibt ein Zeichen
`peek-char`	Gibt das nächste Zeichen oder *eof-object* zurück, ohne es zu lesen
`eof-object?`	Prüft, ob das Argument ein End-of-File repräsentiert
`open-input-port`	Öffne Datei zum Lesen
`open-output-port`	Öffne Datei zum Schreiben
`close-input-port`	Schließt Datei
`close-output-port`	Schließt Datei

Scheme-Übersicht (7 - Funktional und destruktiv)

`lambda`	Gibt eine neue Funktion zurück
`map`	Wendet Funktion auf alle Elemente einer Liste an, gibt Liste der Ergebnisse zurück. Bei mehrstelligen Funktionen entsprechend viele Argumentlisten!
`quote`	Gib einen Ausdruck unausgewertet zurück. Kurzform: ’
`eval`	Wertet einen Scheme-Ausdruck in der mitgegebenen Umgebung aus
`apply`	Ruft eine Funktion mit einer Argumentenliste auf, gibt das Ergebnis zurück
`set!`	Setzt eine Variable auf einen neuen Wert
`set-car!` (nicht Racket)	Setzt das `car` einer existierenden `cons`-Zelle
`set-cdr!` (nicht Racket)	Setzt das `cdr` einer existierenden `cons`-Zelle

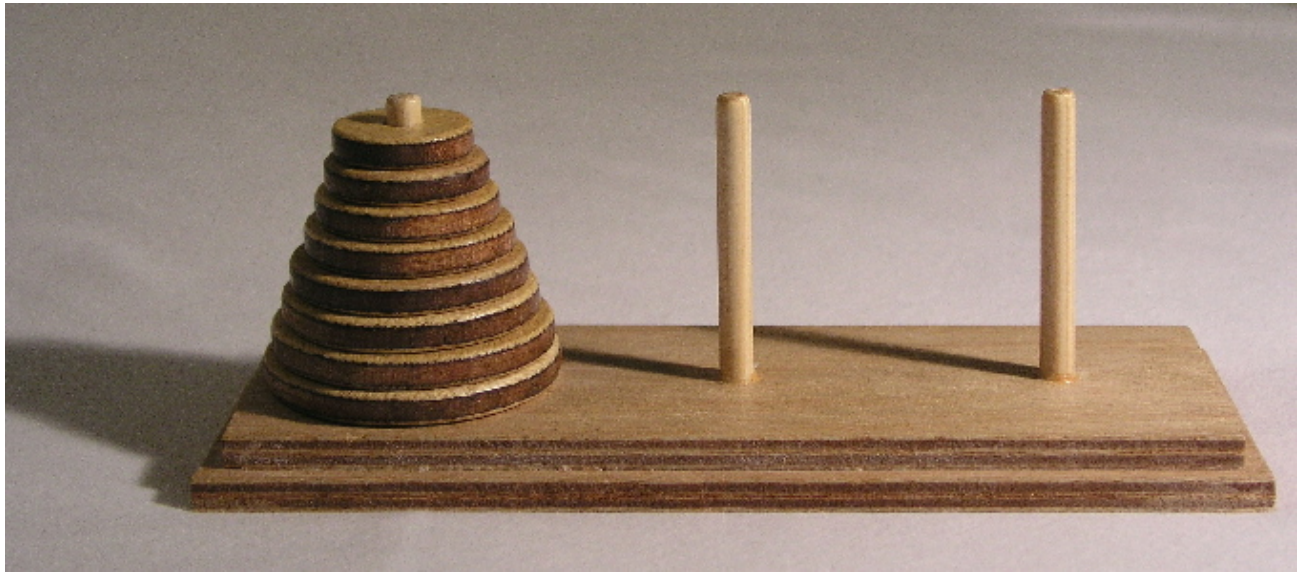
Scheme-Übersicht (8 -Typsystem)

- 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 `'()`

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, dass die *Türme von Hanoi* für beliebige Turmgrößen spielt
  - Was ist eine 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

```
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: ()
```

Einige Lösungen

Erinnerung: Konstruktion der natürlichen Zahlen (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
$\ldots$	$\ldots$	$\ldots$

- Wir definieren rekursive Rechenregeln rein syntaktisch:
  - Addition (a):
    - $a(X, 0) = X$
    - $a(X, s(Y)) = s(a(X, Y))$
  - Multiplikation (m):
    - $m(X, 0) = 0$
    - $m(X, s(Y)) = a(X, m(X, Y))$

Erinnerung: Konstruktion der natürlichen Zahlen (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: 2×2:

$$
\begin{aligned}
& \underline{m(s(s(0)), s(s(0)))} \\
= \quad & a(s(s(0)), \underline{m(s(s(0)), s(0))}) && (4) \text{ mit } X = s(s(0)), Y = s(0) \\
= \quad & a(s(s(0)), a(s(s(0)), \underline{m(s(s(0)), 0)})) && (3) \text{ mit } X = s(s(0)), Y = 0 \\
= \quad & a(s(s(0)), \underline{a(s(s(0)), 0)}) && (3) \text{ mit } X = s(s(0)) \\
= \quad & a(s(s(0)), s(s(0))) && (1) \text{ mit } X = s(s(0)) \\
= \quad & s(\underline{a(s(s(0)), s(0))}) && (2) \text{ mit } X = s(s(0)), Y = s(0) \\
= \quad & s(s(\underline{a(s(s(0)), 0)})) && (2) \text{ mit } X = s(s(0)), Y = 0 \\
= \quad & s(s(s(s(0)))) && (1) \text{ mit } X = s(s(0)), Y = 0
\end{aligned}
$$

Aufgabe: Erweiterung auf $\mathbb{Z}$

- Erweiterung auf negative Zahlen:
  - Idee: $p(X) = X - 1$, $n(X) = -X$, $v(X, Y) = X - Y$

```
n(0) = 0                    n(n(X)) = X
p(n(X)) = n(s(X))           n(p(X)) = s(n(X))
p(s(X)) = X                 s(p(X)) = X

a(X,0) = X                  a(X,s(Y)) = s(a(X,Y))
v(X,0)=X                    v(X,s(Y)) = p(v(X,Y))

a(X,n(Y)) = v(X,Y)          v(X,n(Y)) = a(X,Y)



m(X,0) = 0                  m(X,s(Y)) = a(X,m(X,Y))

m(X,n(Y)) = n(m(X,Y))       m(n(X),Y) = n(m(X,Y))
```

Lösung: Eigenschaften von Relationen (1)

Lösung zu *Übung: Eigenschaften von Relationen*

- Frage jeweils: linkstotal, rechtseindeutig, reflexiv, symmetrisch, transitiv
- $>\subseteq \mathbb{N}^2$
  - $>$ ist nicht linkstotal: Es exisitiert kein x mit $0 > x$
  - $>$ ist nicht rechtseindeutig: $4 > 3, 4 > 2$
  - $>$ ist nicht reflexiv, $(1,1) \notin >$
  - $>$ ist nicht symmetrisch, $2 > 1$, aber nicht $1 > 2$
  - $>$ ist transitiv, wenn $x > y$ und $y > z$, dann auch $x > z$
- $\leq\subseteq \mathbb{N}^2$
  - $\leq$ ist linkstotal: $x \leq x$ für alle $x \in \mathbb{N}$
  - $\leq$ ist nicht rechtseindeutig: $4 \leq 4, 4 \leq 5$
  - $\leq$ ist reflexiv, $(x,x) \in \leq$ für alle $x \in \mathbb{N}$
  - $\leq$ ist nicht symmetrisch, $1 \leq 2$, aber nicht $2 \leq 1$
  - $\leq$ ist transitiv, wenn $x \leq y$ und $y \leq z$, dann auch $x \leq z$

Lösung: Eigenschaften von Relationen (2)

- $=\subseteq A \times A$ (die Gleichheitsrelation auf einer beliebigen nichtleeren Menge A)
  - $=$ ist linkstotal: $x = x$ für alle $x \in A$
  - $=$ ist rechtseindeutig: $x = y$ und $x = z$ impliziert $y = z$
  - $=$ ist reflexiv: $x = x$ für alle $x \in A$
  - $=$ ist symmetrisch: $x = y$ impliziert $y = x$
  - $=$ ist transitiv, wenn $x = y$ und $y = z$, dann $x = z$
  - Damit ist $=$ eine Äquivalenzrelation
- Behauptung: Jede Äquivalenzrelation R über einer Menge M ist linkstotal
  - R ist reflexiv, also gilt $(x, x) \in R$ für alle x in M.
- Behauptung: Nicht jede Äquivalenzrelation ist rechtseindeutig
  - Betrachte $M = \{1, 2\}$ und $R = \{(1, 1), (1, 2), (2, 1), (2, 2)\}$
  - R ist reflexiv, symmetrisch, transitiv, also Äquivalenzrelation
  - Aber: $1R1$ und $1R2$. Also: R ist nicht rechtseindeutig.

