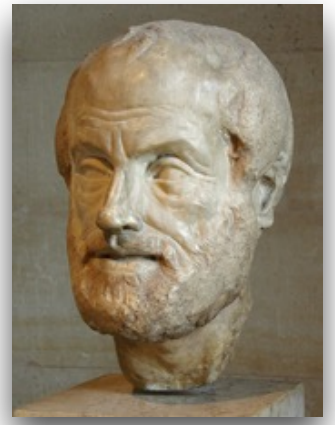
Logik und Grundlagen der Informatik

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

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

Research: Logik & Deduktion

Rechnerausstattung

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

Real Programmers

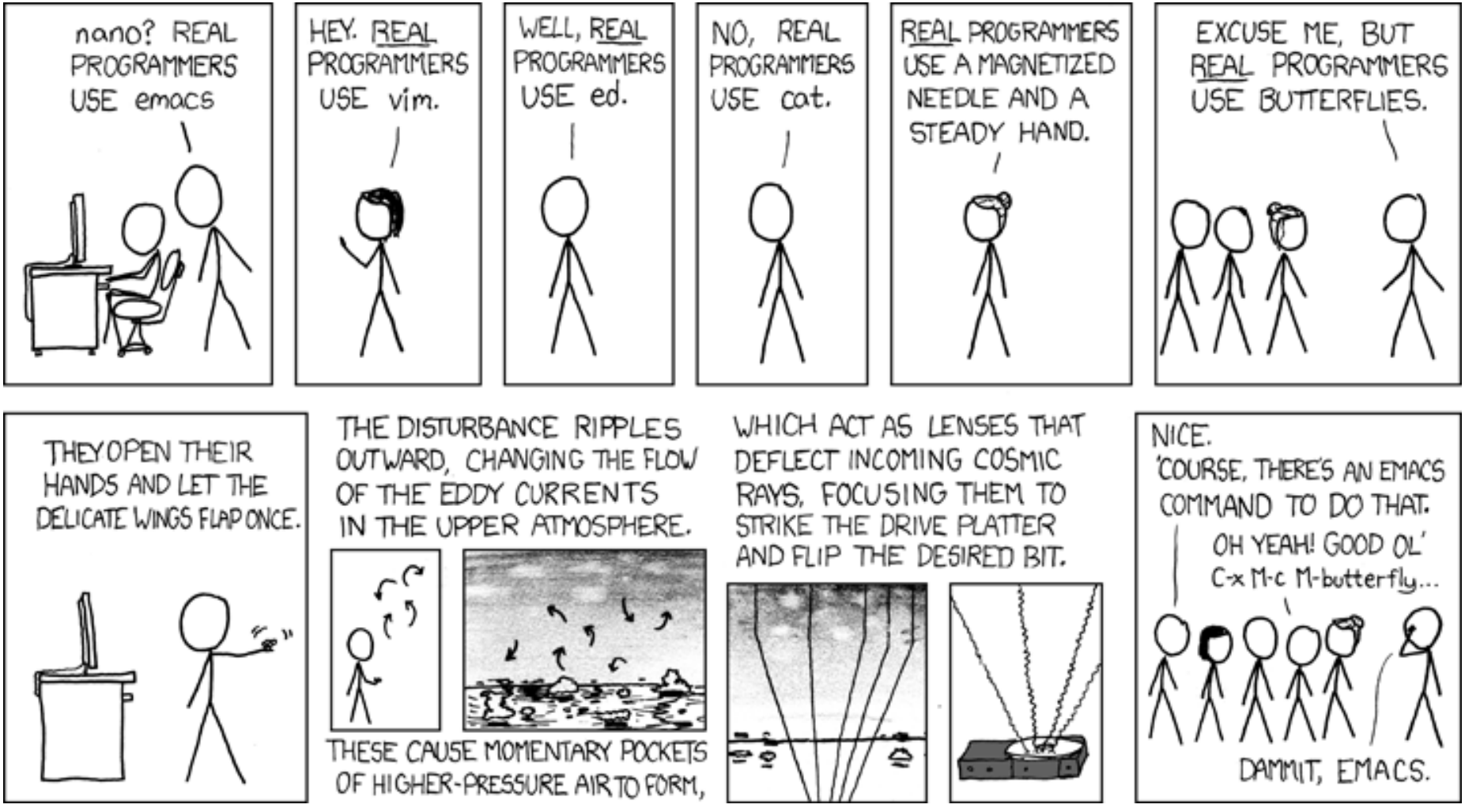


Image credit: `http://xkcd.com/378/`

Semesterübersicht

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

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 Russel: Introduction to Mathematical Philosophy (1918), `http://www.gutenberg.org/ebooks/41654`
  - 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 3.0: `http://mitpress.mit.edu/sicp/`

Ziele der Vorlesung

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

Übung: Das MIU-Rätsel

- Wir betrachten ein formales System mir 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 $x \vdash y$, 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

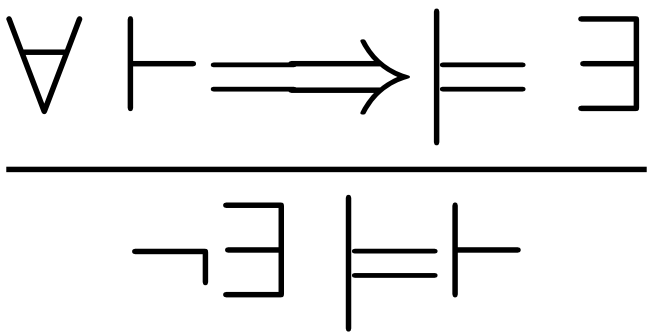
Vokabular
Methoden

Logik

- Ziel
  - Formalisierung rationalen Denkens
  - Ursprünge: Aristoteles („Organon“, „De Interpretatione“), Al-Farabi, Boole, Frege, Russel&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?

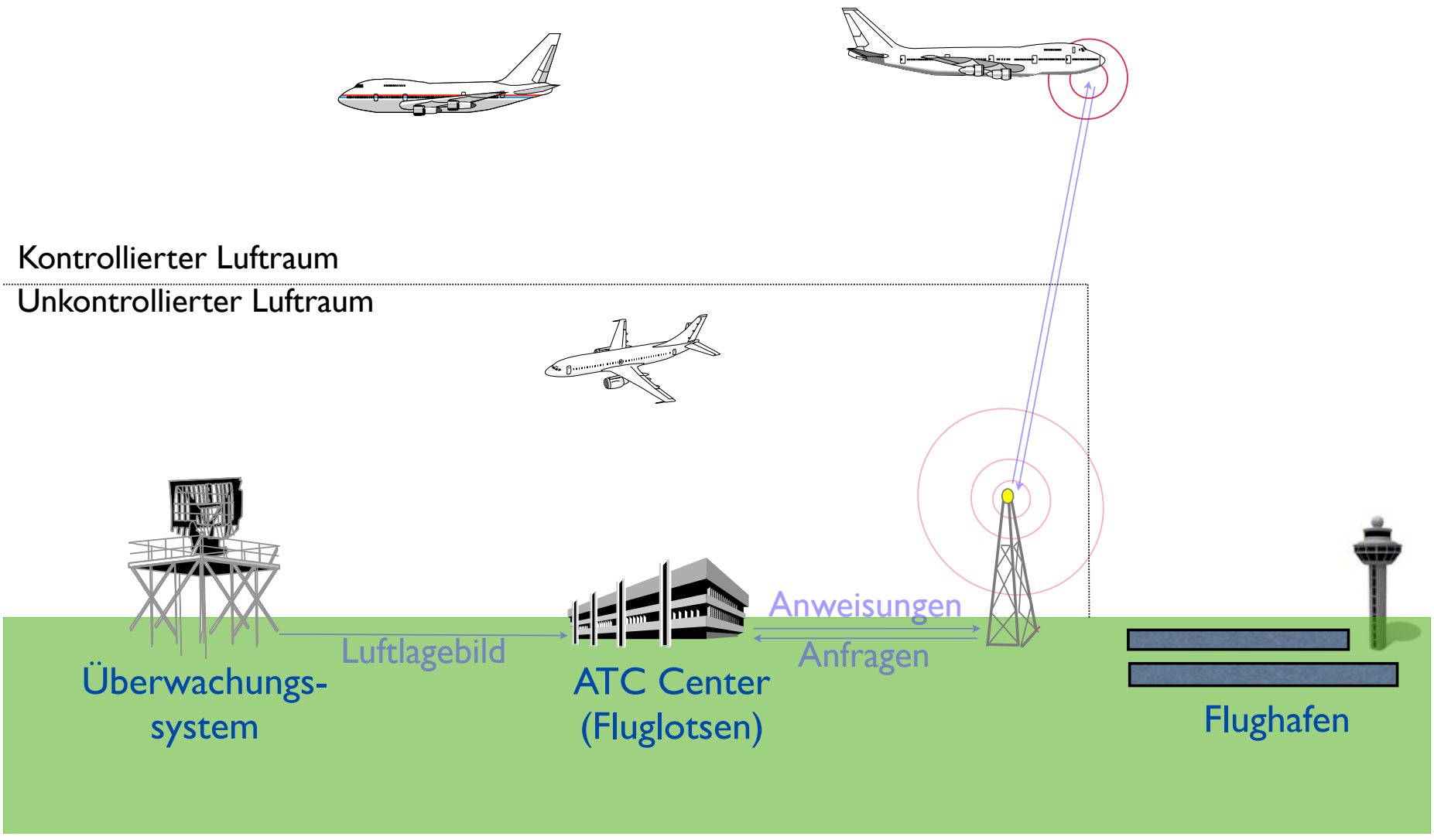


Anwendungsbeispiel: Äquivalenz von Spezifikationen

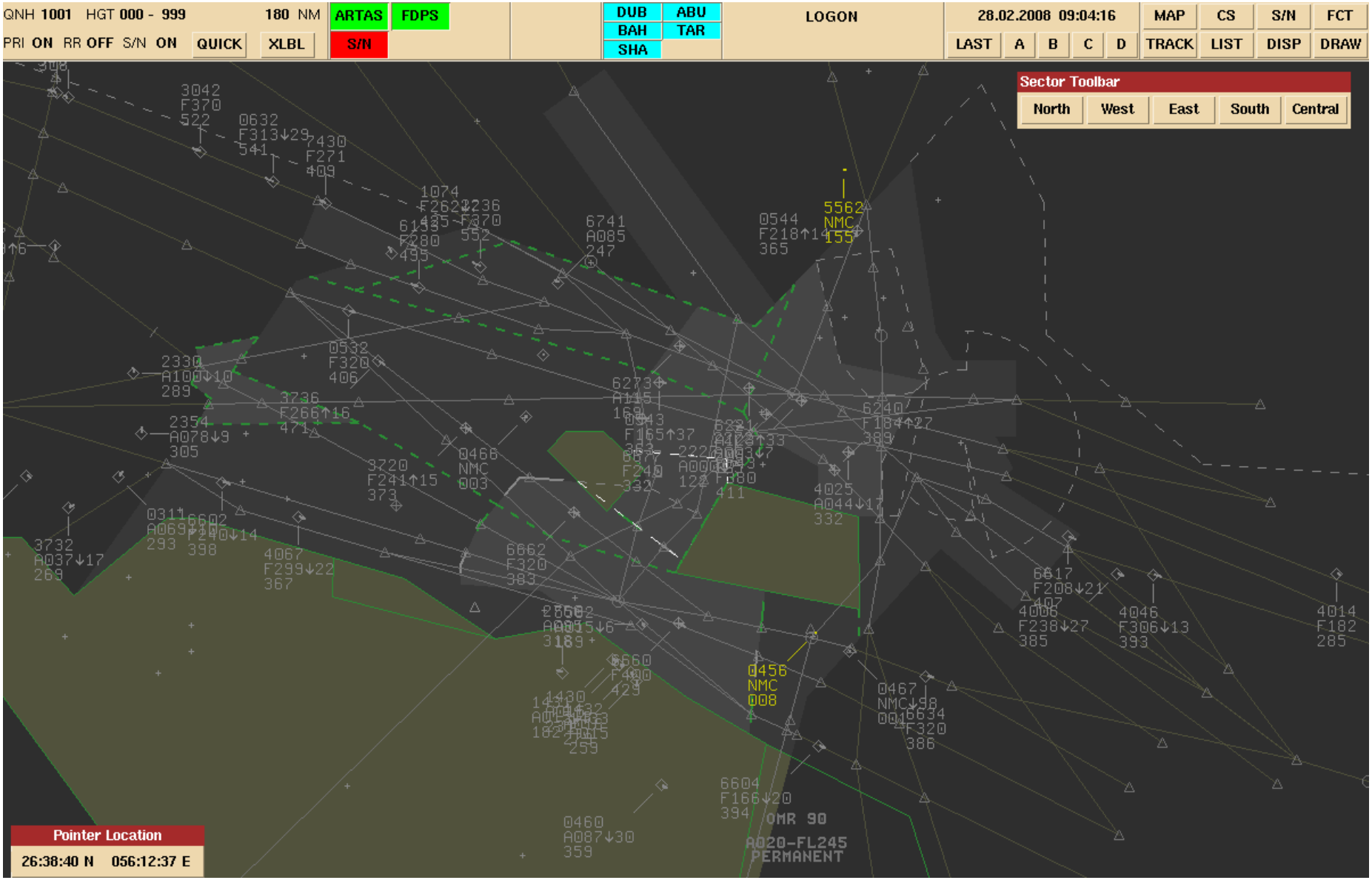
- Entwicklungsprozess (Auszug):
  - Kundenspezifikation
  - Systemspezifikation
  - Software-Design
  - Implementierung
- Problem: Äquivalenz der verschiedenen Ebenen
  - Manuelle Überprüfung aufwendig und fehleranfällig
  - Äquivalenz nicht immer offensichtlich
  - Ausgangsspezifikation nicht immer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

Geht das auch konkreter?