Zurück (4)

Lösung: Hüllenbildung (1)

Ausgangsrelation:
$R = \{(a, b), (b, c), (c, d)\}$

	a	b	c	d
a	0	1	0	0
b	0	0	1	0
c	0	0	0	1
d	0	0	0	0

Reflexive Hülle $R \cup R^0$

	a	b	c	d
a	1	1	0	0
b	0	1	1	0
c	0	0	1	1
d	0	0	0	1

Symmetrische Hülle $R \cup R^{-1}$

	a	b	c	d
a	0	1	0	0
b	1	0	1	0
c	0	1	0	1
d	0	0	1	0

Transitive Hülle R^+

	a	b	c	d
a	0	1	1	1
b	0	0	1	1
c	0	0	0	1
d	0	0	0	0

Lösung: Hüllenbildung (2)

Reflexive transitive Hülle R^*

	a	b	c	d
a	1	1	1	1
b	0	1	1	1
c	0	0	1	1
d	0	0	0	1

Reflexive, symmetrische, transitive Hülle $(R \cup R^{-1})^*$

	a	b	c	d
a	1	1	1	1
b	1	1	1	1
c	1	1	1	1
d	1	1	1	1

Zurück

Lösung: Relationen für Fortgeschrittene (1)

Aufgabenstellung

- Betrachten Sie die Menge $M = \{a, b, c\}$.
  - Wie viele (binäre homogene) Relationen über M gibt es?
  - Definitionen:
    - Eine (n-stellige) Relation R über $M_1, M_2, \ldots M_n$ ist eine Teilmenge des karthesischen Produkts der Mengen, also $R \subseteq M_1 \times M_2 \times \ldots \times M_n$
    - R heißt homogen, falls $M_i = M_j$ für alle $i, j \in \{1, \ldots, n\}$.
    - 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.

Lösung: Relationen für Fortgeschrittene (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: $4^3 = 64$

Lösung: Relationen für Fortgeschrittene (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

Lösung: Relationen für Fortgeschrittene (4)

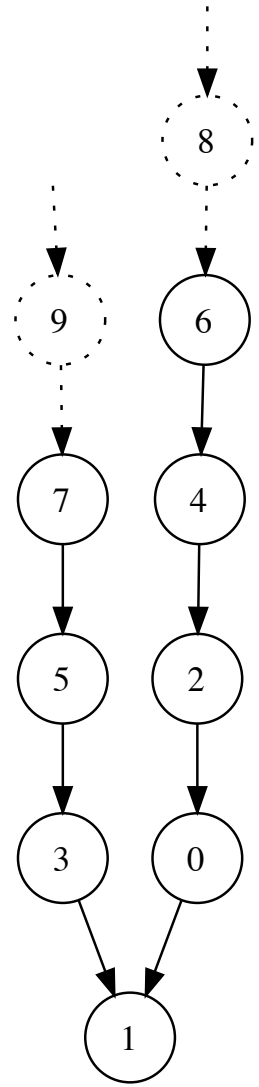
- Wie viele dieser Relationen sind
  - Transitiv?
    - Beobachtung: Wenn R transitiv ist, so gilt: Alles, wass ich in zwei Schritten erreichen kann, kann ich auch in einem Schritt erreichen! Also: $R^2 \subseteq R$
    - Kleines (Scheme-)Programm: 171 der 512 Relationen sind transitiv
  - Funktionen (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

Lösung: Relationen für Fortgeschrittene (5)

- Betrachten Sie folgende Relation über $\mathbb{N}$: xRy gdw. $x = y + 2$
  - Was ist die transitive Hülle von R?
    - xR^+y gdw. $x \mathit{mod} 2 = y \mathit{mod} 2$ und $x > y$
  - Was ist die reflexive, symmetrische, transitive Hülle von R?
    - $(x, y) \in (R \cup R^{-1})^*$ gdw. $x \mathit{mod} 2 = y \mathit{mod} 2$ (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'?
    - $R'^+ = R^+ \cup \{(x, 1) \mid x \in \mathbb{N} \backslash \{1\}\}$
    - ... aber $(R' \cup R'^{-1})^* = \mathbb{N} \times \mathbb{N}$



Relation R'

Lösung: Relationen für Fortgeschrittene (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.

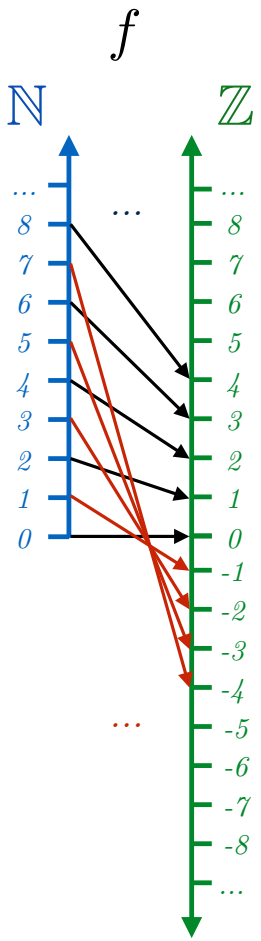
      q.e.d.

Zurück

Lösung: Kardinalität (1.1)

Lösung zu Übung: Kardinalität, als Beispiel mal sehr penibel.

- Behauptung: $\mathbb{Z}$ ist abzählbar.
- Beweis: Zu zeigen: Es gibt eine (totale) bijektive Abbildung von $\mathbb{N}$ nach $\mathbb{Z}$.
  - Wir definieren: $f : \mathbb{N} \to \mathbb{Z}$,
    $$f(x) = \begin{cases} \frac{x}{2} & \text{falls } x \text{ gerade} \\ -\frac{x+1}{2} & \text{falls } x \text{ ungerade} \end{cases}$$
  - Dann gilt: f ist surjektiv ("jedes Element wird getroffen"): Sei $y \in \mathbb{Z}$ beliebig.
    - Fall 1: $y \geq 0$. Dann gilt: $2y \in \mathbb{N}$ und gerade, und damit $f(2y) = \frac{2y}{2} = y$.
    - Fall 2: $y < 0$. Dann gilt: $-2y - 1 \in \mathbb{N}$ und ungerade, und damit $f(-2y-1) = -\frac{(-2y-1)+1}{2} = -\frac{-2y}{2} = --y = y$.
    - In beiden Fällen ist also $y \in f(\mathbb{N})$, damit ist f surjektiv.
- (Fortsetzung nächste Seite)



Lösung: Kardinalität (1.2)

- ...
  - Erinnerung: $f : \mathbb{N} \to \mathbb{Z}$, $f(x) = \begin{cases} \frac{x}{2} & \text{falls } x \text{ gerade} \\ -\frac{x+1}{2} & \text{falls } x \text{ ungerade} \end{cases}$
  - Außerdem gilt: f ist injektiv ("Kein Element wird von zwei verschiedenen Elementen getroffen"): Seien $x, z \in \mathbb{N}$ beliebig und gelte $f(x) = y = f(z)$ für ein $y \in \mathbb{Z}$. Zu zeigen ist: $x = z$.
    - Beweis per Widerspruch: Annahme: $x \neq z$. Ohne Beschränkung der Allgemeinheit gelte $x < z$, also $x + k = z$ für $k \in \mathbb{N}^+$.
    - Fall 1: x ist gerade, z ist ungerade. Dann folgt $f(x) \geq 0, f(z) < 0$, im Widerspruch zu $f(x) = f(z)$.
    - Fall 2: x und z sind gerade. Damit ist auch k gerade. Dann gilt: $f(x) = \frac{x}{2}$, $f(z) = f(x + k) = \frac{x+k}{2} = \frac{x}{2} + \frac{k}{2}$. Da $k > 0$ und gerade ist $\frac{k}{2} \geq 1$ und damit $\frac{x}{2} \neq \frac{x}{2} + \frac{k}{2}$, im Widerspruch zu $f(x) = f(z)$.
    - Fall 3: x und y sind ungerade. Analog zu Fall 2.
    - Damit gilt also: f ist injektiv.
  - Also: f ist injektiv und surjektiv, damit bijektiv.
- ...und damit gilt die Behauptung. q.e.d.

Lösung: Kardinalität (2.1)

Behauptung: Für endliche Mengen M gilt: $|2^M| = 2^{|M|}$

Beweis: Per Induktion nach $|M|$.

Induktionsanfang: $|M| = 0$. Damit gilt: $M = \emptyset$ und $2^M = \{\emptyset\}$, damit $|2^M| = 1$. Also: $|2^M| = 1 = 2^0 = 2^{|M|}$. Die Behauptung gilt für Mengen mit der Mächtigkeit 0.

Induktionsvoraussetzung: Die Behauptung gelte für alle Mengen M mit Mächtigkeit n.

Lösung: Kardinalität (2.2)

Induktionsschritt: Wenn die Behauptung für alle Mengen M mit Mächtigkeit n gilt, dann auch für Mengen M' mit $|M'| = n + 1$.

- Sei also M' eine beliebige Menge mit $|M'| = n + 1$, sei $a \in M'$ und $M = M' \backslash \{a\}$. Dann gilt: $|M| = n$ und $M' = M \cup \{a\}$. Wir zeigen: $|2^{M'}| = 2 \cdot |2^M|$. Es gilt:
  1. Jede Teilmenge von $|M|$ ist auch Teilmenge von $|M'|$
  2. Sei $m \subseteq M$. Dann ist $m \cup \{a\} \subseteq M'$
  3. Sei $T_a = \{m \cup \{a\} \mid m \subseteq M\}$. Aus jeder Teilmenge von M entsteht genau eine neue Menge in T_a. Also ist $|T_a| = |2^M|$.
  4. Auch sind $|T_a|$ und 2^M disjunkt (d.h. sie haben keine gemeinsamen Elemente) - alle Elemente in T_a enthalten a, und kein Element in 2^M enthält a.
  5. Weiterhin gilt: $2^{M'} = 2^M \cup T_a$.
  6. Da die beiden Mengen disjunkt sind, folgt $|2^{M'}| = |2^M| + |T_a|$, und damit (Schritt 3)
     $|2^{M'}| = |2^M| + |2^M| = 2 \cdot |2^M| =_{IA} 2 \cdot 2^n = 2^{n+1} = 2^{|M'|}$
- Damit gilt der Induktionsschritt und also die Behauptung. q.e.d.

Listenumkehr in Scheme

- Die leere Liste bleibt die leere Liste
- Ansonsten:
  - Drehe Liste ohne erstes Element rekursiv um
  - Hänge erstes Element *hinten* an

```
(define (revert lst)
  (if (null? lst)
      lst
      (append (revert (cdr lst))
              (list (car lst)))
      )))
```

- Trick: `append` hängt hinten an, braucht aber `(list x)`

Bonus: Split/Mix

```
(define (split l)
  (if (null? l)
      (list '() '())
      (if (null? (cdr l))
          (list l '())
          (list (cons (car l)
                      (car (split (cdr (cdr l)))))
                (cons (car (cdr l))
                      (car (cdr (split (cdr (cdr l))))))))))

(define (mix l1 l2)
  (if (null? l1)
      l2
      (if (null? l2)
          l1
          (cons (car l1)
                (cons (car l2)
                      (mix (cdr l1) (cdr l2)))))))
```

Mergesort (1)

Eingabe: Eine Liste `lst`

Ausgabe: Liste von zwei Listen `(r1 r2)`

```
(define (split l)
  (cond ((null? l)              ; Leeres l -> ('() '())
         (list '() '()))
        ((null? (cdr l))        ; Nur ein Element: ('() l)
         (list '() l))
        (else                   ; Sonst
         (let* ((res (split (cddr l)))
                (l1 (car res))      ;; Erste Liste des Teilergeb
                (l2 (cadr res))     ;; Zweite
                )
           (list (cons (car l) l1)
                 (cons (cadr l) l2))))))
```

Mergesort (2)

```
(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))))))

(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))))
```

Zurück

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.
  - $A \vee B \vee C$
- 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

Es gibt kein Modell der Aussagen

$\Longrightarrow$

Versicherungsbetrug durch den Ladenbesitzer!

Zurück

Lösung: 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.
- Aufgabe: Zeigen Sie:
  - $\{\rightarrow, \neg\}$ ist eine Basis

$\{\rightarrow, \neg\}$ ist eine Basis der Aussagenlogik (1)

Zu zeigen: Für jedes $F \in For0_\Sigma$ existiert ein $G \in For0_\Sigma$ 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. $F = \bot$: 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)

Induktionsvorraussetzung: Die Behauptung gelte für beliebige $A, B \in For0_\Sigma$ (also: es existieren$A', B' \in For0_\Sigma$ 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 $\rightarrow, \neg$.
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.

Zurück

Lösung: Logische Umformungen

- Ziel: $\vdash_{LU} (A \to (B \to A))$
- Ableitung:

$$
\begin{array}{lll}
\top & \vdash_{LU} & \neg B \vee \top \quad (19) \\
& \vdash_{LU} & \neg B \vee (A \vee \neg A) \quad (16) \\
& \vdash_{LU} & \neg B \vee (\neg A \vee A) \quad (2) \\
& \vdash_{LU} & (\neg B \vee \neg A) \vee A \quad (4) \\
& \vdash_{LU} & (\neg A \vee \neg B) \vee A \quad (2) \\
& \vdash_{LU} & \neg A \vee (\neg B \vee A) \quad (4) \\
& \vdash_{LU} & A \to (\neg B \vee A) \quad (9) \\
& \vdash_{LU} & A \to (B \to A) \quad (9)
\end{array}
$$

- Wie kommt man drauf?
- Ich gehe anders herum vor:
  - $(A \to (B \to A))$
  - $\equiv \neg A \vee (\neg B \vee A)$ ($\to$ ersetzen)
  - $\equiv (\neg A \vee A) \vee \neg B$ (umklammern und sortieren)
  - $\equiv \top \vee \neg B$
  - $\equiv \top$
- Dann vorwärts aufschreiben und ausgelassene Zwischenschritte einfügen

Zurück

Lösung: KNF Transformation

- Gesucht: KNF für
  $(A \vee B \vee C) \wedge (A \rightarrow ((B \wedge \neg C) \vee (\neg B \wedge C))) \wedge (\neg B \rightarrow \neg C) \wedge \neg(B \wedge C \wedge \neg A) \wedge (\neg C \rightarrow \neg B)$
- Wir können die Teile der Konjunktion einzeln in KNF umwandeln:
  1. $(A \vee B \vee C)$ ist bereits in KNF
  2. $(A \rightarrow ((B \wedge \neg C) \vee (\neg B \wedge C)))$
     $\equiv \neg A \vee ((B \wedge \neg C) \vee (\neg B \wedge C)))$
     $\equiv \neg A \vee ((B \vee (\neg B \wedge C)) \wedge (\neg C \vee (\neg B \wedge C)))$
     $\equiv \neg A \vee ((B \vee \neg B) \wedge (B \vee C)) \wedge ((\neg C \vee \neg B) \wedge (\neg C \vee C))$
     $\equiv \neg A \vee ((B \vee C) \wedge (\neg C \vee \neg B))$
     $\equiv (\neg A \vee B \vee C) \wedge (\neg A \vee \neg C \vee \neg B)$
  3. $(\neg B \rightarrow \neg C) \equiv (B \vee \neg C)$
  4. $\neg(B \wedge C \wedge \neg A) \equiv (\neg B \vee \neg C \vee A)$
  5. $(\neg C \rightarrow \neg B) \equiv (C \vee \neg B)$
- Die Gesamtlösung ist dann dann die Ver-undung der 6 Klauseln aus 1-5.

Zurück

Lösung: Formeln der Prädikatenlogik 1. Stufe

- „Alle, die in Stuttgart studieren, sind schlau“

$$\forall X(\text{studiertin}(X, \text{stuttgart}) \rightarrow \text{schlau})X))$$

  Eine Möglichkeit mit:
  - Prädikatssymbol studiertin /2
  - Prädikatssymbol schlau /1
  - Funktionssymbol stuttgart /0 (damit *Konstante*)
- „Es gibt jemand, der in Mannheim studiert und schlau ist“

$$\exists X(\text{studiertin}(X, \text{mannheim}) \wedge \text{schlau}(X))$$

- „Die Summe zweier Primzahlen ist eine Primzahl“

$$\forall X, Y((\text{prim}(X) \wedge \text{prim}(Y)) \rightarrow \text{prim}(\text{sum}(X, Y)))$$

Arbeitsblatt: Auswertung von Formeln (1)

$F = \forall X(\exists Y(gt(X, plus(Y, 1))))$

- $U = \{T, F\}$
- $I(gt) = \{(T, F)\}$
- $I(plus) = \{((F, F), F), ((F, T), T), ((T, F), T), ((T, T), T)\}$
- $I(1) = T$.