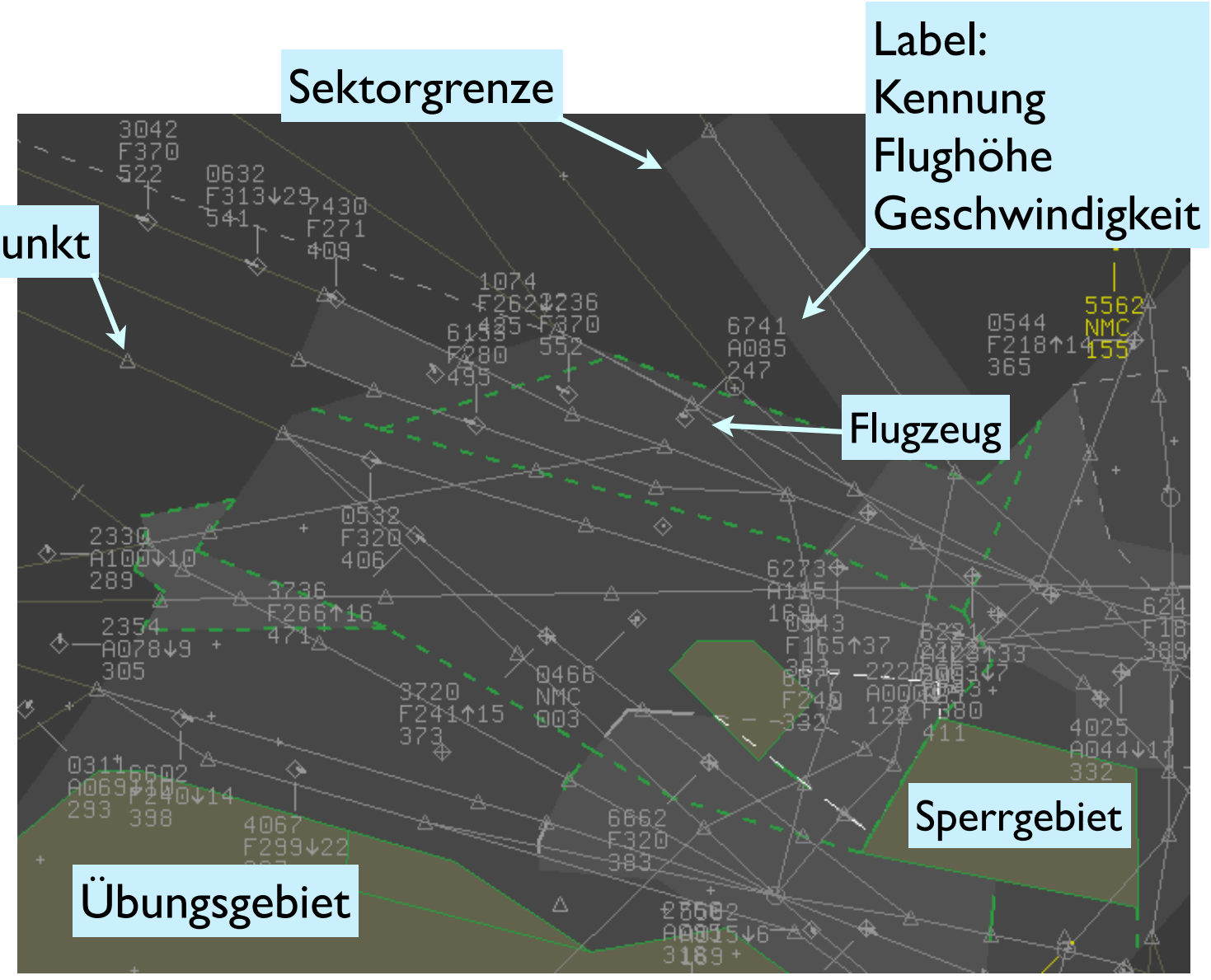
Beispiel: Flugsicherung



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 $\leftrightarrow$ inregion(X, [abu-dhabi-koord])
  - $\forall X$: dub-app $\leftrightarrow$ inregion(X, [dubai-koord])
  - $\forall X$: milregion $\leftrightarrow$ inregion(X, [training-koord])
  - $\forall X$: lowairspace $\leftrightarrow$ altitude(X, 0, 100)
  - $\forall X$: uppairspace $\leftrightarrow$ altitude(X, 100, 900)
  - $\forall X$: military(X) $\leftrightarrow$ modeA(X, [mil-code-list])

Luftraum UAE (II)

- Filterlösung: Stelle Flugzeug X genau dann da, 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:

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

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 n
% 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

Hausaufgabe

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

Zurück B

Zurück C

Mengenlehre

Definitionen

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: 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: 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“):
  - $A = \{x | x \text{ ist Primzahl und } x \leq 13\}$
- Mengenzugehörigkeit
  - $2 \in A$ (2 ist in A, 2 ist Element von A)
  - $4 \notin A$ (4 ist nicht in A, 4 ist kein Element von A)

Basiseigenschaften von Mengen

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

Teilmengen, Mengengleichheit

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

- Schreibweise: $M_1 \subseteq M_2$

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

- Schreibweise: $M_1 = M_2$

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

Vokabular: gdw. steht für „genau dann, wenn“

Echte Teilmengen, Obermengen

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

- Schreibweise: $M_1 \subset M_2$
- 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

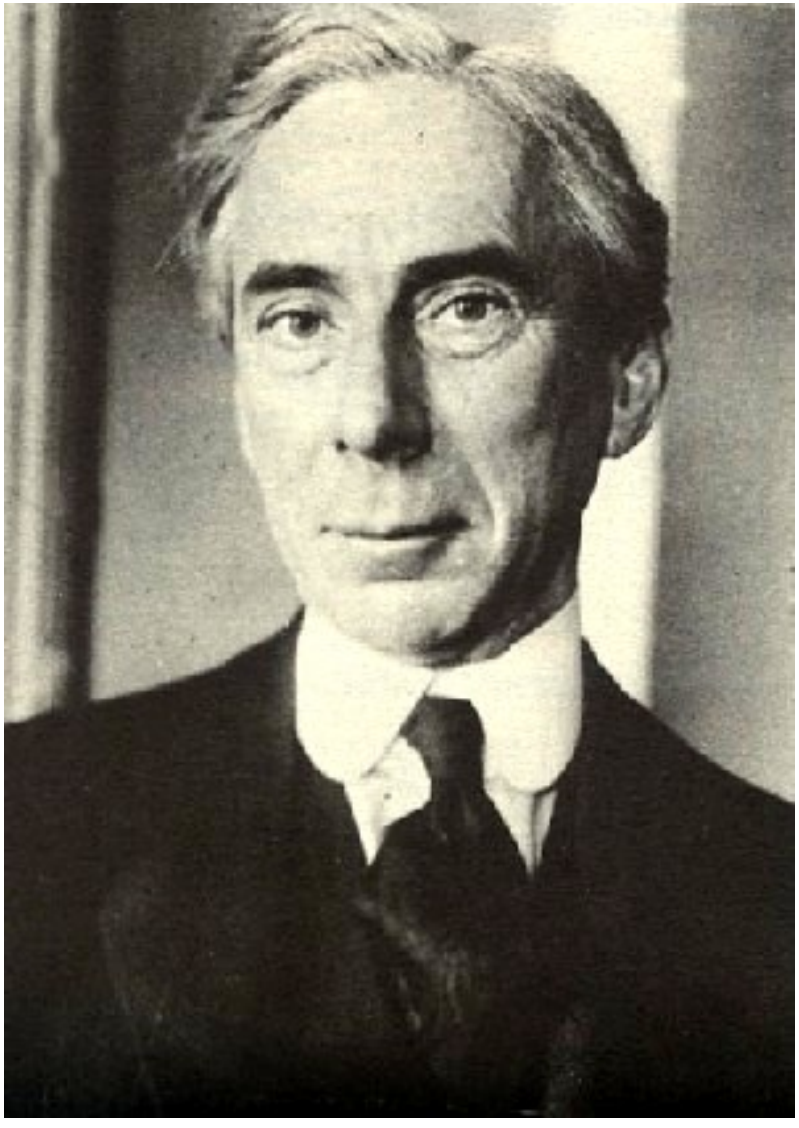
Die ganzen Zahlen hat der liebe Gott gemacht, alles andere ist Menschenwerk.
Leopold Kronecker (1823–1891)

Übung: Mengenbeschreibungen

- Geben Sie formale Beschreibungen 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 Russel (1872–1970) Alfred North Whitehead (1861-1947)

Mengenlehre ist die Grundlage der Mathematik (1)

Wir definieren eine Familie von Mengen wie folgt:

$M_0 =$	$\{\}$	Leere Menge
$M_1 =$	$\{\{\}\}$	Menge, die (nur) $\{\}$ enthält
$M_2 =$	$\{\{\{\}\}\}$	usw.
$M_3 =$	$\{\{\{\{\}\}\}\}$	usf.
$\ldots$	$\ldots$	$\ldots$
$M_{i+1} =$	$\{M_i\}$	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:

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

Mengenlehre ist die Grundlage der Mathematik (3)

Und bequemere Schreibweise:

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

- Wir geben folgende Regeln an:
  - $a(x, 0) = x$
  - $a(x, s(y)) = s(a(x, y))$
- Beispielrechnung:

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

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

Übung: Multiplikation rekursiv

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

Übung: Konstruktion der negativen Zahlen

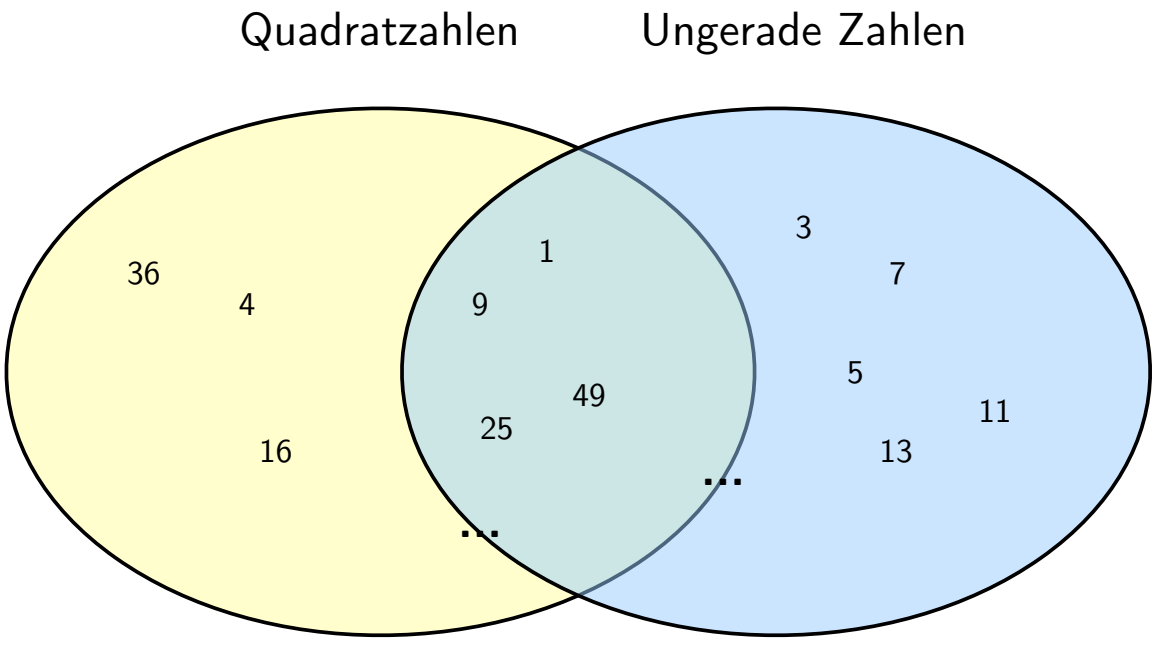
- Erweitern Sie die rekursiven Definitionen von p und m (Plus und Mal) auf $s, 0$-Termen, um auch die negativen Zahlen behandeln zu können.
- Hinweis: Benutzen sie p („Vorgänger von“) und n (Negation)

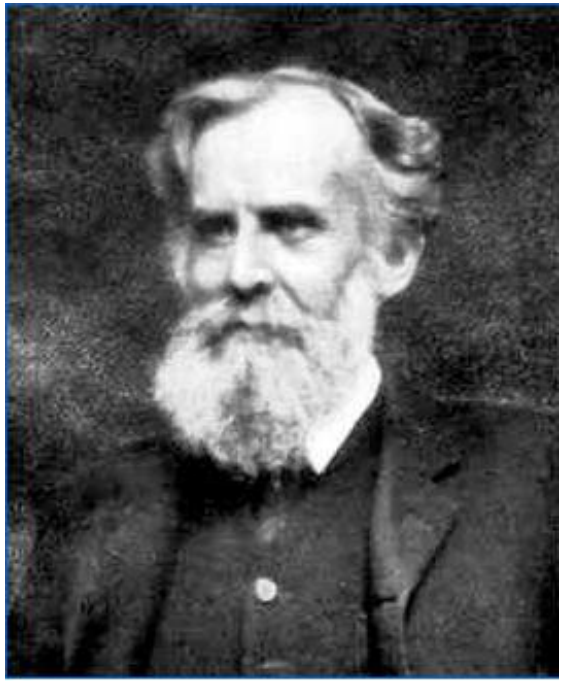
Zurück C

Mengenoperationen

Venn-Diagramme

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





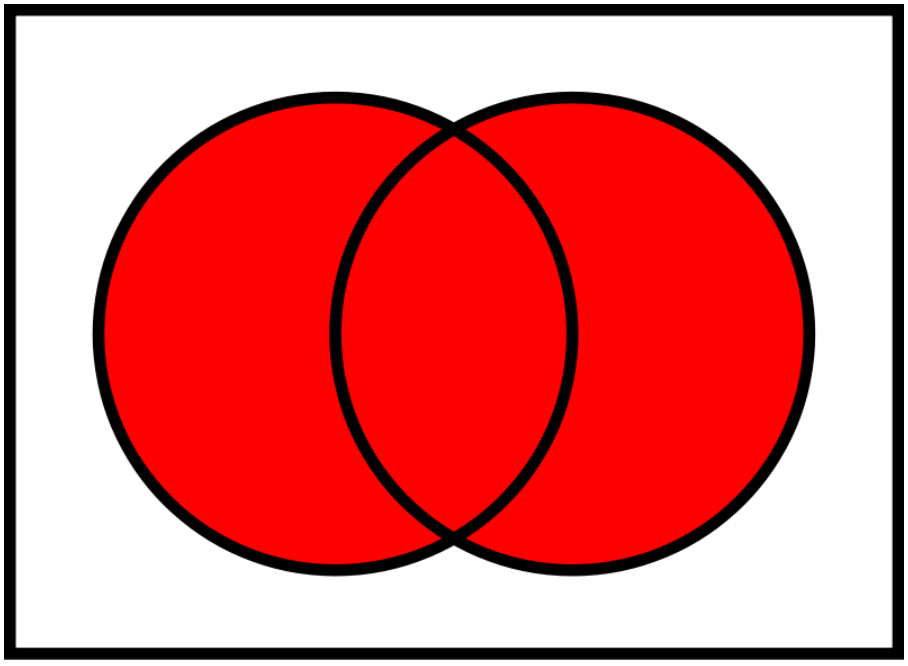
John Venn
(1834–1923)

Mengenoperationen

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

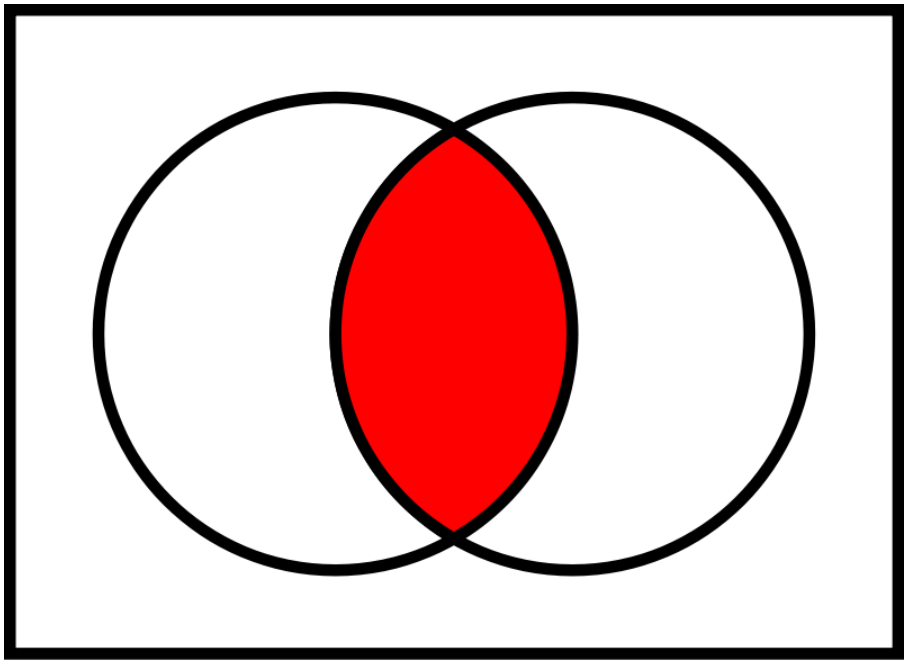
- Vereinigung
- Schnitt
- Differenz
- Symmetrische Differenz
- Komplement