Arbeitsblatt: Auswertung von Formeln (1)

$F = \forall X(\exists Y(gt(X, plus(Y, 1))))$

- $U = \{T, F\}$
- $I(gt) = \{(T, F)\}$
- $I(plus) = \{((F, F), F), ((F, T), T), ((T, F), T), ((T, T), T)\}$
- $I(1) = T$.

Arbeitsblatt: Auswertung von Formeln (2)

$F = \forall X(\exists Y(gt(X, plus(Y, 1))))$

- $U = \mathbb{N}$
- $I(gt) = <$
- $I(plus) = +$
- $I(1) = 1$.

Zurück

Lösung: Primzahlen

- Primzahlen:

$$\forall x(\text{prim}(x) \leftrightarrow$$

$$\neg = (x, 1) \wedge \neg \exists y \exists z\ (= (\text{mult}(y, z), x) \wedge \neg = (y, 1) \wedge \neg = (z, 1))$$

  Eine (natürliche) Zahl ist prim, wenn sie weder 1 ist, noch das Produkt von zwei Zahlen ist, die beide ungleich 1 sind.

- $$\forall x(\text{prim}(x) \rightarrow \exists y(\text{prim}(y) \wedge > (y, x)))$$

- Betrachte $U = \mathbb{N}$, $I(\mathit{prim}(x)) = 1$ gdw $x \in \mathbb{P}$, $I(1) = 1$, $I(\text{mult}) = x, y \mapsto x \cdot y$, $I(>) = >$, $I(=) = =$
- Betrachte Interpretation I' identisch zu I bis auf $I'(>) = <$ ($>$ wird als die kleiner-Relation interpretiert).

Zurück

Lösung: NNF (Prädikatenlogik)

- $\neg\exists x\ (p(x) \to \forall z\ q(z))$
  $\equiv \neg\exists x\ (\neg p(x) \vee \forall z\ q(z))$ (9)
  $\equiv \forall x\ \neg(\neg p(x) \vee \forall z\ q(z))$ (24)
  $\equiv \forall x\ (p(x) \wedge \neg\forall z\ q(z))$ (12,7)
  $\equiv \forall x\ (p(x) \wedge \exists z\ \neg q(z))$ (23) **(NNF)**
- $\forall y\ \forall z\ (p(z) \to \neg\forall x\ (q(x,z) \wedge \exists z\ r(y,z)))$
  $\equiv \forall y\ \forall z\ (\neg p(z) \vee \neg\forall x\ (q(x,z) \wedge \exists z\ r(y,z)))$ (9)
  $\equiv \forall y\ \forall z\ (\neg p(z) \vee \exists x\ \neg(q(x,z) \wedge \exists z\ r(y,z)))$ (23)
  $\equiv \forall y\ \forall z\ (\neg p(z) \vee \exists x\ (\neg q(x,z) \vee \neg\exists z\ r(y,z)))$ (12)
  $\equiv \forall y\ \forall z\ (\neg p(z) \vee \exists x\ (\neg q(x,z) \vee \forall z\ \neg r(y,z)))$ (24) **(NNF)**

Zurück

Lösung: PNF

- $\neg\exists x\ (p(x) \to \forall z\ q(z))$
  $\equiv \forall x\ (p(x) \wedge \exists z\ \neg q(z))$ **(NNF)**
  $\equiv \forall x\ \exists z\ (p(x) \wedge \neg q(z))$ (26) **(PNF)**
  $\equiv \forall x_0\ \exists x_1\ (p(x_0) \wedge \neg q(x_1))$ (26) **(Alternative PNF)**
- $\forall y\ \forall z\ (p(z) \to \neg\forall x\ (q(x,z) \wedge \exists z\ r(y,z)))$
  $\equiv \forall y\ \forall z\ (\neg p(z) \vee \exists x\ (\neg q(x,z) \vee \forall z\ \neg r(y,z)))$ **(NNF)**
  $\equiv \forall x_0\ \forall x_1\ (\neg p(x_1) \vee \exists x_2\ (\neg q(x_2,x_1) \vee \forall x_3\ \neg r(x_0,x_3)))$ (27,28)
  $\equiv \forall x_0\ \forall x_1\ \exists x_2\ \forall x_3\ (\neg p(x_1) \vee (\neg q(x_2,x_1) \vee \neg r(x_0,x_3)))$ (25,26) **(PNF)**

Zurück

Lösung: SNF

- $\neg\exists x\ (p(x) \to \forall z\ q(z))$
  $\equiv \forall x_0\ \exists x_1\ (p(x_0) \wedge \neg q(x_1))$ **(PNF)**
  SNF: $\forall x_0\ (p(x_0) \wedge \neg q(\mathsf{sk}_0(x_0)))$ **(SNF)**
- $\forall y\ \forall z\ (p(z) \to \neg\forall x\ (q(x,z) \wedge \exists z\ r(y,z)))$
  $\equiv \forall x_0\ \forall x_1\ \exists x_2\ \forall x_3\ (\neg p(x_1) \vee (\neg q(x_2,x_1) \vee \neg r(x_0,x_3)))$ **(PNF)**
  SNF: $\forall x_0\ \forall x_1\ \forall x_3\ (\neg p(x_1) \vee (\neg q(\mathsf{sk}_2(x_0,x_1),x_1) \vee \neg r(x_0,x_3)))$
  (SNF)

Anmerkung: Die Namen der Skolem-Funktionen und die Namen der Variablen sind beliebig. Bei der Wahl des Namens einer Skolem-Funktion muss man nur darauf achten, dass dieser bisher in der Formel nicht vorkommt.

Zurück

Lösung: Unifikation (1)

- Unikation von $f(X, a)$ und $f(g(a), Y)$:

Gleichungen	σ	Regel
$\{f(X, a) = f(g(a), Y)\}$	$\{\}$	Zerlegen
$\{X = g(a), a = Y\}$	$\{\}$	Orientieren
$\{X = g(a), Y = a\}$	$\{\}$	Binden
$\{X = g(a)\}$	$\{Y \leftarrow a\}$	Binden
$\{\}$	$\{Y \leftarrow a, X \leftarrow g(a)\}$	

Ergebnis: $mgu(f(X, a), f(g(a), Y)) = \{Y \leftarrow a, X \leftarrow g(a)\}$

Lösung: Unifikation (2)

- Unikation von $f(X, a)$ und $f(g(a), X)$:

Gleichungen	σ	Regel
$\{f(X, a) = f(g(a), X)\}$	$\{\}$	Zerlegen
$\{X = g(a), a = X\}$	$\{\}$	Orientieren
$\{X = g(a), X = a\}$	$\{\}$	Binden
$\{a = g(a)\}$	$\{X \leftarrow a\}$	Konflikt
FAIL		

Ergebnis: $f(X, a)$ und $f(g(a), Y))$ sind nicht unifizierbar.

Lösung: Unifikation (3)

- Unikation von $f(X, f(Y, X))$ und $f(g(g(a)), f(a, g(Z)))$:

Gleichungen	σ	Regel
$\{f(X, f(Y, X)) = f(g(g(a)), f(a, g(Z)))\}$	$\{\}$	Zerlegen
$\{X = g(g(a)), f(Y, X) = f(a, g(Z))\}$	$\{\}$	Binden
$\{f(Y, g(g(a))) = f(a, g(Z))\}$	$\{X \leftarrow g(g(a))\}$	Zerlegen
$\{Y = a, g(g(a)) = g(Z)\}$	$\{X \leftarrow g(g(a))\}$	Binden
$\{g(g(a)) = g(Z)\}$	$\{X \leftarrow g(g(a)), Y \leftarrow a\}$	Zerlegen
$\{g(a) = Z\}$	$\{X \leftarrow g(g(a)), Y \leftarrow a\}$	Orientieren
$\{Z = g(a)\}$	$\{X \leftarrow g(g(a)), Y \leftarrow a\}$	Binden
$\{\}$	$\{X \leftarrow g(g(a)), Y \leftarrow a,$ $Z \leftarrow g(a)\}$	

Ergebnis: $mgu(f(X, f(Y, X)), f(g(g(a)), f(a, g(Z)))) = \{X \leftarrow g(g(a)), Y \leftarrow a, Z \leftarrow g(a)\}$

Zurück

Lösung: Tierische Resolution

- Initiale Klauselmenge:
  1. $\neg h(X) \vee l(X)$
  2. $\neg k(X) \vee \neg hat(Y, X) \vee \neg hat(Y, Z) \vee \neg m(Z)$
  3. $\neg e(X) \vee \neg hat(X, Y) \vee \neg l(Y)$
  4. $hat(john, tier)$
  5. $h(tier) \vee k(tier)$
  6. $e(john)$
  7. $hat(john, maus)$
  8. $m(maus)$

- Beweis:
  9. $\neg k(X) \vee \neg hat(john, X) \vee \neg m(maus)$ (2,7)
  10. $\neg k(X) \vee \neg hat(john, X)$ (8,9)
  11. $\neg k(tier)$ (10, 4)
  12. $\neg e(john) \vee \neg l(tier)$ (3,4)
  13. $\neg l(tier)$ (12, 6)
  14. $\neg h(tier)$ (1,13)
  15. $k(tier)$ (5,14)
  16. $\square$ (15,11)

Zurück

Einzelvorlesungen

Ziele Vorlesung 1

- Gegenseitiges Kennenlernen
- Praktische Informationen
- Übersicht und Motivation

Vorstellung

- 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
    - Grundlagen der Informatik

Forschung: Logik & Deduktion

Kennenlernen

- Ihre Erfahrungen
  - Informatik allgemein?
  - Programmieren? Sprachen?
- Ihre Erwartungen?
  - ...
- Feedbackrunde am Ende der Vorlesung

Praktische Informationen TINF21C

- Regelmäßige Vorlesungszeiten
  - Mi., 14:45–16:45
  - Do., 15:00–17:00 (außer 9.1.)
  - Ersatztermin 12.12.2024, 12:15-14:15
  - 11 Wochen Vorlesung+Klausurwoche
  - Weihnachtspause: 23.12.–7.1.
  - Im Zweifelsfall gilt RAPLA!
- Klausur
  - 6.3.2025, 11:00
  - 90 Minuten, Open-Book
- Webseite zur Vorlesung

  `http://wwwlehre.dhbw-stuttgart.de/~sschulz/lgli2024.html`

Praktische Informationen TINF21IN

- Regelmäßige Vorlesungszeiten
  - Di., 13:00–15:00
  - Mi., 10:30–12:30 (eventuell müssen wir gegen Ende einige Vorlesungen verlegen)
  - 11 Wochen Vorlesung+Klausurwoche
  - Weihnachtspause: 23.12.–7.1.
  - Im Zweifelsfall gilt RAPLA!
- Klausur
  - 6.3.2025, 11:00
  - 90 Minuten, Open-Book
- Webseite zur Vorlesung

  `http://wwwlehre.dhbw-stuttgart.de/~sschulz/lgli2024.html`

Rechnerausstattung

- Für die praktischen Übungen benötigen Sie einen Rechner, auf dem Sie Scheme-Programme entwickeln und ausführen können
- Empfohlene Implementierung: Racket (aktuelle Version 8.11)
  `https://racket-lang.org/`
  - Solide, performante Implementierung
  - Integrierte IDE (DrRacket)
  - Unterstützt alle wichtigen Plattformen
    - OS-X
    - Linux
    - (Windows)
- Gute Libraries

Zur Vorlesung

Hausaufgabe

- Installieren Sie auf Ihrem Laptop eine aktuelle Version von *Racket*
  - `http://racket-lang.org/`
  - `http://download.racket-lang.org/`
- Bringen Sie (mindestens) “Hello World” zur Ausführung
  - Codebeispiele für Racket/Scheme gibt es unter

    `http://wwwlehre.dhbw-stuttgart.de/~sschulz/lgli2024.html`

Zusammenfassung

- Gegenseitiges Kennenlernen
- Praktische Informationen
- Übersicht und Motivation

Image credit, when not otherwise specified: Wikipedia, OpenClipart

Feedback

- Was war der beste Teil der heutigen Vorlesung?
- Was kann verbessert werden?
  - Optional: Wie?

Ziele Vorlesung 2

- Wiederholung/Auffrischung
- Mengenlehre - Grundbegriffe
- Formale Konstruktion der natürlichen Zahlen
- Mengenlehre - weiterführend
  - Venn-Diagramme
  - Mengenoperationen

Wiederholung

- Einführung
- MIU als formales System
  - Ableitungen im System
  - Argumentation über das System
- Logik
  - Formalisierung von rationalem Denken
  - Grundlage von/Anwendung auf/Anwendung in Informatik
  - Beispiel Flugsicherung

Zur Vorlesung

Hausaufgabe: Konstruktion der negativen Zahlen

- Erweitern Sie die rekursiven Definitionen von a 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)

Zusammenfassung

- Wiederholung/Auffrischung
- Mengenlehre
  - Grundbegriffe
  - Definition/Teilmengen
  - Beispiele für Mengen
- Formale Konstruktion der natürlichen Zahlen
  - $0 \simeq \{\}$
  - $1 \simeq s(0) \simeq \{\{\}\}$
  - $2 \simeq s(s(0)) \simeq \{\{\{\}\}\}$
- Mengenlehre
  - Venn-Diagramme
  - Mengenoperationen

Feedback

- Was war der beste Teil der heutigen Vorlesung?
- Was kann verbessert werden?
  - Optional: Wie?

Ziele Vorlesung 3

- Wiederholung/Auffrischung
- Mengenlehre:
  - Tupel/Kartesische Produkte
  - Potenzmengen
  - Mengenalgebra: Rechnen mit Mengen
- Relationen
  - Definition
  - Eigenschaften

Wiederholung

- Mengenlehre
  - Grundbegriffe
  - Teilmengen
  - Venn-Diagramme
  - Mengenoperationen
- Termalgebra/Konstruktion der natürlichen Zahlen
  - $2 \simeq s(s(0)) \simeq \{\{\{\}\}\}$
  - Addition/Multiplikation
    - Rekursive Definitionen: Basisfall (0), rekursiver Fall ($s(Y)$)
  - Hausaufgabe: Erweiterung auf negative Zahlen

Zur Vorlesung

Hausaufgabe

Bearbeiten Sie die *Übung: Eigenschaften von Relationen* zu Ende.

Zusammenfassung

- Mengenlehre:
  - Kartesische Produkte/Tupel
  - Potenzmengen - Menge aller Teilmengen
  - Mengenalgebra: Rechnen mit Mengen
- Relationen
  - Definition: Menge von Tupeln
  - Beispiele
    - Praktisch
    - Mathematisch
  - Eigenschaften
    - homogen, binär,
    - linsktotal, rechtseindeutig
    - Reflexiv, symmetrisch, transitiv – Äquivalenzrelationen

Feedback

- Was war der beste Teil der heutigen Vorlesung?
- Was kann verbessert werden?
  - Optional: Wie?

Ziele Vorlesung 4

- Wiederholung/Auffrischung
- Darstellung von (endlichen) Relationen
- Relationenalgebra
  - Verknüpfungen von Relationen
  - Erweiterungen von Relationen

Wiederholung

- Mengenlehre:
  - Kartesische Produkte & Tupel
  - Potenzmengen - Menge aller Teilmengen
  - Mengenalgebra: Rechnen mit Mengen
- Relationen
  - Definition: Menge von Tupeln
  - Beispiele
  - Eigenschaften
    - homogen, binär,
    - linsktotal, rechtseindeutig
    - Reflexiv, symmetrisch, transitiv – Äquivalenzrelationen
- Hausaufgabe: *Eigenschaften von Relationen*

Zur Vorlesung

Zusammenfassung

- Darstellung von (endlichen) Relationen
  - Mengendarstellung
  - Graphdarstellung
  - Tabellendarstellung
- Relationenalgebra
  - Inverse Relation
  - Identitätsrelation
  - Verknüpfung von Relationen/Potenzierung
  - Hüllenbildung (reflexiv, symmetrisch, transitiv)

Feedback

- Was war der beste Teil der heutigen Vorlesung?
- Was kann verbessert werden?
  - Optional: Wie?