Vereinigungsmenge



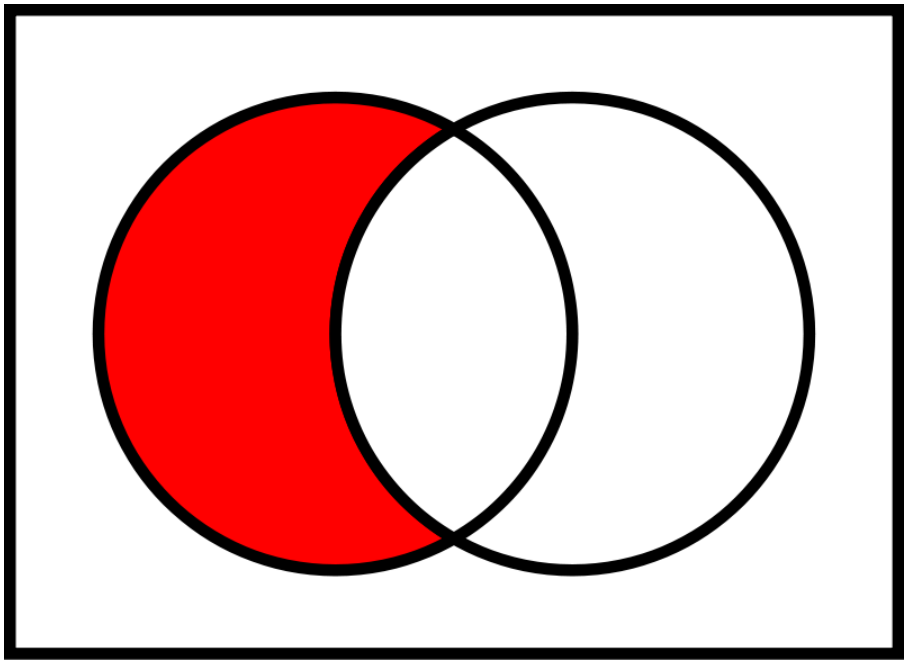
- $M_1 \cup M_2 = \{x | x \in M_1 \text{ oder } x \in M_2\}$
- $x \in M_1 \cup M_2$ gdw. $x \in M_1$ oder $x \in M_2$

Schnittmenge



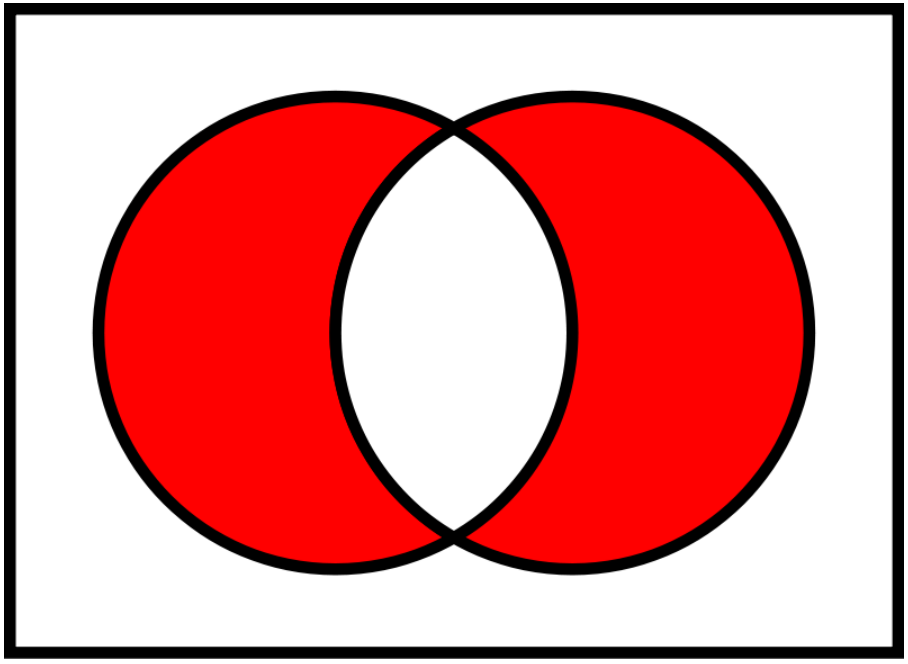
- $M_1 \cap M_2 = \{x | x \in M_1 \text{ und } x \in M_2\}$
- $x \in M_1 \cap M_2$ gdw. $x \in M_1$ und $x \in M_2$

Differenz



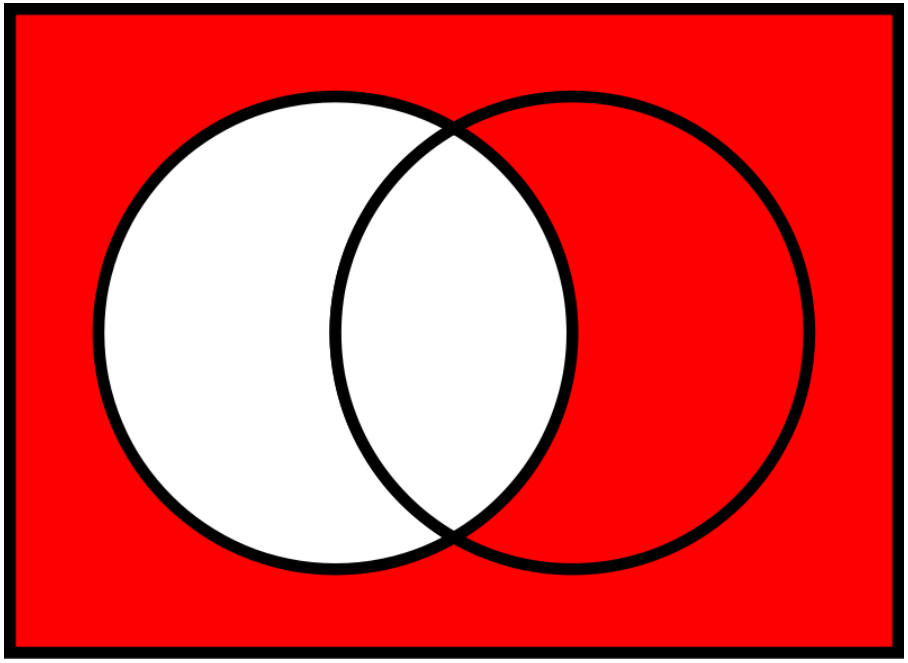
- $M_1 \setminus M_2 = \{x | x \in M_1 \text{ und } x \notin M_2\}$
- $x \in M_1 \setminus M_2$ gdw. $x \in M_1$ und $x \notin M_2$

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!

Zurück B

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 und eine umgangssprachliche Charakterisierung des Ergebnisses an.
  - $M_1 \cup M_2$
  - $M_1 \cap M_2$
  - $M_1 \backslash M_2$
  - $M_1 \backslash \overline{M_2}$
  - $M_1 \triangle M_2$

Diskussion: Übung Mengenoperationen

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

Karthesisches Produkt

Definition: Das kartesische Produkt $M_1 \times M_2$ zweier Mengen M_1 und $M2$ ist die Menge $\{(x, y)|x \in M_1, y \in M_2\}$.

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

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

Potenzmengen

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

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

Übung: Karthesisches Produkt und Potenzmenge

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

Lösung: $M_1 \times M_1$

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

Lösung: $M_1 \times M_2$

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

Lösung: $M_2 \times M_1$

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

Lösung: Potenzmenge

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

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

Übung: M^3

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

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

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
  - $M_1 \cup (M_2 \cap M_3) = (M_1 \cup M_2) \cap (M_1 \cup M_3)$
  - $M_1 \cap (M_2 \cup M_3) = (M_1 \cap M_2) \cup (M_1 \cap M_3)$
- Inverse Elemente
  - $M_1 \cup \overline{M_1} = T$
  - $M_1 \cap \overline{M_1} = \emptyset$
- Idempotenz
  - $M_1 \cup M_1 = M_1$
  - $M_1 \cap M_1 = M_1$
- Gesetze von De-Morgan
  - $\overline{M_1 \cup M_2} = \overline{M_1} \cap \overline{M_2}$
  - $\overline{M_1 \cap M_2} = \overline{M_1} \cup \overline{M_2}$
- Doppelte Komplementbildung
  - $\overline{\overline{M_1}} = M_1$

Relationen

Allgemeine Relationen

Definition: 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 karthesischen 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: 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, fall R homogen und binär ist.**

- Wenn R homogen ist, so nennen wir R auch eine Relation über M
- Im Fall von binären Relationen schreiben wir oft xRy statt $R(x, y)$ (z.B. $1 < 2$ statt $<(1, 2)$ oder $(1, 2) \in <$)
- Im folgenden nehmen wir bis auf weiteres an, dass Relationen homogen und binär sind, soweit nichts anderes spezifiziert ist

Beispiele

- Beispiele für homogene binäre Relationen:
  - $=$ über $\mathbb{N}$
    - $= \; = \{(0,0),(1,1),(2,2),\ldots\}$
  - $<$ über $\mathbb{Z}$
    - $< \; = \{(i, i+j) \mid i \in \mathbb{Z}, j \in \mathbb{N}^+\}$
  - $\neq$ über $\{w | w$ 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: Sei R eine binäre Relation über $A \times B$.

- **Gilt für alle $\forall a \in A \exists b \in B$ mit $R(a, b)$, so heißt R linkstotal**
- **Gilt für alle $\forall a \in A, \forall b, c \in B : R(a, b)$ und $R(a, c)$ impliziert $b = c$, so heisst R rechtseindeutig**

- Linkstotal: Jedes Element aus A steht mit mindestens einem Element aus B in Relation
- Rechtseindeutig: Jedes Element aus A steht mit höchstens einem Element aus B in Relation.

Eigenschaften von Relationen (2)

Definition: Sei R eine homogene binäre Relation über A.

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

Übung

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

Zurück C

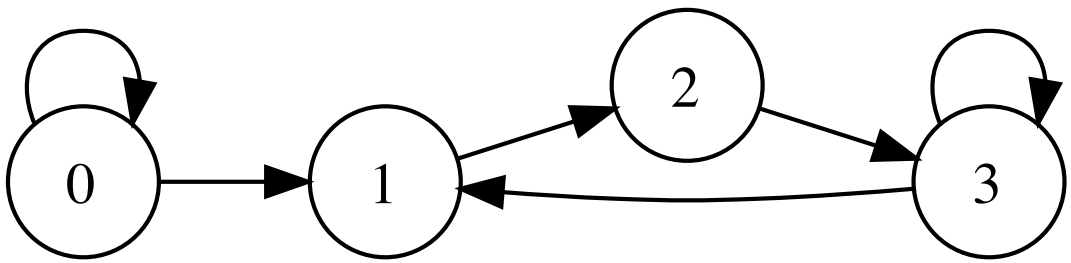
Darstellung von Relationen: Mengendarstellung

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)

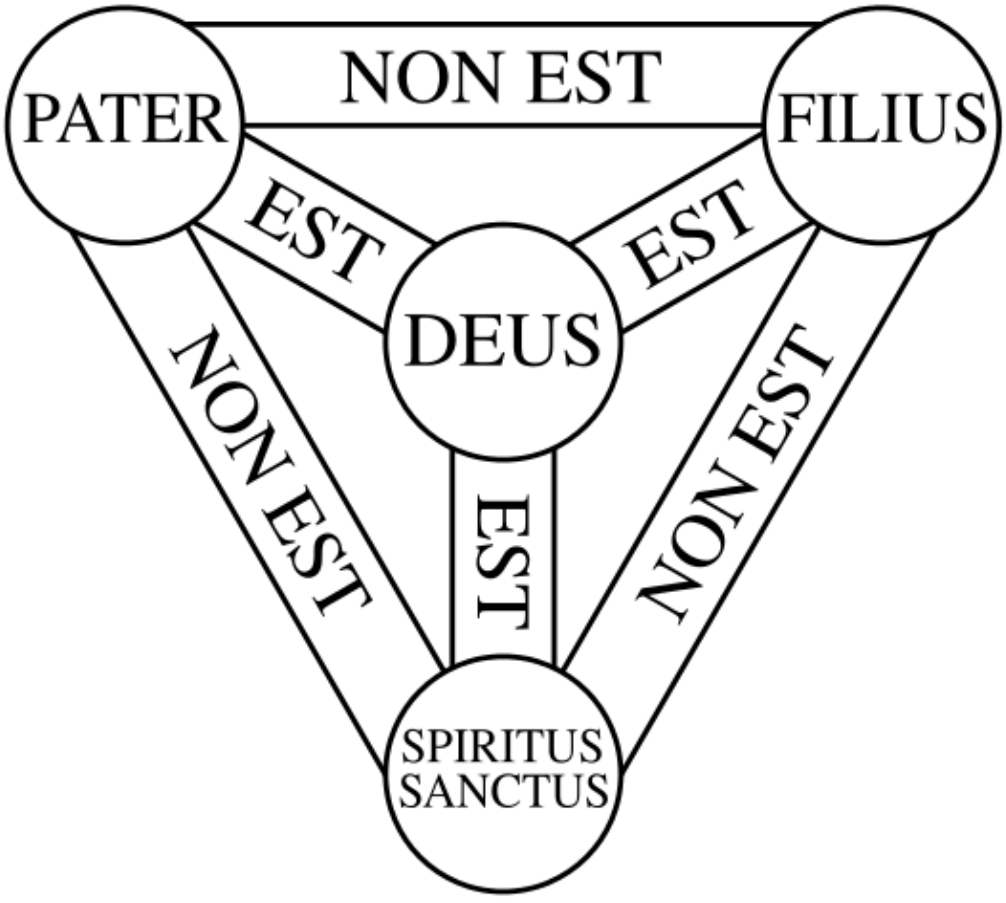
Darstellung von (endlichen) Relationen: Graphdarstellung

- 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



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

Darstellung von (endlichen) Relationen: Tabellendarstellung

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

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

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

- Vorteile:
  - Sehr einfach im Rechner realisierbar
  - Prüfen, ob xRy geht schnell („lookup“)
  - Übersichtlicher als Mengen
  - Manche Eigenschaften können leicht erkannt werden
- Nachteile:
  - Viel Speicherbedarf (immer n^2 Einträge)
  - Ich verwechsele immer Zeilen und Spalten ;-)

Die inverse Relation

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

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

Verknüpfung von Relationen

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

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

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

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

Übung: Relationen

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

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

Zurück B

Hüllenbildung

Definition: Seien R eine Relation. Dann gilt:

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

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

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

Übung

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

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

und stellen Sie sie als Tabelle/Matrix da.

Funktionen

Definition: Seien M, N Mengen.

- Eine (totale) Funktion $f : M \to N$ 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.

- Anmerkungen:
  - 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
  - M heißt Definitionsmenge von f
  - N heißt Zielmenge von f
- Oft wird eine konkrete Funktion durch eine Zuordnungsvorschrift definiert:
  - $f : \mathbb{N} \to \mathbb{N}, x \mapsto x^2$
  - $g : \mathbb{Z} \to \mathbb{N}, x \mapsto |x|$
  - Wir schreiben konkret: $f(x) = y$ statt $(x, y) \in f$ oder xfy

Bild einer Menge

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

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

Eigenschaften von Funktionen

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

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

- Wenn f surjektiv ist, so gilt $f(M) = N$
- Wenn f injektiv ist, so ist f^{-1} rechtseindeutig (also eine (partielle) Funktion)

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 injektiv, surjektiv, bijektiv?
- Für f_1, f_2, f_3: Was ist jeweils das Bild und das Urbild von $\{2, 4, 6, 8\}$?
- Für f_4: Was ist das Urbild von $\{6, 8\}$?