Ziele Vorlesung 5

- Wiederholung/Auffrischung
- Funktionen
  - Total/partiell/Bild/Urbild
  - Injektiv, surjektiv, bijektiv
- Kardinalität

Wiederholung

- Darstellung von (endlichen) Relationen
  - Mengendarstellung
  - Graphdarstellung
  - Tabellendarstellung
- Relationenalgebra
  - Inverse Relation/Identität
  - Verknüpfung von Relationen/Potenzierung
  - Hüllenbildung (reflexiv, symmetrisch, transitiv)

Zur Vorlesung

Hausaufgabe

- Bearbeiten Sie die *Übung: Relationen für Fortgeschrittene* zu Ende.
- Bearbeiten Sie die *Übung: Kardinalität* zu Ende.

Zusammenfassung

- Funktionen
  - Total/partiell/Bild/Urbild
  - Injektiv, surjektiv, bijektiv
- Kardinalität

Feedback

- Was war der beste Teil der heutigen Vorlesung?
- Was kann verbessert werden?
  - Optional: Wie?

Ziele Vorlesung 6

- Wiederholung/Auffrischung
- Besprechung der Hausaufgaben zu Relationen/Kardinalität
- Vorstellung Scheme
- Syntax von Scheme
- Arbeiten mit Scheme/DrRacket
  - "Hello World" - Starten und Ausführen von Programmen
  - Fakultätsfunktion
  - Interaktives Arbeiten
- Semantik von Scheme
- *Special Forms*
- Datentypen und Basisfunktionen

Wiederholung

- Funktionen
- Kardinalität

Lösungen zu Relationen/Kardinalität

- Relationen für Fortgeschrittene
- Kardinalität

Zur Vorlesung

Zusammenfassung

- Syntax von Scheme - *s-expressions*
- Einführung in die Semantik von Scheme
- *Special Forms*
  - `if`
  - `quote`
  - `define`
- Datentypen und Basisfunktionen
  - Typ hängt am Wert (nicht an der Variable)
  - Boolsche Werte, Zahlen, Strings, Listen, Funktionen,...
  - `=`, `equal?`, `+`, `-`, `*`, `/`

Feedback

- Was war der beste Teil der heutigen Vorlesung?
- Was kann verbessert werden?
  - Optional: Wie?

Ziele Vorlesung 7

- Wiederholung/Auffrischung
- Listenverarbeitung
- Funktionsaufrufe/Auswertung
- Rekursion als allgemeiner Lösungsansatz
- Programmierübungen
- Speichermodell und Umgebungen
- *Special forms*
  - Sequenzen: `begin`
  - Bedingungen: `cond`
  - Logische Verknüpfungen

Wiederholung

- Scheme-Programme sind Ansammlungen von *s-expressions*
  - Atome
  - Listen
- Berechnung durch Ausrechnen
  - Atome haben Wert (oder auch nicht $\rightarrow$ Fehler)
  - Listenelemente werden erst (rekursiv) ausgerechnet
  - Erstes Element einer Liste wird als Funktion auf den Rest angewendet
- *Special Forms*
  - `if`, `define`
- Datentypen hängen am Objekt
  - Polymorphismus ist trivial, Variablen sind nicht getypt

Zur Vorlesung

Hausaufgabe

- Bearbeiten Sie die *Übung: Revert* zu Ende.
- Bearbeiten Sie die *Übung: Mengenlehre* zu Ende.

Zusammenfassung

- Listenverarbeitung
  - `car`/`cdr` (auch Grundlage für Rekursion)
  - `null?`, `cons`, `append`, `list`
- Funktionsaufrufe/Auswertung
  - Formale und konkrete Parameter
- Rekursion als allgemeiner Lösungsansatz
- Übungen: Listenvervielfachung, Mengenlehre
- Speichermodell und Umgebungen
- *Special forms*
  - Sequenzen: `begin`
  - Bedingungen: `cond`
  - Logische Verknüpfungen

Feedback

- Was war der beste Teil der heutigen Vorlesung?
- Was kann verbessert werden?
  - Optional: Wie?

Ziele Vorlesung 8

- Wiederholung/Auffrischung
- Temporäre Variablen: `let` und `let*`
- Naives Sortieren
  - Einfaches Sortieren durch Einfügen
  - Sortieren mit Ordnungsfunktion

Wiederholung

- Listenverarbeitung
  - `car`/`cdr` (auch Grundlage für Rekursion)
  - `null?`, `cons`, `append`, `list`
- Funktionsaufrufe/Auswertung
- Rekursion als allgemeiner Lösungsansatz
  - Fallunterscheidung, Unterprobleme, Kombinieren
- Speichermodell und Umgebungen
  - Variablen, Speicher, Werte
- *Special forms*
  - Sequenzen: `begin`
  - Bedingungen: `cond`
  - `and`/`or`/`not`

Zur Vorlesung

Hausaufgabe

- Bearbeiten Sie die *Übung: Alter Wein in Neuen Schläuchen* zu Ende.
- Bearbeiten Sie die *Übung: Generisches Sortieren* zu Ende.

Zusammenfassung

- Temporäre Variablen: `let` und `let*`
- Sortieren durch Einfügen
  - Einfügen eines Elements
  - Alle Elemente sortiert einfügen
- Funktionales Sortieren: Ordnung als Parameter

Feedback

- Was war der beste Teil der heutigen Vorlesung?
- Was kann verbessert werden?
  - Optional: Wie?

Schöne Pause!

Ziele Vorlesung 9

- Wiederholung/Auffrischung
- Effizienter sortieren: Mergesort
  - Naiver Split (aber rekursiv)
  - Intelligenter Merge
- Input/Output in Scheme
- Mehr über Listen
  - Implementierung - `cons`-Paare
  - Weitere Listenfunktionen
  - (Association lists)
- Größere Übung: n-Queens

Wiederholung

- Temporäre Variablen: `let` und `let*`
- Sortieren durch Einfügen
- Funktionales Sortieren: Ordnung als Parameter

Zur Vorlesung

Hausaufgabe

Bearbeiten Sie die *Übung: Höfliche Damen* zu Ende.

Zusammenfassung

- Mergesort
  - Naiver Split (aber rekursiv)
  - Intelligenter Merge
- Input und Output
  - Ports
    - `current-input-port`, `current-output-port`
  - `read/write`
    - Lesen/schreiben beliebiger (!) Scheme-Objekte!
  - High/Human-level und low-level I/O
    - `read-char`, `display`, ...
- Mehr über Listen
  - Implementierung - `cons`-Paare
  - Weitere Listenfunktionen
  - Association lists
- *n*-Queens

Feedback

- Was war der beste Teil der heutigen Vorlesung?
- Was kann verbessert werden?
  - Optional: Wie?

Ziele Vorlesung 10

- Wiederholung/Auffrischung
- Abschluss Scheme
  - Funktionale Features von Scheme
  - (Destruktive Features von Scheme)
  - Reste und Lücken
  - Große Übung: Mastermind

Wiederholung

- Mergesort
- Input/Output
  - Ports als Abstraktion von Dateien/Terminals/...
  - `read`/`write` als universelle Serialisierung
- Cons-Paare und Listen
  - Listen-Struktur (`car`/`cdr`)
  - Listen-Funktionen
  - (Assoc-Listen)
- Damenproblem

Zur Vorlesung

Hausaufgabe

Bearbeiten Sie die *Übung: Mastermind* zu Ende.

Zusammenfassung

- Abschluss Scheme
  - Funktionale Features von Scheme
    - `lambda`, `map`, `eval`, `apply`
  - Reste und Lücken
    - Typen
    - Symbole
    - ...

Feedback

- Was war der beste Teil der heutigen Vorlesung?
- Was kann verbessert werden?
  - Optional: Wie?

Ziele Vorlesung 11

- Wiederholung/Auffrischung
- Besprechung Hausaufgaben
  - (Höfliche Damen)
  - Mastermind
- Einführung Aussagenlogik
  - Was ist eine Logik?
  - Syntax der Aussagenlogik
  - Einführung in die Semantik der Aussagenlogik

Wiederholung

- Funktionales Scheme
  - `lambda`, `map`, `apply`, `eval`
- (Destruktives Scheme)
  - (`set!` und Varianten)
- Abschluß

Zur Vorlesung

Zusammenfassung

- Was ist eine Logik?
- Syntax der Aussagenlogik
  - Atomare Aussagen
  - Verknüpfungen
  - Weniger Klammern mit Assoziativität und Präzedenz
- Semantik der Aussagenlogik
  - Interpretationen
  - Evaluierungsfunktion

Feedback

- Was war der beste Teil der heutigen Vorlesung?
- Was kann verbessert werden?
  - Optional: Wie?

Ziele Vorlesung 12

- Wiederholung/Auffrischung
- Semantik der Aussagenlogik
  - Modelle
  - Modellmengen
  - Logische Folgerung
- Wahrheitstafelmethode

Wiederholung

- Was ist eine Logik?
  - Syntax, Semantik, Kalkül
- Syntax der Aussagenlogik
  - Atomare Aussagen
  - $\top, \bot$
  - Verknüpfungen $(\neg, \wedge, \vee, \rightarrow, \leftrightarrow)$
  - Weniger Klammern mit Assoziativität und Präzedenz
- Semantik
  - Interpretation legt Werte für Atome fest
  - Fortgesetzt zu Evaluierung von Formeln

Zur Vorlesung

Hausaufgabe: 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?“

Zusammenfassung

- Semantik der Aussagenlogik
  - Interpretationen ("Variablenbelegungen")
  - Modelle (Interpretationen, die F wahr machen)
- Modellmengen
  - Logik und Mengenlehre
- Logische Folgerung
  - $M \models F$ gdw. $\mathrm{Mod}(M) \subseteq \mathrm{Mod}(F)$
- Wahrheitstafelmethode

Feedback

- Was war der beste Teil der heutigen Vorlesung?
- Was kann verbessert werden?
  - Optional: Wie?

Ziele Vorlesung 13

- Wiederholung/Auffrischung
- Hausaufgabe: Craig Nr. 2
- Das *Deduktionstheorem*
- Motivation: Echte Anwendungen
- Der Tableaux-Kalkül
  - Exkurs: Bäume
  - Tableaux-Konstruktion

Wiederholung

- Semantik der Aussagenlogik
  - Wahl der atomaren Aussagen definiert eine Abstraktion der Welt
  - Interpretation (=“Variablenbelegungen”): Mögliche Ausprägung dieser Welt
  - Modell von F (Interpretation mit $I(F) = 1$): Mit F verträgliche Welt
- Modellmengen
  - Semantik von logischen Operatoren entspricht Mengenoperationen auf Modellmengen
  - Z.B. $\mathrm{Mod}(A \wedge B) = \mathrm{Mod}(A) \cap \mathrm{Mod}(B)$
  - Z.B. $\mathrm{Mod}(A \to B) = \overline{\mathrm{Mod}(A)} \cup \mathrm{Mod}(B)$
- Logische Folgerung
  - $M \models F$ gdw. $\mathrm{Mod}(M) \subseteq \mathrm{Mod}(F)$
- Wahrheitstafelmethode
- Hausaufgabe: Craig 2

Hausaufgabe: 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.

Zusammenfassung

- Das *Deduktionstheorem*
  - $KB \models F$ gdw. $\models KB \rightarrow F$
- Motivation: Echte Anwendungen
  - Enumerieren von Interpretationen ist unplausibel
  - Idee: Widerspruchskalküle
- Der Tableaux-Kalkül
  - Exkurs: Bäume
  - Tableaux-Konstruktion

Feedback

- Was war der beste Teil der heutigen Vorlesung?
- Was kann verbessert werden?
  - Optional: Wie?

Ziele Vorlesung 14

- Wiederholung/Auffrischung
- Abschluss Tableaux
  - Praktische Umsetzung
  - Korrektheit und Vollständigkeit des Tableaux-Kalküls
  - Beweise und Modelle

Wiederholung

- *Deduktionstheorem*: $KB \models F$ gdw. $\models KB \rightarrow F$
- Der Tableaux-Kalkül
  - Exkurs: Bäume
  - Tableaux-Konstruktion
    - Uniforme Notation, $\alpha-$ und $\beta-$Regeln
    - Geschlossene und vollständige Äste
    - Geschlossene und vollständige Tableaux

Zur Vorlesung

Hausaufgabe: Mehr Tableaux

- Bearbeiten Sie die Übung: Mehr Tableaux zu Ende.

Zusammenfassung

- Abschluss Tableaux
  - Praktische Umsetzung
  - Korrektheit und Vollständigkeit des Tableaux-Kalküls
    - Semantik von Ästen
    - Erfüllbare Formel hat offenen Ast
    - Geschlossenes Tableau impliziert unerfüllbare Formel
    - Bei Aussagenlogik: Tableau-Konstruktion endet irgendwann
  - Beweise und Modelle
    - Geschlossenes Tableau ist Beweis
    - Offenes vollständiges Tableaux gibt Modelle an

Feedback

- Was war der beste Teil der heutigen Vorlesung?
- Was kann verbessert werden?
  - Optional: Wie?

Ziele Vorlesung 15

- Wiederholung/Auffrischung
- Logische Äquivalenz
  - Begriff
  - Basen der Aussagenlogik
  - Substitutionstheorem
  - Äquivalenzen und Kalkül der Äquivalenzumformungen

Wiederholung

- Tableaux
  - Praktische Umsetzung: α vor β, Abhaken, ...
  - Korrektheit und Vollständigkeit des Tableaux-Kalküls
    - Semantik von Ästen
    - Erfüllbare Formel hat offenen Ast
    - Geschlossenes Tableau impliziert unerfüllbare Formel
    - Bei Aussagenlogik: Tableau-Konstruktion endet irgendwann
  - Beweise und Modelle
    - Geschlossenes Tableau ist Beweis
    - Bei einem offenen vollständigen Tableaux gibt jeder offene Ast ein Modell (genauer: Eine Familie von Modellen) an.

Zur Vorlesung

Hausaufgabe

- Bearbeiten Sie von der Übung: *Basis der Aussagenlogik* einen der nicht in der Vorlesung gerechneten Fälle.

Zusammenfassung

- Logische Äquivalenz
  - Begriff
  - Basen der Aussagenlogik
    - $\neg, \wedge, \vee$ ist Basis
  - Beweisprinzip: Induktion über den Aufbau/Strukturelle Induktion
  - Substitutionstheorem: Teilformeln dürfen durch äquivalente Teilformeln ersetzt werden
  - Konkrete Äquivalenzen
    - $A \vee B \equiv B \vee A$, $A \vee \top \equiv \top$, ...
  - Kalkül der Äquivalenzumformungen
    - $\models A$ gdw. $\vdash_{LU} A$

Feedback

- Was war der beste Teil der heutigen Vorlesung?
- Was kann verbessert werden?
  - Optional: Wie?

Ziele Vorlesung 16

- Wiederholung/Auffrischung
- Normalformen
  - NNF
  - Klauseln und Klauselnormalform

Wiederholung

- Logische Äquivalenz
  - $A \equiv B$ gdw. $\mathrm{Mod}(A) = \mathrm{Mod}(B)$ gdw. $A \models B$ und $B \models A$
  - Basen der Aussagenlogik
    - $\neg, \wedge, \vee$ ist Basis
    - $\neg, \wedge$ ist Basis
    - $\neg, \rightarrow$ ist Basis
    - ...
  - Beweisprinzip: Induktion über den Aufbau/Strukturelle Induktion
  - Substitutionstheorem: Teilformeln dürfen durch äquivalente Teilformeln ersetzt werden
  - Konkrete Äquivalenzen
    - $A \vee B \equiv B \vee A$, $A \vee \top \equiv \top$, ...
  - Kalkül der Äquivalenzumformungen
    - $\models A$ gdw. $\vdash_{LU} A$

Zur Vorlesung

Zusammenfassung

- Normalformen
  - Negations-Normalform (nur $\neg, \wedge, \vee$ und $\neg$ nur vor Atomen)
    - Elimination von Implikation und Äquivalenz
    - De-Morgan und $\neg\neg A \equiv A$
  - Klauseln und Klauselnormalform
    - NNF, dann "Ausmultiplizieren"
    - Ergebnis: Menge (implizit Konjunktion) von Klauseln (Disjunktionen von Literalen)
- Disjunktive Normalform gibt es auch

Feedback

- Was war der beste Teil der heutigen Vorlesung?
- Was kann verbessert werden?
  - Optional: Wie?