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 heißt abzählbar, wenn Sie die selbe Kardinalität wie $\mathbb{N}$ hat

Übung: Kardinalität

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

Zurück C

Übung: Relationen für Fortgeschrittene

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

Funktionales Programmieren mit Scheme

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 Russel (1937–)

Unpersonal Computers

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

Lisp in the Real World

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

Übung: Hello World

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

;; Aufruf der Funktion
(hello)
```

- Führen Sie das Programm aus
  - Eintippen oder Download von `http://wwwlehre.dhbw-stuttgart.de/~sschulz/lgli2014.html`
  - Von der Kommandozeile (z.B.): `> guile-2.0 hello.scm`
  - Aus dem Scheme-Interpeter: `> (load "hello.scm")`

Ein funktionaleres Beispiel

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

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)))))`

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)
```

Berechnen durch Ausrechnen

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

```
> 10
==> 10
> "Hallo"
==> "Hallo"
> hallo
;;; <unknown-location>: warning: possibly unbound variable
```

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

Beispiel

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

Besonderheiten

- Problem:

  `( 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 Argumenten, und der Rechenvorschrift *exprs*
  - `> (define (plus3 x) (+ x 3))`
  - `> (plus3 10) ==> 13`
- Der Rückgabewert einer `define`-Anweisung ist undefiniert
- `defines` stehen typischerweise auf der obersten Ebene eines Programms

Datentypen

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

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)

Zurück B

Lisp Jedi

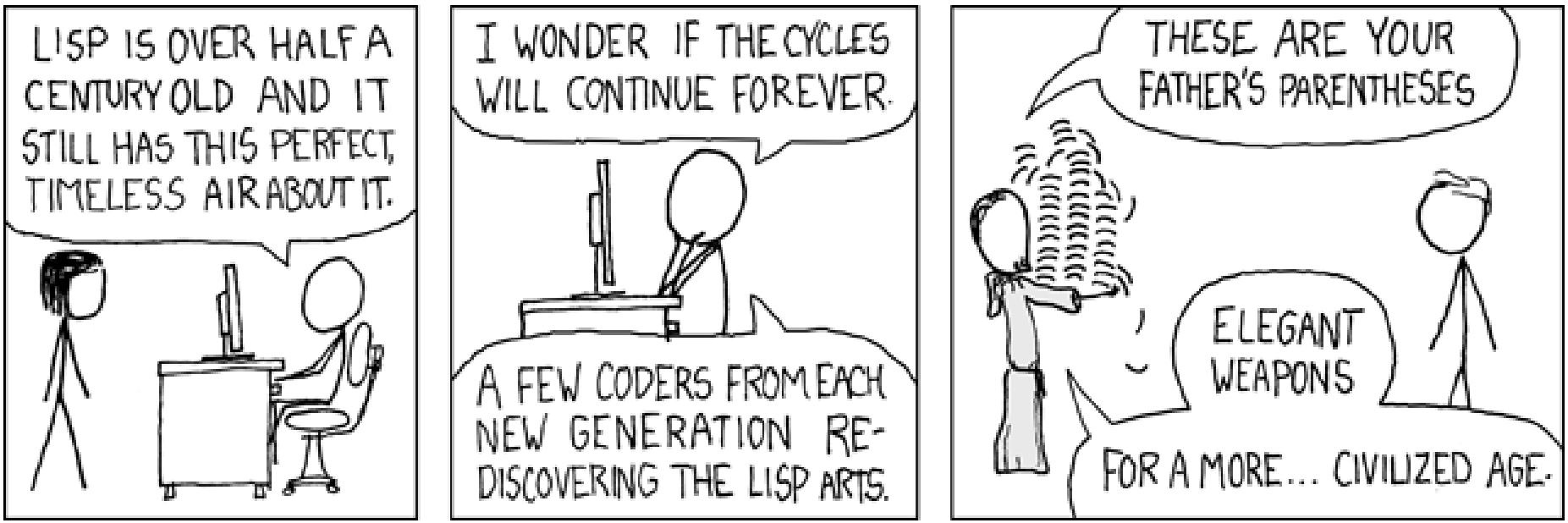


Image credit: `http://xkcd.com/297/`

Listenverarbeitung

- `'()`
  - Die leere Liste
- `(cons` *`obj list`*`)` („constructor“)
  - Hänge *obj* als erstes Element in *list* ein und gib das Ergebnis zurück
  - `> (cons 1 '(2 3 4)) ==> (1 2 3 4)`
- `(append` $list_1$ $list_2$`)`
  - 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)))
      )
  )
```

Übung: Revert

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

Tips zur Programmierung

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

Übung: Mengenlehre in Scheme

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

Funktionale Programmierung?

- Ein Programm besteht aus Funktionen
  - Die Ausführung entspricht der Berechnung 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 Eigenschaftren zurück
    - Aber: Aus Effizienzgründen manchmal aufgeweicht
  - Ein-/Ausgabe sind in Scheme immer Seiteneffekte
- Funktionen sind Werte
  - Funktionen können als Parameter übergeben werden
  - Funktionen können dynamisch erzeugt werden

Funktionsaufrufe

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

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

Was passiert beim Aufruf `(sumsquare 3 6)`?

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

Speichemodell und Variablen (1)

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

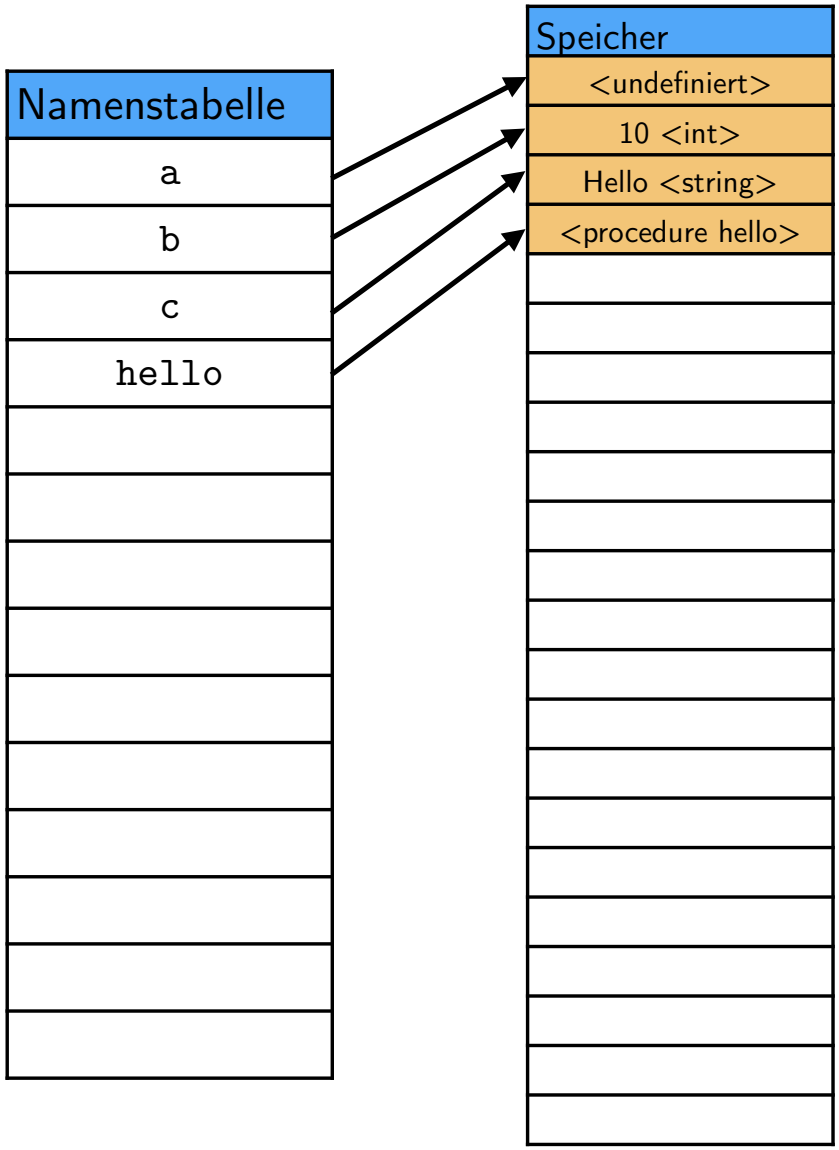
Speichermodell und Variablen (2)

```
(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)))

==> 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
```

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
```

Special Form: `let`

- `let` führt temporäre Variablen für Zwischenergebnisse ein
- Syntax: Das Schlüsselwort `let` wird gefolgt von einer Liste von Bindungen und einem Rumpf
  - Eine Bindung besteht aus einer Variablen und einem Ausdruck (dem Initialsierer)
  - 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 $n > 1$:q!
- Finden Sie für möglichst viele der Mengenfunktionen elegantere oder effizientere Implementierung.
  - Bearbeiten Sie dabei mindestens die Potenzmengenbildung.

Zurück C

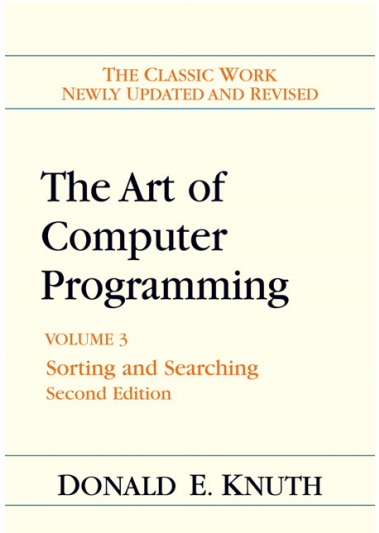
Sortieren durch Einfügen

- 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 mann `k` an der richtigen Stelle in `lst` einfügt
- Erstellen Sie eine Funktion `(isort lst)`
  - `(isort lst)` soll `lst` in aufsteigender Reihenfolge zurückgeben
  - Verwenden Sie Sortieren durch Einfügen als Algorithmus
- Beispiellisten können (in Guile) mit folgender Funktion generiert werden (auch auf `http://wwwlehre.dhbw-stuttgart.de/~sschulz/lgli2014.html`):

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

Sortieren allgemeiner

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

```
> (isort '(2 5 3 1))
==> (1 2 3 5)
> (isort_g '(2 5 3 1) <)
==> (1 2 3 5)
> (isort_g '(2 5 3 1) >)
==> (5 3 2 1)
> (isort_g '("Hallo" "Tschuess" "Ende") string>?)
==> ("Tschuess" "Hallo" "Ende")
```

Übung: Generisches Sortieren

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

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.

Zurück B

Zurück C

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

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

Rekursion – `is-element` (2)

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

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

Übung: Listenvervielfachung

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

```
> (scalar-mult '(1 2 3) 5)
==> (5 10 15)
> (scalar-mult '(7 11 13) 0.5)
==> (3.5 5.5 6.5)
> (scalar-mult '() 42)
==> ()
```

Input und Output

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

Ein- und Ausgabebefehle (1)

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

Ein- und Ausgabebefehle (2)

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

Ports 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!
>
```

Guile/R^{6}RS-Erweiterung: Kommandozeilen und Programmende

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

```
> (command-line)
==> ("myprog.scm" "Eins" "zwei" "drei")
```

  - Bei Racket: `(current-command-line-arguments)`
- Beenden des Programs: `(exit` *obj*`)`
  - *obj* ist optional
  - Ohne *obj*: Normales Ende (Erfolg)
  - Ist *obj* `#f`, dann abnormes Ende (Fehlerfall)
  - Ansonsten: Implementierungsdefiniert (aber in erster Näherung: Kleine numerische Werte werden als OS-*exit()*-Werte interepretiert

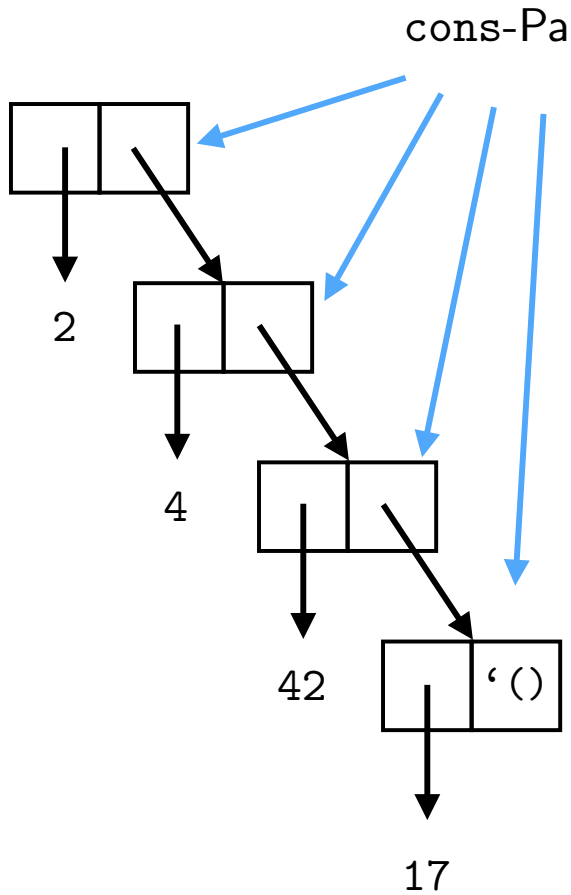
Übung: Komplexität von Sortierverfahren

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

Zurück C

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 (einzen 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`

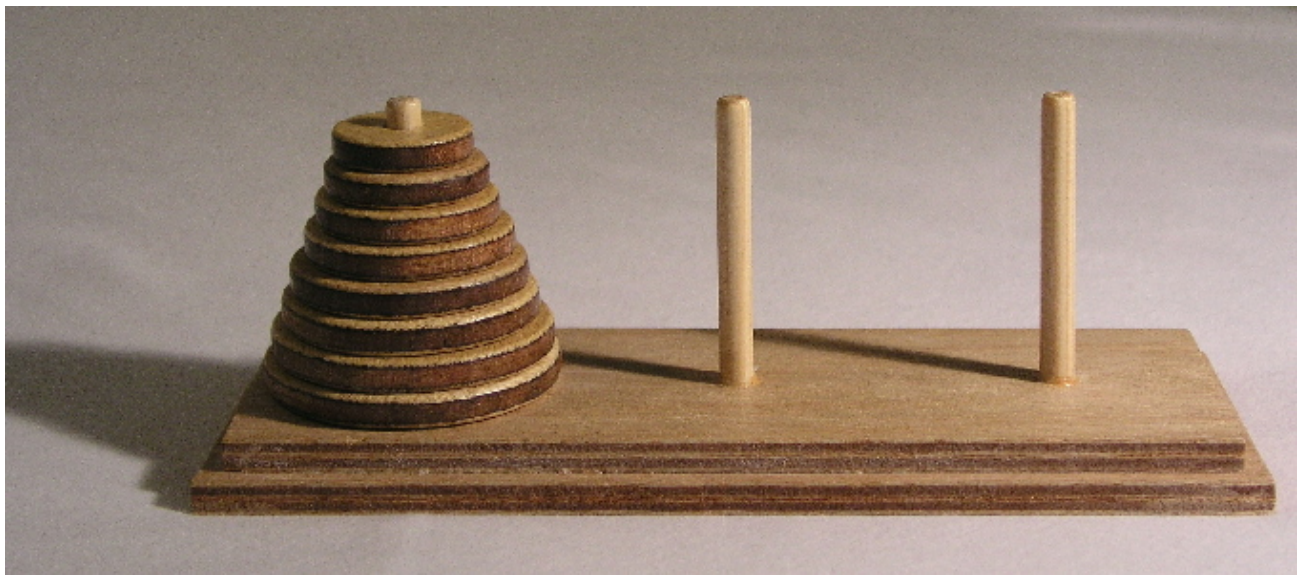
Alists - Schlüssel/Wert Paare

- *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
```