Ziele Vorlesung 17

- Wiederholung/Auffrischung
- Resolution für die Aussagenlogik
  - KNF in Mengendarstellung
  - Interpretationen als Literalmengen
  - Saturierung
  - Resolutionsregel
  - Korrektheit und Vollständigkeit
  - Saturierungsalgorithmen

Wiederholung

- Normalformen
  - Negations-Normalform (nur $\neg, \wedge, \vee$ und $\neg$ nur vor Atomen)
    - Elimination von Implikation und Äquivalenz
    - De-Morgan und $\neg\neg A \equiv A$
  - Klauseln und Klauselnormalform
    - NNF, dann "Ausmultiplizieren"
    - Ergebnis: Menge (implizit Konjunktion) von Klauseln (Disjunktionen von Literalen)
    - Vereinfachungen sind möglich (mehrfach-Literale, Klauseln mit komplementären Literale, ...)

Zur Vorlesung

Hausaufgabe

- Bearbeiten Sie Übung: Resolution von Jane.

Zusammenfassung

- Resolution für die Aussagenlogik
  - KNF in Mengendarstellung
  - Interpretationen als Literalmengen
  - Saturierung
  - Resolutionsregel
    - $a \vee R, \neg a \vee S \vdash R \vee S$
  - Korrektheit und Vollständigkeit
  - Level-Saturierung, *Given-Clause*-Algorithmus

Feedback

- Was war der beste Teil der heutigen Vorlesung?
- Was kann verbessert werden?
  - Optional: Wie?

Ziele Vorlesung 18

- Wiederholung/Auffrischung
- Prädikatenlogik erster Stufe
  - Syntax
  - Semantik

Wiederholung

- Resolution für die Aussagenlogik
  - KNF in Mengendarstellung
  - Interpretationen als Literalmengen
  - Saturierung
  - Resolutionsregel
    - $a \vee R, \neg a \vee S \vdash R \vee S$
  - Korrektheit und Vollständigkeit
  - Level-Saturierung, *Given-Clause*-Algorithmus

Zur Vorlesung

Zusammenfassung

Prädikatenlogik erster Stufe

- Syntax
  - Terme, Atome
  - Quantoren und komplexe Formeln
  - Freie/gebundene Variablen
- Semantik
  - Universum (Trägermenge)
  - Interpretationsfunktion
    - Variablen werden Werte des Universums zugeordnet
    - Funktionssymbolen werden Funktionen zugeordnet
    - Prädikatssymbolen werden Relationen zugeordnet
  - $\forall, \exists, \wedge, \vee, \neg, ...$

Feedback

- Was war der beste Teil der heutigen Vorlesung?
- Was kann verbessert werden?
  - Optional: Wie?

Ziele Vorlesung 19

- Wiederholung/Auffrischung
- Ziel: Logische Folgerung für Prädikatenlogik
  - Deduktionstheorem für Prädikatenlogik
  - Resolution für Prädikatenlogik (langfristiges Ziel)
- Normalformen für Prädikatenlogik
  - Negations-Normalform
  - Prenex-Normalform
  - Skolemisierung

Wiederholung

Prädikatenlogik erster Stufe

- Syntax
  - Terme, Atome
  - Quantoren und komplexe Formeln
  - Freie/gebundene Variablen
- Semantik
  - Universum (Trägermenge)
  - Interpretationsfunktion
    - Variablen werden Werte des Universums zugeordnet
    - Funktionssymbolen werden Funktionen zugeordnet
    - Prädikatssymbolen werden Relationen zugeordnet
  - $\forall, \exists, \wedge, \vee, \neg, ...$

Zur Vorlesung

Zusammenfassung

- Deduktionstheorem für Prädikatenlogik
  - Nur für geschlossene Formeln!
- Normalformen für Prädikatenlogik
  - Negations-Normalform
    - Neu: $\neg$ über Quantoren schieben
  - Prenex-Normalform
    - Variablen-Normierung
    - Prefix und Matrix
  - Skolemisierung
    - Eliminierung von $\exists$

Feedback

- Was war der beste Teil der heutigen Vorlesung?
- Was kann verbessert werden?
  - Optional: Wie?

Ziele Vorlesung 20

- Wiederholung/Auffrischung
- Konjunktive Normalform/Klauselnormalform
- Satz von Herbrand
  - Grundtermmodelle sind ausreichend!
- Substitutionen und Instanzen
- Unifikation

Wiederholung

- Deduktionstheorem für Prädikatenlogik
- Normalformen für Prädikatenlogik
  - Negations-Normalform
    - Neu: $\neg$ über Quantoren schieben
  - Prenex-Normalform
    - Variablen-Normierung
    - Trennung in Prefix und Matrix
  - Skolemisierung
    - Eliminierung von $\exists$

Zur Vorlesung

Hausaufgabe

- Bearbeiten Sie Übung: Unifikation.

Zusammenfassung

- Konjunktive Normalform/Klauselnormalform
- Satz von Herbrand
  - Grundtermmodelle sind ausreichend!
- Substitutionen und Instanzen
- Unifikation
  - Unifikation als paralleles Gleichungslösen

Feedback

- Was war der beste Teil der heutigen Vorlesung?
- Was kann verbessert werden?
  - Optional: Wie?

Ziele Vorlesung 21

- Wiederholung/Auffrischung
- Resolutionskalkül (endlich!)
  - Kalkül
  - Beispiele
  - Diskussion
- Rückblick Gesamtvorlesung

Wiederholung

- Satz von Herbrand
  - Grundtermmodelle sind ausreichend!
- Substitutionen und Instanzen (Ersetzen von Variablen)
- Unifikation
  - Unifikation als paralleles Gleichungslösen
  - Berechnen des *mgu*

Zur Vorlesung

Wiederholung (1)

- Mengenlehre
  - Mengen und Mengenoperationen
  - Formale Systeme (MIU, Rechnen in Termalgebra)
  - Relationen (Teilmengen des kartesischen Produkts)
    - Eigenschaften: Binär, Homogen, ...
    - Relationenalgebra
    - Hüllenbildung (reflexiv/transitiv/symmetrisch)
    - Tabellen/Graphdarstellung
  - Funktionen
- Funktionales Programmieren/Scheme
  - Syntax (alles Klammern)
  - Semantik (normal vs. *special forms*)
  - Rekursion
  - Listen/Sortieren (`cons`, '(), `car`, `cdr`...)
  - Funktionale und destruktive Features

Wiederholung (2)

- Aussagenlogik
  - Syntax
  - Semantik: Interpretationen, Modelle, (Un-)Erfüllbarkeit
  - Logisches Folgern
  - Äquivalenz und Normalformen
  - Beweisverfahren: Strukturelle Induktion
  - Tableaux
    - $\alpha-$ und $\beta-$Regeln
    - Offene (vollständige) Äste und Modelle
    - Geschlossene Tableaux und Unerfüllbarkeit
  - Resolution
    - KNF-Transformation
    - Klauseln als Mengen
    - Resolutions-Inferenz
    - Saturieren

Wiederholung (3)

- Prädikatenlogik
  - Syntax (neu: Variablen, Terme, Quantoren)
  - Semantik (Interpretationen und Modelle)
    - Allgemein: Beliebiges Universum, beliebige Funktionen!
    - Spezialfall: Herbrand-Interpretationen/Modelle
  - Normalformen (NNF, PNF, SNF, KNF)
    - Skolemisieren!
  - Substitutionen und Instanzen
  - Resolution
    - Unifikation (als Gleichungslösen)
    - Faktorisierung (selten gebraucht)
    - Resolution (oft gebraucht) mit Unifikation
    - Saturieren (fair!)

Zusammenfassung

- Resolutionskalkül
  - Faktoriseren (Literale einer Klausel durch Instanziierung gleich machen)
  - Resolution (Potentiell komplementäre Literale aus zwei Klauseln unifizieren und resolvieren)
  - Anwendung: Fair saturieren
- Rückblick Gesamtvorlesung
  - Mengenlehre/Funktionen/Relationen
  - Funktionales Programmieren/Scheme/Rekursion
  - Aussagenlogik/Äquivalenzen/Tableaux/Resolution
  - Prädikatenlogik/Skolemisierung/Resolution

Feedback

- Was war der beste Teil der heutigen Vorlesung?
- Was kann verbessert werden?
  - Optional: Wie?

Ziele Vorlesung 22

- Übungsklausur
- Diskussion Übungsklausur
- Weitere Fragen

Übungsklausur

Zusammenfassung

- Yep!

Feedback

- Was hat die Vorlesung gebracht?
  - Was haben Sie gelernt?
  - Hat es Spaß gemacht?
  - Habe ich das “warum” ausreichend klargemacht?
- Was kann verbessert werden?
  - Optional: Wie?

Ende