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

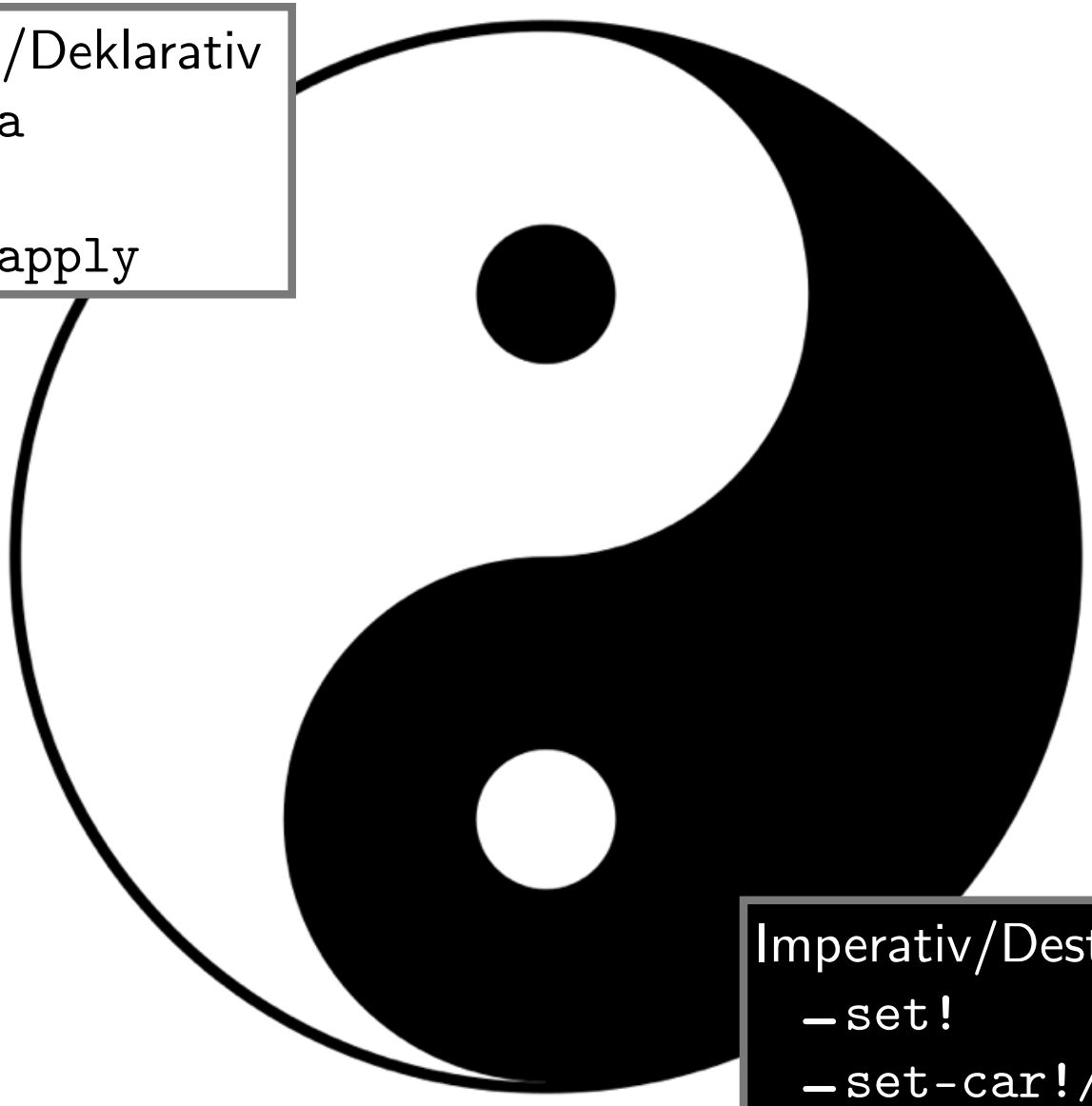
– Gruppen von ca. 3 Studierenden
– Entwurfsideen schriftlich (informell) festhalten

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

Zurück B Zurück C

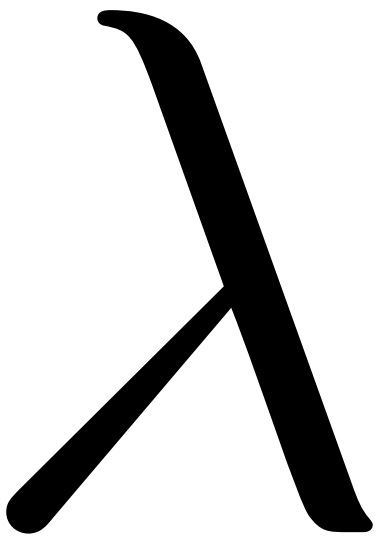
Die Helle und die Dunkle Seite der Macht

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:
  - Kreire 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 Variable (!)
- `lambda` kann `let` und (mit geistigem Strecken) `define` ersetzen:
  - `(let ((a init1) (b init2)) (mach-was a b))` ist äquivalent zu
  - `((lambda (a b) (mach-was a b)) init1 init2)`

Von Mengen zu funktionierenden Funktionen

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

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

> (define rel '((1 2) (2 4) (3 6)))
> (define fun (set->fun rel))
> (fun 3)
==> 6
> (fun 10)
==> #f
```

Für faule: `map`

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

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

==> (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 $f((\texttt{car}\, l_1), \ldots, (\texttt{car}\, l_2))$
- Beispiel:

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

==> (2 5 8 4)
```

Program oder Daten? apply

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

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

==> 32

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

==> (4 5 6)
```

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

Scheme in einem Befeh: `eval`

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

- Beispiele:

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

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

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 (interaction-environment 5))
==> 13
```

set-car!/set-cdr!

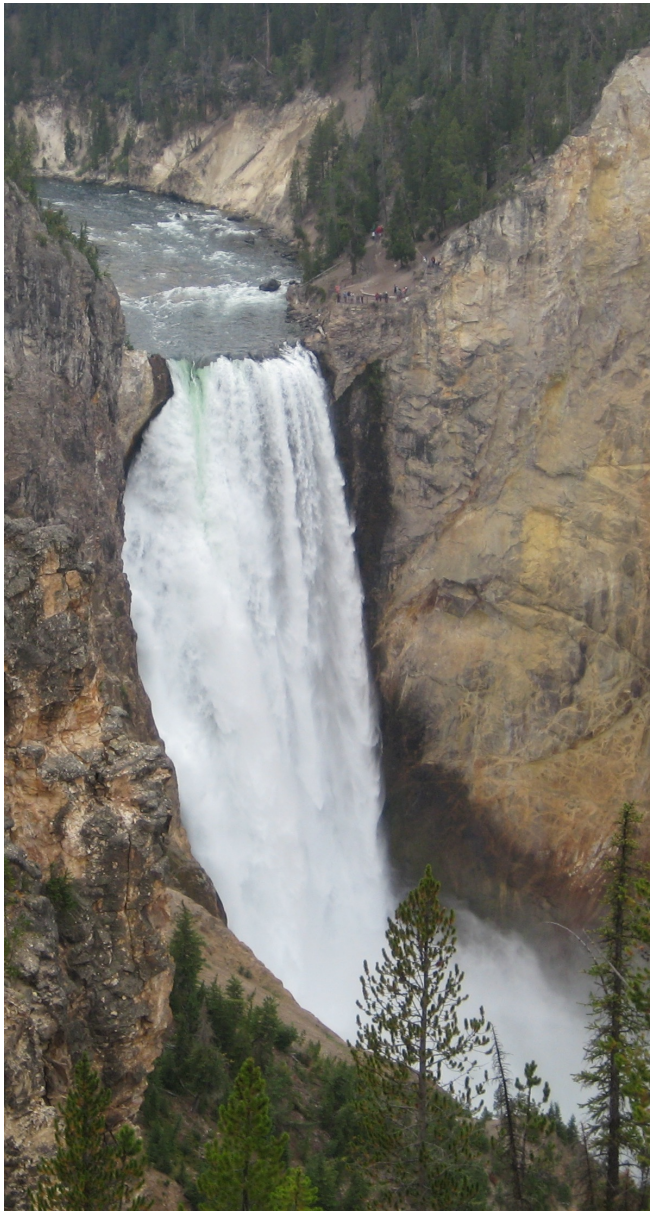
- `set-car!` verändert den ersten Wert eines existierenden `cons`-Paares
- `set-cdr!` verändert den zweiten Wert eines existierenden `cons`-Paares

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

Scheme Namenskonventionen

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

Das Typsystem von Scheme



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

Die Typen von Scheme

- Typprädikate (jedes Objekt hat genau einen dieser Typen):

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

Symbole als Werte

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

R5RS (6.3.3)

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

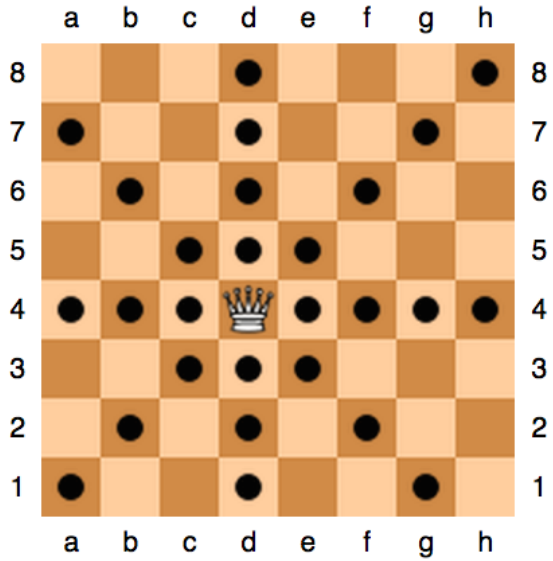
```
> (define a 'hallo)
> a
==> hallo
> (define l '(hallo dings bums))
> l
==> (hallo dings bums)
> (equal? (car l) a)
==> #t
```

Mut zur Lücke

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

Übung: Höfliche Damen

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

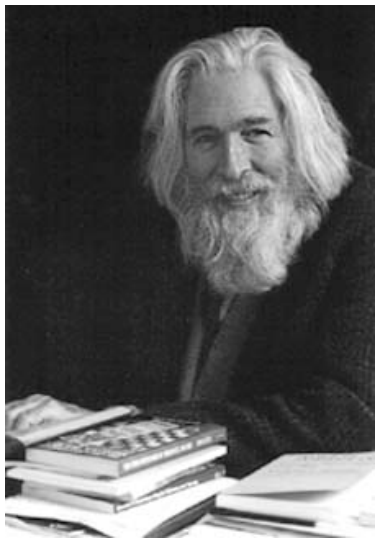


Zurück B

Zurück C

Aussagenlogik

Übung: Verbrechensaufklärung



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

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

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

Wen und warum?

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

Entscheidungshilfe

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

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

Formale Logik

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

Idee: Aussagenlogik

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: 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 $A, B \in For0_\Sigma$, dann sind auch
  $(\neg A), \quad (A \wedge B), \quad (A \vee B), \quad (A \to B), \quad (A \leftrightarrow B)$
  Elemente von $For0_\Sigma$

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

Übung: Syntax der Aussagenlogik

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

a) $A \to \bot$

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

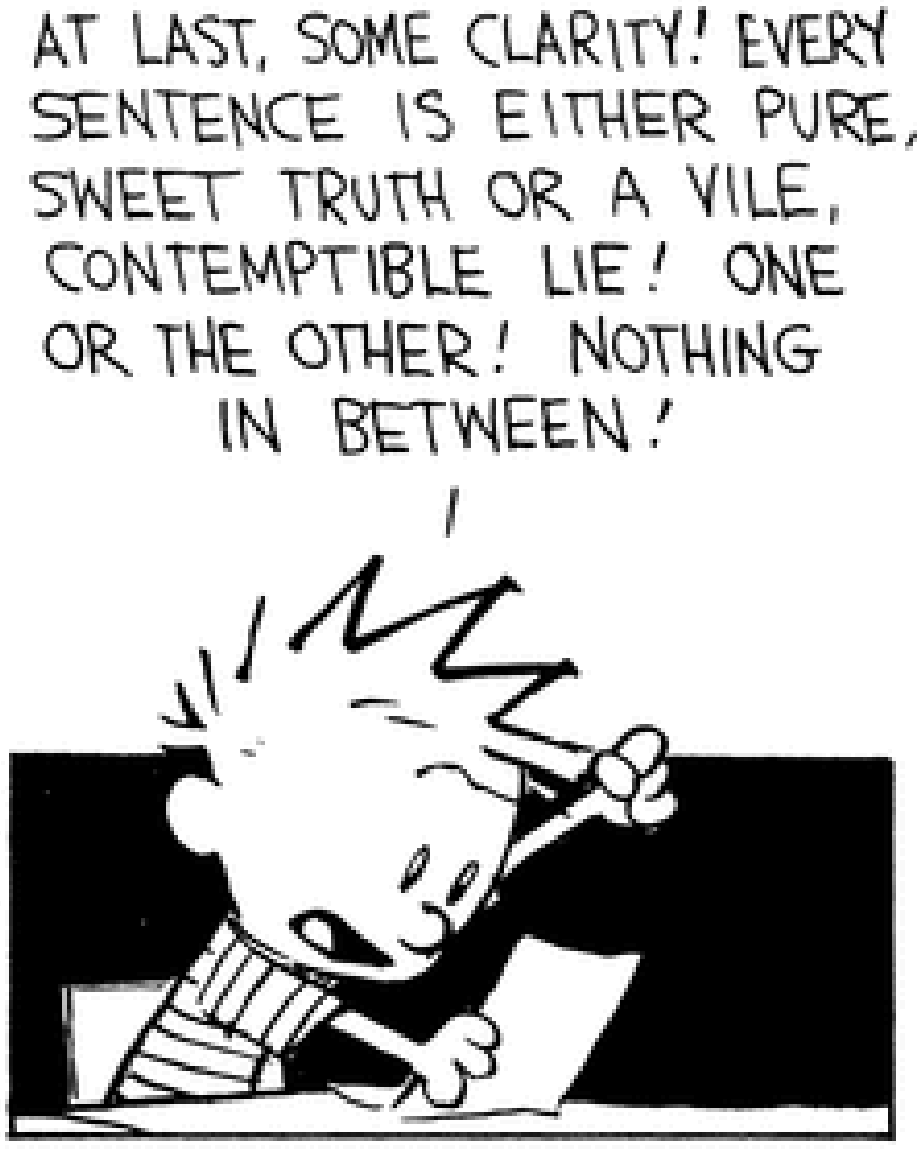
Operator	Präzedenz
$\neg$	1 (stärkste)
$\wedge$	2
$\vee$	3
$\rightarrow$	4
$\leftrightarrow$	5 (schwächste)

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

Sei Σ eine aussagenlogische Signatur

Definition: Aussagenlogische Interpretation

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

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

Tabellarisch:

A	B	C
1	1	0

Bei drei Atomen gibt es 8 mögliche Interpretationen

Semantik der Aussagenlogik (1)

Definition: Auswertung von Formeln unter einer Interpretation

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)

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)

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“

Modell einer Formel(menge)

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

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

Definition: 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 („verundung“) ihrer einzelnen Formeln

Erinnerung: Inspektor Craig

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

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

Zurück C

Tautologie

Definition: 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 ist unter allen Umständen wahr, unabhängig von der Belegung der Variablen
- Beispiele:
  - $\top$
  - $A \vee \neg A$
  - $A \to A$
  - $A \to (B \to A)$

Zurück B

Übung: Formalisierung von Raubüberfällen

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

1. Jeder der Männer A, B und C war am Tag des Geschehens in dem Laden gewesen, und kein anderer hatte den Laden an dem Tag betreten.
2. Wenn A schuldig ist, so hat er genau einen Komplizen.
3. Wenn B unschuldig ist, so ist auch C unschuldig.
4. Wenn genau zwei schuldig sind, dann ist A einer von ihnen.
5. Wenn C unschuldig ist, so ist auch B unschuldig.

- Beschreiben Sie die Ermittlungsergebnisse als logische Formeln.
- Lösen sie das Verbrechen!

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

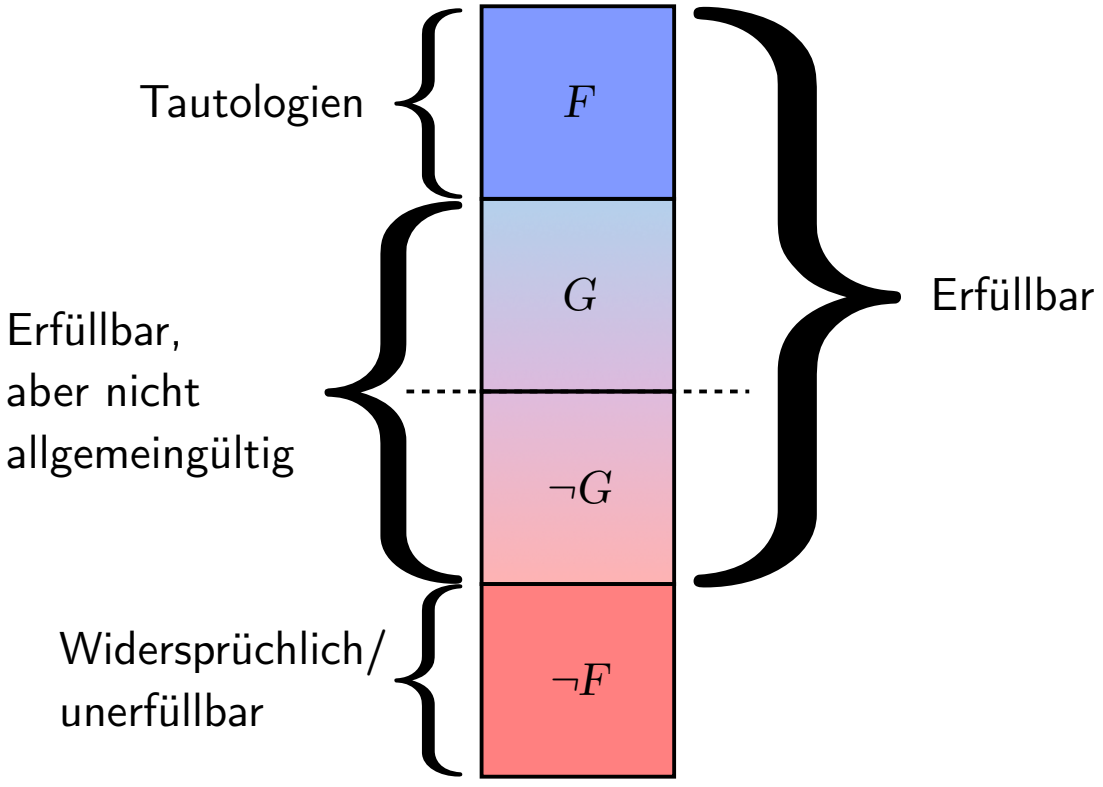
Unerfüllbarkeit

Definition: (Unerfüllbarkeit)

- Eine Formel $F \in For0_\Sigma$ heißt unerfüllbar, falls sie kein Modell hat, d.h falls $val_I(F) = 0$ 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.
Schreibweise: $KB \models A$

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

Die Wahrheitstafelmethode

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

Folgerungsproblem von Craig

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

Schritt 1: Aufzählung aller möglichen Welten

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

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

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

Vermutung 1: A ist schuldig!

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

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

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

Vermutung 2: B ist schuldig!

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

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 $KB \models A$
- In allen möglichen Welten, in denen die Annahmen gelten, ist B schuldig!

Komplexere Vermutung

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

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!

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

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

Unerfüllbarkeit

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

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

Sind wir mit der Aussagenlogik fertig?

Übung: Aussagenlogik in Scheme

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

Anwendungen von Aussagenlogik

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

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

Beobachtung

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

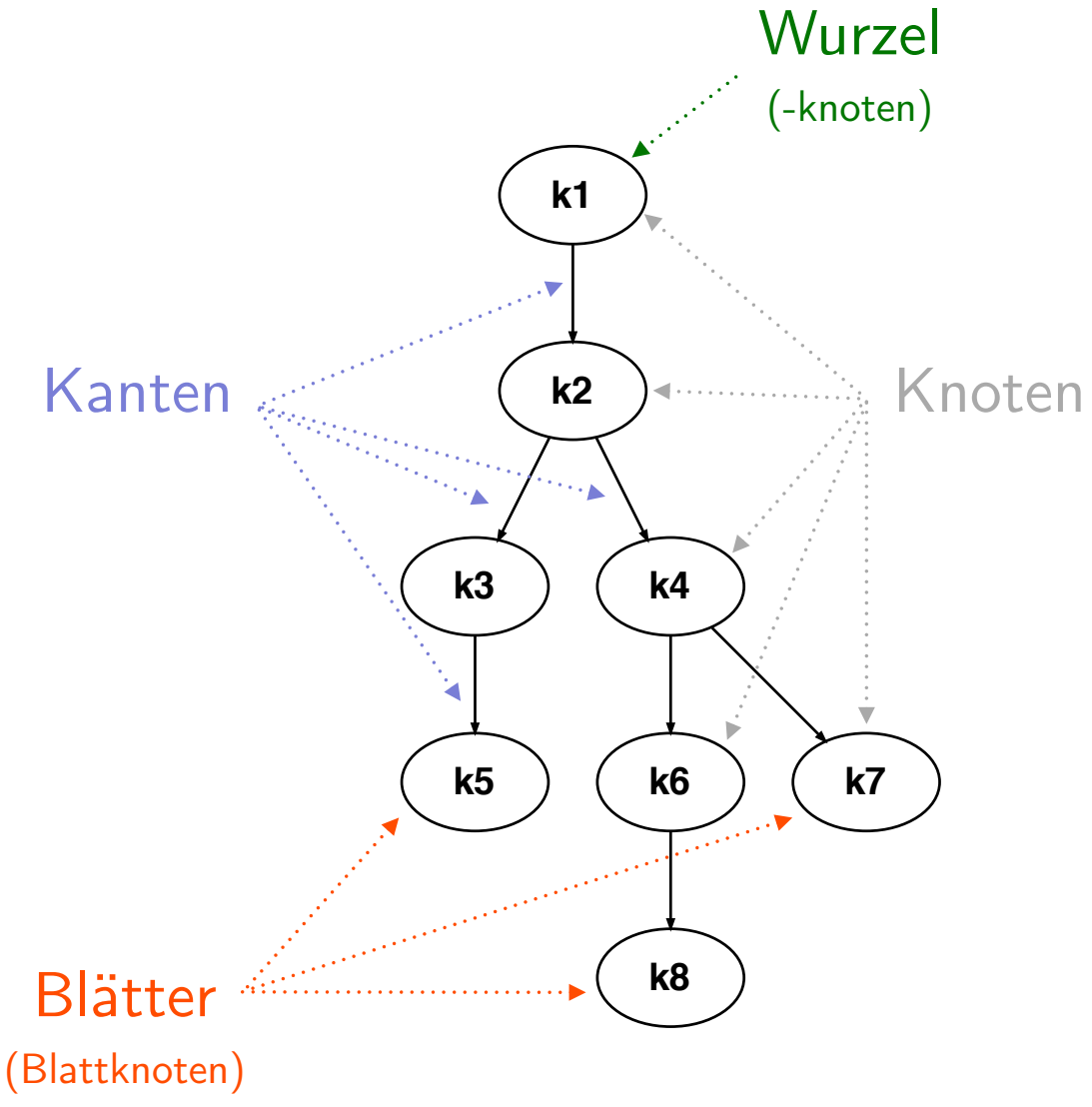
Idee: Zerlege eine Formel systematisch in Teilformeln, so dass die Erfüllbarkeit/Unerfüllbarkeit offensichtlich wird!

Der Tableaux-Kalkül 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“)

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 E.
  Dann ist das Tupel (V, E) ein (ungeordneter) Baum, wenn folgende Eigenschaften gelten:
  - Es existiert ein ausgezeichnetes Element $r \in V$ (die Wurzel), so dass kein Knoten $x \in V$ mit $(x, r) \in E$ 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.“

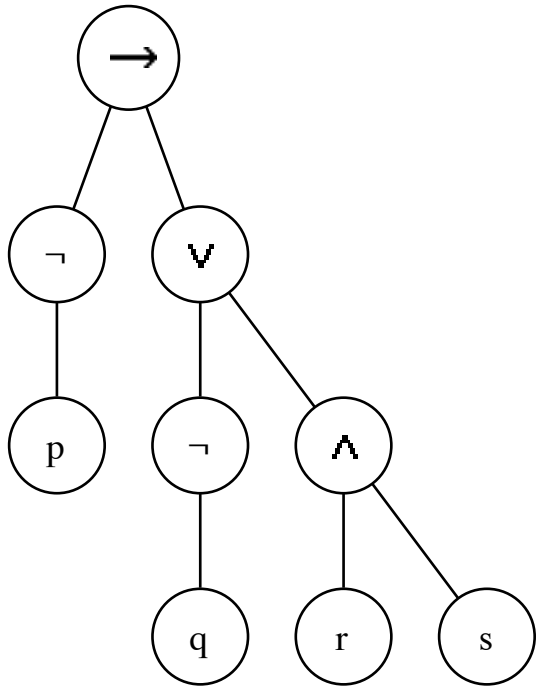
Bäume (2)

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.

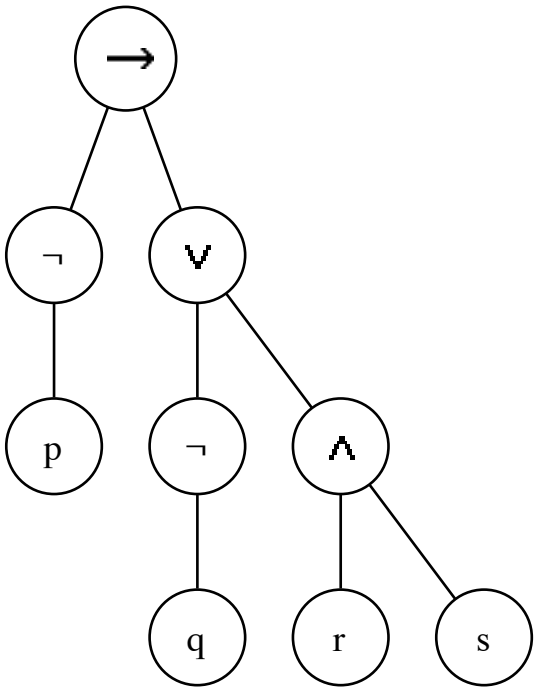
Bäume (3)

- 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.



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

Übung: Bäume



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

- 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?

Zurück C

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...

Uniforme Notation: Idee

Fallunterscheidung nach Formeltyp

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

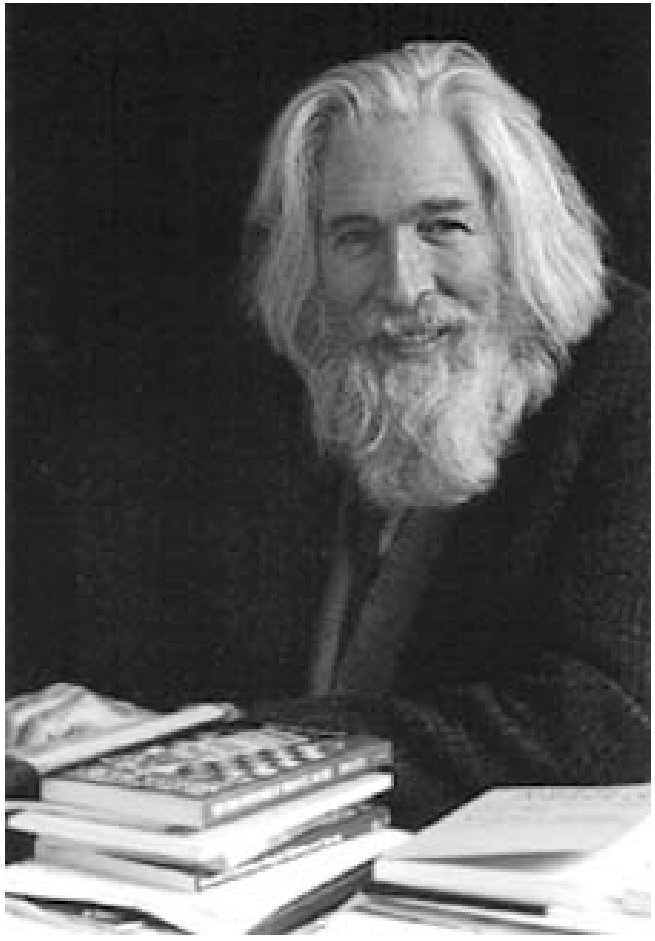
Uniforme Notation: α und β

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

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
    $$\begin{array}{c} | \\ \alpha_1 \\ | \\ \alpha_2 \end{array}$$
- 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
    $$\begin{array}{ccc} & / & \backslash \\ \beta_1 & & \beta_2 \end{array}$$
- Schluss-Regel: Ein Ast, auf dem eine Formel und ihre Negation vorkommt, kann als geschlossen markiert werden.

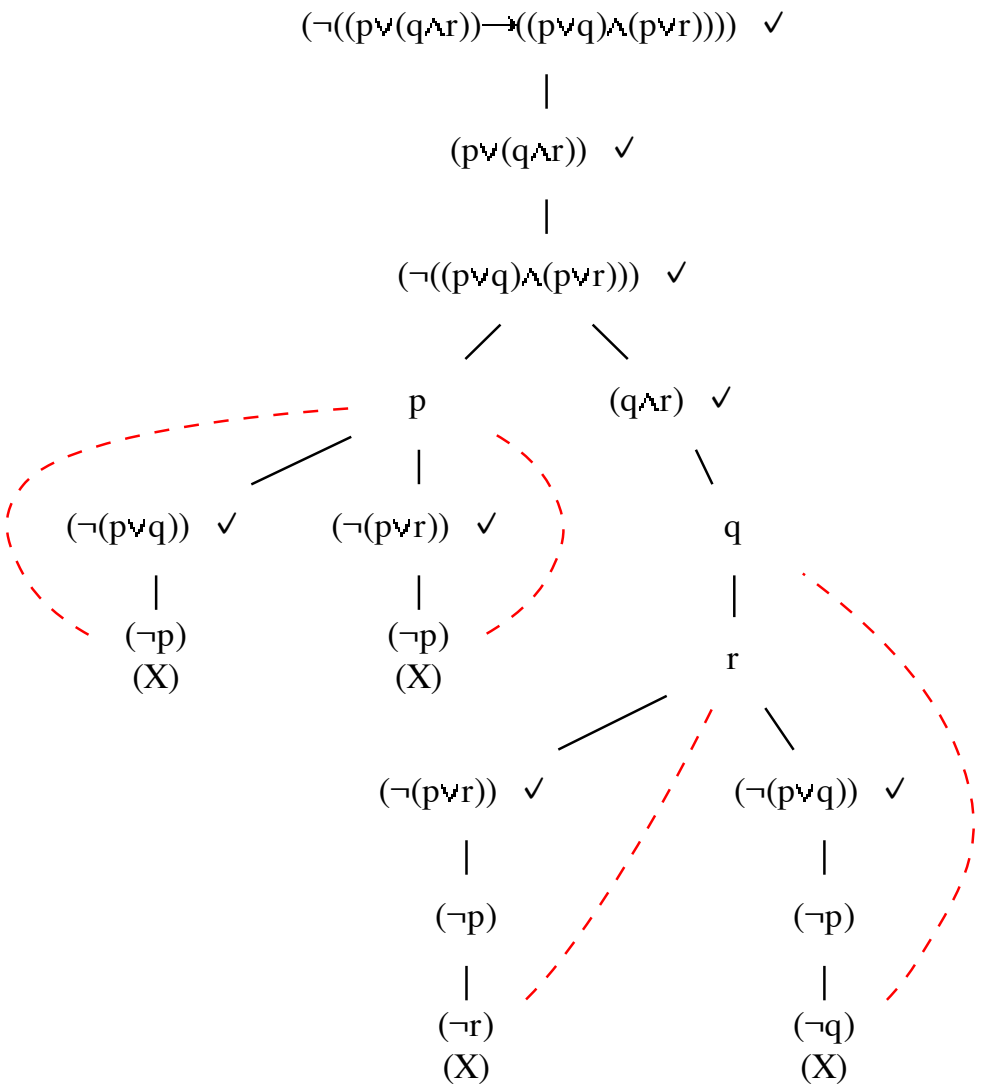
Tableaux und Formeln

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

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

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-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 gschlossen 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?

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?

Zurück C

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

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

Lemma (Fortsetzung)

- Induktionsschritt: Gehe T_{i+1} aus T_i durch Anwendung einer Tableaux-Regel hervor.
- Fallunterscheidung:
  1) M wird durch die Regel nicht erweitert. Dann ist M ein Ast in T', und $I(M) = 1$.
  2) M wird durch die Regel erweitert. Der expandierte Knoten sei A. Fallunterscheidung:
     a) 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 existiert der gesuchte Ast.

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

Korrektheit des Tableaux-Kalküls

Satz: Geschlossene Tableaux

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

Beweis: Per Widerspruch

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

q.e.d.

Tableaux und Modelle

Satz (Tableaux-Modelle):
Sei F eine aussagenlogische Formel, T ein vollständiges Tableau, und M ein offener Ast in T. Dann beschreiben die Literale in M eine erfüllende Interpretation von F, d.h. jede Interpretation, die die Literale in T wahr macht, ist auch ein Modell von F.

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

Übung: Modellextraktion

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

Vollständigkeit und Korrektheit

Satz (Vollständigkeit und Korrektheit des Tableaux-Kalküls)
Sei $F \in For0_\Sigma$ eine aussagenlogische Formel und T ein vollständiges Tableau für F. Dann gilt:

- Wenn T offen ist, so ist F erfüllbar
- Wenn T geschlossen ist, so ist F unerfüllbar

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

Umsetzung des Tableaux-Kalküls

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

Übung: Mehr Tableaux

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

Übung: Tableaux für Inspektor Craig

Craig 1:

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

Craig 2:

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

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

Widerspruchskalküle

- Viele praktische Kalküle zeigen die Unerfüllbarkeit einer Formel(-Menge):
  - Zu zeigen: Eine Hypothese H folgt aus einer Menge von Formeln $KB = \{F_1, F_2, \ldots, F_n\}$:
    $$KB \models H$$
  - 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:
    $$F_1 \wedge F_2 \wedge \ldots \wedge F_n \wedge \neg H \text{ unerfüllbar}$$

Im Tableauxkalkül: Herleitung eines geschlossenen Tableaus

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

Zurück B

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:

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

Beweisprinzip: Strukturelle Induktion

- Die strukturelle Induktion oder Induktion über den Aufbau kann verwendet werden, um Eigenschaften von rekursiv definierten Objekten zu zeigen.
- Idee:
  - Zeige die Eigenschaft für die elementaren Objekte.
  - Zeige die Eigenschaft für die zusammengesetzten Objekte unter der Annahme, dass die Teile bereits die Eigenschaft haben
- Konkret für aussagenlogische Formeln:
  - Basisfälle sind die Aussagenlogischen Variablen oder $\top$, $\bot$.
  - 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 (wahlweise):
  - $\{\wedge, \vee, \neg\}$ ist eine Basis
  - $\{\rightarrow, \neg\}$ ist eine Basis
  - $\{\wedge, \neg\}$ ist eine Basis

Zurück C

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

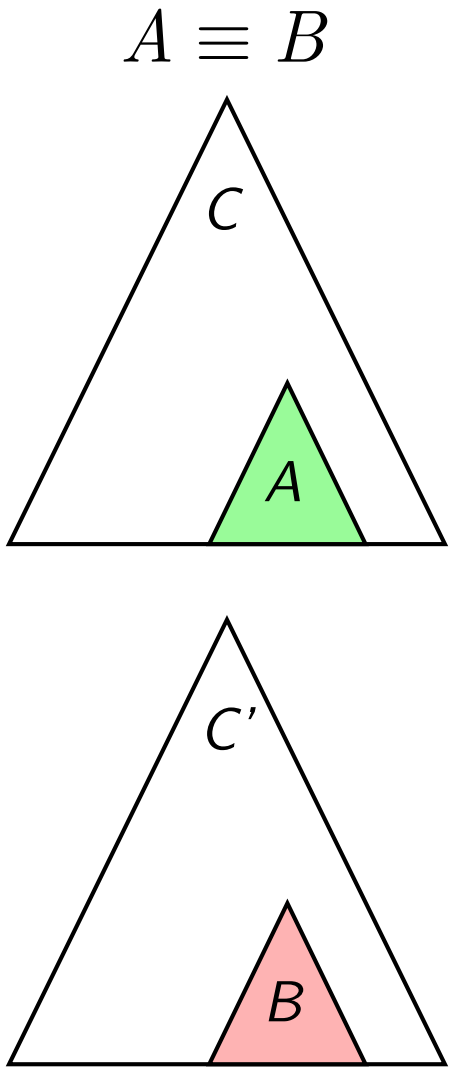
- Beispiel:
  - $TF((a \vee (\neg b \leftrightarrow c))) = \{(a \vee (\neg b \leftrightarrow c)), a, (\neg b \leftrightarrow c), \neg b, b, c\}$
  - $TF(\neg\neg\neg a) = \{\neg\neg\neg a, \neg\neg a, \neg a, a\}$

Zurück C

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

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

$(A \wedge B)$	$\equiv$	$(B \wedge A)$	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
$(A \rightarrow B)$	$\equiv$	$(\neg B \rightarrow \neg A)$	Kontraposition
$(A \rightarrow B)$	$\equiv$	$(\neg A \vee B)$	Elimination Implikation
$(A \leftrightarrow B)$	$\equiv$	$((A \rightarrow B)) \wedge (B \rightarrow A))$	Elimination Äquivalenz
$\neg(A \wedge B)$	$\equiv$	$(\neg A \vee \neg B)$	De-Morgans Regeln
$\neg(A \vee B)$	$\equiv$	$(\neg A \wedge \neg B)$	De-Morgans Regeln
$(A \wedge (B \vee C))$	$\equiv$	$((A \wedge B) \vee (A \wedge C))$	Distributivität von $\wedge$ über
$(A \vee (B \wedge C))$	$\equiv$	$((A \vee B) \wedge (A \vee C))$	Distributivität von $\vee$ über

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

- $(A \wedge \neg A) \equiv \bot$
- $(A \vee \neg A) \equiv \top$ Tertium non datur
- $(A \wedge \top) \equiv A$
- $(A \wedge \bot) \equiv \bot$
- $(A \vee \top) \equiv \top$
- $(A \vee \bot) \equiv A$
- $(\neg\top) \equiv \bot$
- $(\neg\bot) \equiv \top$

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:

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

Übung: Logische Umformungen

Zeigen Sie:

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

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

Normalformen

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

- 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 Konkunktion)

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 $\wedge$

Umformung in KNF: Beispiel

- Ausgangsformel
  $p \leftrightarrow (q \vee r)$
- 1. Elimination von $\leftrightarrow$
  $(p \rightarrow (q \vee r)) \ \wedge \ ((q \vee r) \rightarrow p)$
- 2. Elimination von $\rightarrow$
  $(\neg p \vee q \vee r) \ \wedge \ (\neg(q \vee r) \vee p)$
- 3. Nach innen schieben von $\neg$
  $(\neg p \vee q \vee r) \ \wedge \ ((\neg q \wedge \neg r) \vee p)$
- 4. Nach innen schieben von $\vee$
  $(\neg p \vee q \vee r) \ \wedge \ (\neg q \vee p) \ \wedge \ (\neg r \vee p)$

Zurück B

Übung: KNF Transformation

Transformieren Sie die folgenden Formeln in konjunktive Normalform:

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

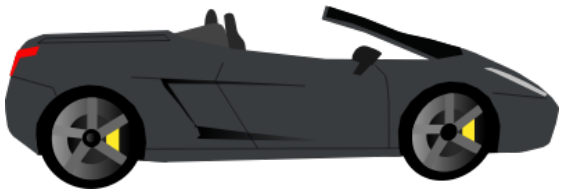
Tipps:

- Wenn die Formel auf der obersten Ebene bereits eine Konjunktion („und“-verknüpft) ist, kann mann die beiden Teilformeln einzeln transformieren.
- Wenn eine Formel eine Disjunktion ist, und in den Unterformeln noch $\wedge$ vorkommt, dann wendet man $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$ in dieser Richtung an. A kann dabei durchaus eine komplexe Formel sein.

Übung: Formalisieren in KNF

Konfiguration von Fahrzeugen:

- Ein Cabrio hat kein Schiebedach
- Ein Cabrio hat keinen Subwoover
- 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: Subwoover
  - d: Doppelauspuff
  - q: Quattro
  - v: Hoher Verbrauch
- In Klauselform:
  1. $\neg c \vee \neg s$
  2. $\neg c \vee \neg w$
  3. $\neg d \vee q \vee c$
  4. $\neg q \vee v$
  5. $\neg v$
  6. s
  7. d

Erfüllbarkeit naiv: $2^6 = 64$ Interpretationen

Resolution

=

Verfahren zum Nachweis der Unerfüllbarkeit einer Klauselmengea

Klauseln mal anders

Definition (Klauseln als Menge): Eine Klausel ist eine Menge von Literalen

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

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

Interpretationen mal anders

Definition (Interpretationen als Literalmengen):

- Eine Interpretation ist eine Menge von Literalen mit:
  - Für jedes $p \in P$ ist entweder $p \in I$ oder $\neg p \in I$
  - Für kein $p \in P$ ist $p \in I$ und $\neg p \in I$
- Eine Klausel C ist wahr unter I, falls eines ihrer Literale in I vorkommt
- Formal: $I(C) = \begin{cases} 1 & \text{falls } C \cap I \neq \emptyset \\ 0 & \text{sonst} \end{cases}$

Auch hier: Diese Definition steht in enger Beziehung zur bekannten:

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

Semantik: Beispiele

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

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

Semantik von Klauselmengen

- Die Elemente einer Menge von Klauseln sind implizit und-verknüpft
- Eine Interpretation I erfüllt eine Menge von Klauseln S, falls sie jede Klausel in S wahr macht
- Formal:

$$I(S) = \begin{cases} 1 & \text{falls } I(C) = 1 \text{für alle} C \in S \\ 0 & \text{sonst} \end{cases}$$

- Die Begriffe Modell, erfüllbar und unerfüllbar werden 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 $\square = \{\}$ enhält 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

$$\frac{C \vee p \qquad D \vee \neg p}{C \vee D}$$

Mengenschreibweise

$$\frac{C \uplus \{p\} \qquad D \uplus \{\neg p\}}{C \cup D}$$

- 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

Automobile in Klauselform

- Elementare Aussagen:
  - c: Cabrio
  - s: Schiebedach
  - w: Subwoover
  - d: Doppelauspuff
  - q: Quattro
  - v: Hoher Verbrauch

1. $\neg c \vee \neg s$
2. $\neg c \vee \neg w$
3. $\neg d \vee q \vee c$
4. $\neg q \vee v$
5. $\neg v$
6. s
7. d

- Mögliche Resolventen:

8. $\neg s \vee \neg d \vee q$ (aus 1 und 3)
9. $\neg c$ (aus 1 und 6)
10. $\neg q$ (aus 4 und 5)
11. $\neg d \vee q$ (aus 3 und 9)
12. $\neg d$ (aus 11 und 10)
13. $\square$ (7 und 12)

Pech gehabt! Ich fahre weiter Fahrrad!

Übung: Resolution von Craig 2

Aussagen des Ladenbesitzers:

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

Konvertieren Sie die Aussagen in KNF und zeigen Sie per Resolution die Unerfüllbarkeit.

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

Zurück C

Übung: Resolution von Jane

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

Konvertieren Sie das Folgerungsproblem in ein KNF-Problem und zeigen Sie per Resolution die Unerfüllbarkeit!

Prädikatenlogik

Grenzen der Aussagenlogik

Alle Menschen sind sterblich
Sokrates ist ein Mensch

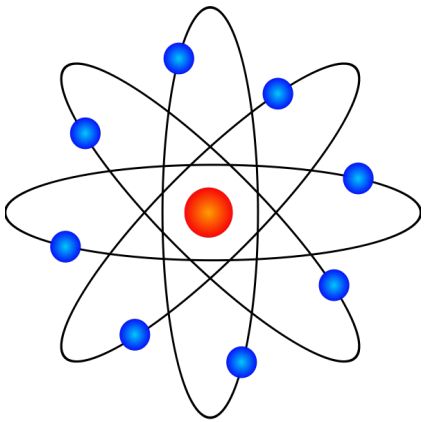
Also ist Sokrates sterblich

Alle Vögel können fliegen
Ein Pinguin ist ein Vogel

Also kann ein Pinguin fliegen

Prädikatenlogik erster Stufe

- Logik von Relationen und Funktionen über einem Universum
- Idee: Atome sind nicht mehr atomar
- Prädikatssybole repräsentieren Relationen
- Funktionssymbole repräsentieren Funktionen
- Variablen stehen beliebige Objekte (Universelle und existentielle Quantifizierung)



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

- Eine Signatur Σ ist ein 3-Tupel (P, F, V)
- P: Menge der Prädikatssymbole mit Stelligkeit
  - n-stellige Prädikatssymbole repräsentieren n-stellige Relationen
  - Z.B. $P = \{\mathit{mensch}/1, \mathit{sterblich}/1, \mathit{lauter}/2\}$
- 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!
- 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 $\{x, y, z, u, v, x_0, x_1, \ldots\}$

Terme

- Gegeben: $\Sigma = (P, F, V)$
- Definition: T_Σ ist die Menge der Terme über F, V
  - Sei $X \in V$. Dann $X \in T_\Sigma$
  - Sei $f/n \in F$ und seien $t_1, \ldots, t_n \in T_\Sigma$. 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)
    - $gruppe(X, sokrates)$
    - $guppe(lehrer(sokrates), gruppe(sokrates, Y))$

Terme

Bemerkungen

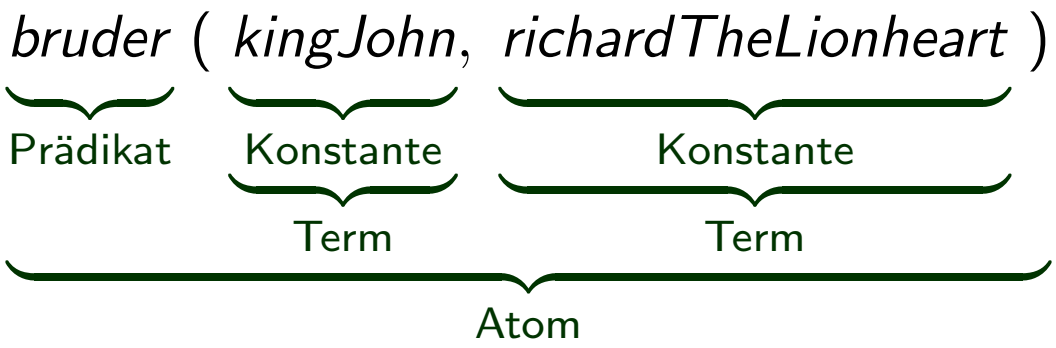
- Insbesondere sind alle Konstanten in T_Σ
  - Wir schreiben in der Regel a, 1, statt $a(), 1()$
- Terme ohne Variablen heißen Grundterme oder einfach grund
- Terme bezeichnen (Mengen von) Elementen des Universums
- Grundterme bezeichnen einfache Elemente

Atome

- Gegeben: $\Sigma = (P, F, V)$
- Definition: A_Σ ist die Menge der Atome über P, F, V
  - 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:
  - $P = \{\mathit{mensch}/1, \mathit{sterblich}/1, \mathit{lauter}/2\}$,
    $F = \{\mathit{gruppe}/2, \mathit{lehrer}/1, \mathit{sokrates}/0, \mathit{aristoteles}/0\}$
- Atome:
  - $\mathit{mensch}(X)$
  - $\mathit{lauter}(\mathit{sokrates}, X)$
  - $\mathit{lauter}(\mathit{sokrates}, \mathit{sokrates})$
  - $\mathit{lauter}(\mathit{sokrates}, \mathit{lehrer}(\mathit{lehrer}(\mathit{lehrer}(X))))$

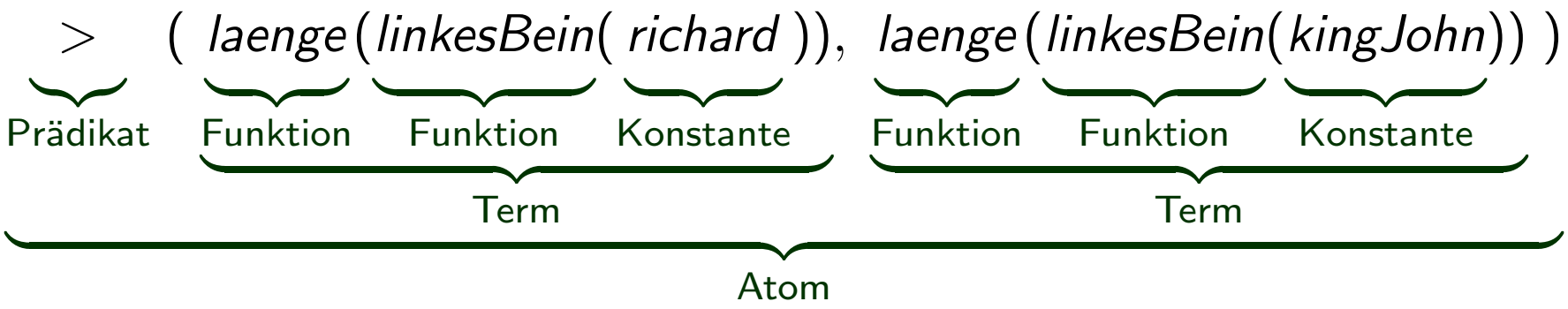
Atome

Beispiel



Syntax der Prädikatenlogik: Atomare Formeln

Beispiel



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: Anmerkungen zum Vokabular

- Wir verlangen, dass V, P, F disjunkt sind
- Funktions- und Prädikatssymbole haben eine Stelligkeit $n \geq 0$
  - Schreibweise manchmal auch: $f|_n$ oder $\mathit{arity}(f) = n$
  - Oft implizit ersichtlich aus der Verwendung
- Funktionsssymbole mit Stelligkeit $n = 0$ heißen Konstanten
  z.B. 2, $\mathit{stuttgart}$, c

Zurück B

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

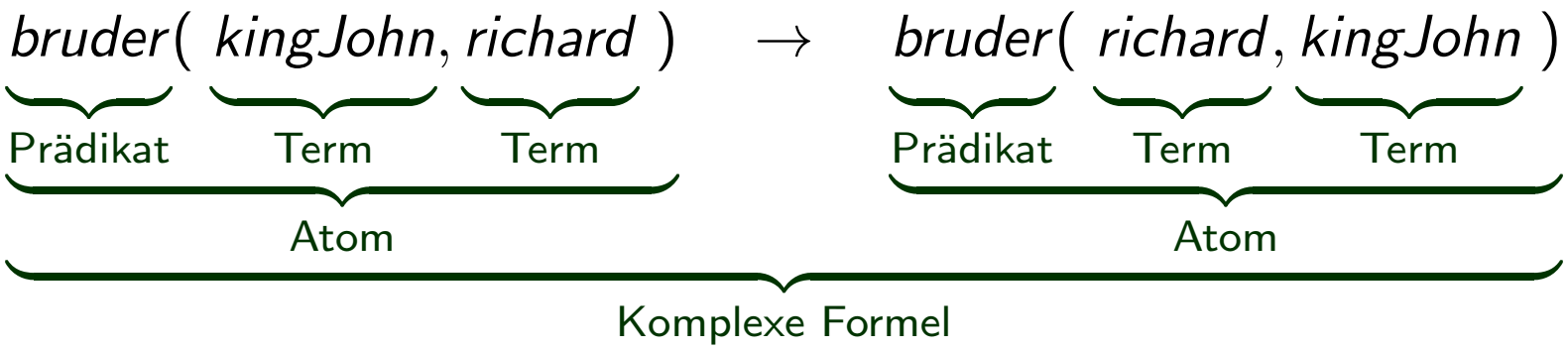
Definition (Formeln der Prädikatenlogik 1. Stufe): Sei $\Sigma = (P, F, V)$ eine prädikatenlogische Signatur.

For_Σ ist die kleinste Menge mit:

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

Syntax der Prädikatenlogik: Komplexe Formeln

Beispiel



Übung: Formeln der Prädikatenlogik 1. Stufe

Formalisieren Sie:

- „Alle, die in Stuttgart studieren, sind schlau“
- „Es gibt jemand, der in Mannheim studiert und schlau ist“
- „Die Summe zweier Primzahlen ist eine Primzahl“

Gottlob Frege (1879): *„Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens“*

Freie und gebundene Variablen

Definition (Freie / gebundene Variable)

Ein Vorkommen einer Variablen X heißt

- gebunden, wenn sie im Bereich (Skopus) einer Quantifizierung $\forall X$ / $\exists X$ ist
- frei sonst

Beispiel

$$p(z) \to \forall x\ (q(x,z) \wedge \exists z\ r(y,z))$$

- 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 $>, +, *, \leq, \ldots$

Semantik?

Semantik der Prädikatenlogik

Definition (Prädikatenlogische Interpretation):
Eine (prädikatenlogische) Interpretation ist ein Paar $\langle U, A \rangle$, wobei

U: ist eine nicht-leere Menge (Universum)

I: ist eine Interpretationsfunktion – sie interpretiert

- (freie) Variablen: durch ein Element des Universums
- Prädikate: durch eine Relation auf dem Universum (mit passender Stelligkeit)
- Funktionen: durch eine Funktion auf dem Universum (mit passender Stelligkeit)

Semantik der Prädikatenlogik

Bemerkungen

- Im allgemeinen ist das Universum U einen prädikatenlogischen Modells unendlich
- Auch schon für ein endliches U gibt es eine riesige Zahl verschiedener Interpretationen

Semantik der Prädikatenlogik

Notation

Seit $\langle U, A \rangle$ eine prädikatenlogische Interpretation und $s \in P \cup F \cup V$. Dann sei:

$$s^I = I(s)$$

Also: s^I bezeichnet $I(s)$, die Interpretation des Prädikats-, Funktions- oder Variablensymbols s unter der gegebenen Interpretation

Semantik der Prädikatenlogik (0)

Definition (Semantik eines Terms t):
Sei eine prädikatenlogische Interpretation $\langle U, I \rangle$ gegeben.

Die Semantik von $t \in Term_\Sigma$ ist das Element $I(t)$ aus U, das rekursiv definiert ist durch

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

Semantik der Prädikatenlogik (1)

Definition (Semantik einer Formel)

Sei eine prädikatenlogische Interpretation $\langle U, I \rangle$ gegeben.

Die Semantik $I(F)$ einer Formel F unter I ist einer der Wahrheitswerte 1 oder 0

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

Semantik der Prädikatenlogik (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)

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)

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)

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 mit I mit der Ausnahme, dass $x^{I_{x/d}} = d$

Übung: Auswertung von Formeln

Betrachten Sie die Formel $F = \forall X(\exists Y(gt(X, plus(Y, 1))$

- Sei $U = \{T, F\}$ und $I(gt) = \{(T, F)\}, I(plus) = \{((F, F), F), ((F, T), T), ((T, F), T), ((T, T), T)\}, I(1) = T$. Bestimmen Sie $I(F)$.
- Sei $U = \mathbb{N}$ und $I(gt) =>, I(plus) = +, I(I) = 1$. Bestimmen Sie $I(F)$.

Eigenschaften von Quantoren

Quantoren gleicher Art kommutieren

$\forall x \forall y$	ist äquivalent zu	$\forall y \forall x$
$\exists x \exists y$	ist das gleiche wie	$\exists y \exists x$

Eigenschaften von Quantoren

Verschiedene Quantoren kommutieren NICHT

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

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
$(\forall x\ \mathit{eiskrem}(x)) \vee (\forall x\ \mathit{broccoli}(x))$

Eigenschaften von Quantoren

$\exists$ distributiert NICHT über $\wedge$

$\exists x\,(\ldots \wedge \ldots)$ ist NICHT äquivalent zu $(\exists x \ldots) \wedge (\exists x \ldots)$

Beispiel

$\exists x\,(\mathit{gerade}(x) \wedge \mathit{ungerade}(x))$ ist NICHT äquivalent zu
$(\exists x\ \mathit{gerade}(x)) \wedge (\exists x\ \mathit{ungerade}(x))$

Beispiele: Familienverhältnisse

- „Brüder sind Geschwister“
  $\forall x \forall y\ (\mathit{bruder}(x, y) \rightarrow \mathit{geschwister}(x, y))$
- „bruder“ ist symmetrisch
  $\forall x \forall y\ (\mathit{bruder}(x, y) \leftrightarrow \mathit{bruder}(y, x))$
- „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“
  $\forall x \forall y\ (\mathit{cousin}1(x, y) \leftrightarrow$
  $\exists p \exists ps\ (\mathit{elter}(p, x) \wedge \mathit{geschwister}(ps, p) \wedge \mathit{elter}(ps, y)))$

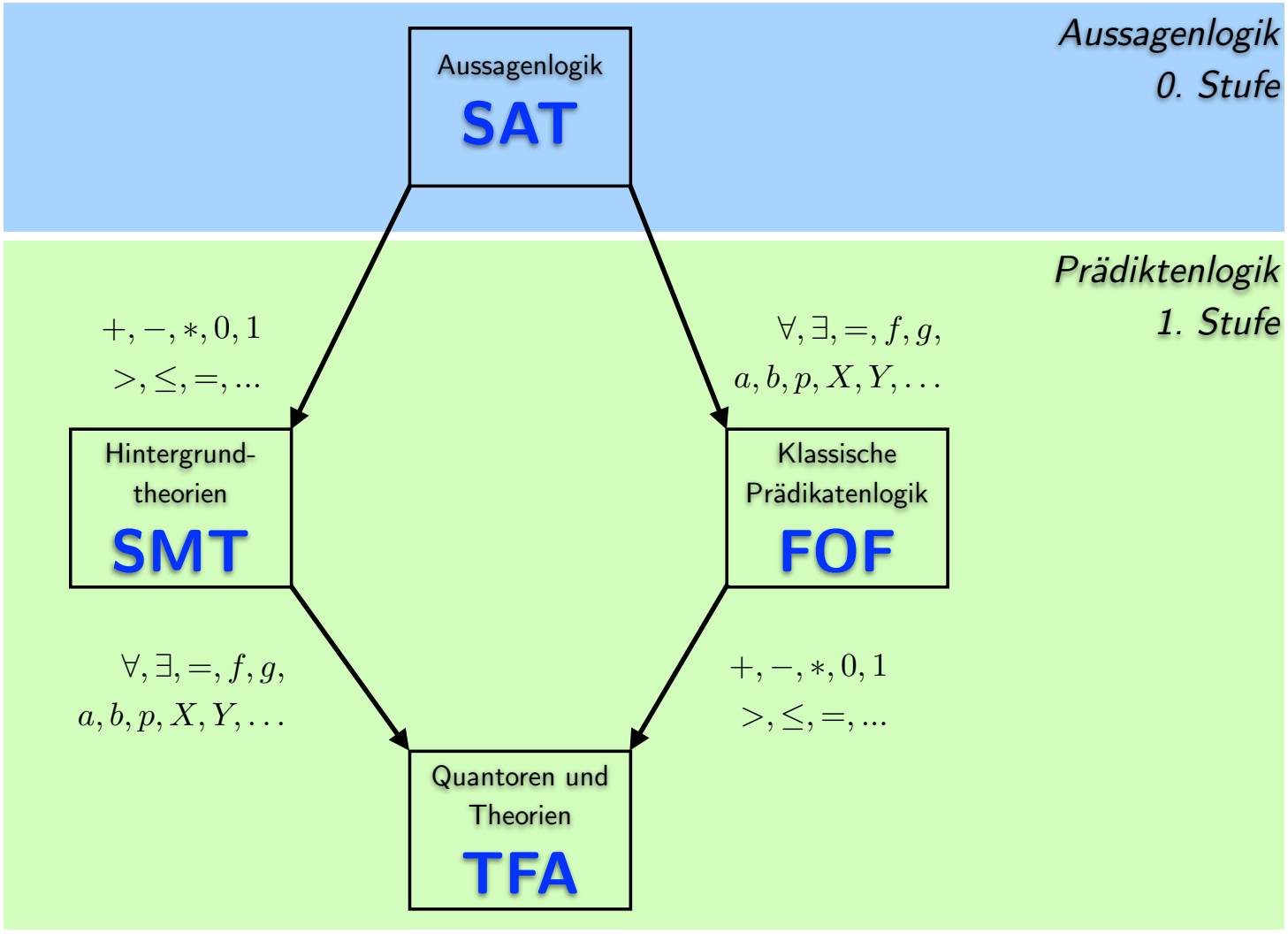
Wichtige Begriffe für Prädikatenlogik

Definition:

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

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

Prädikatenlogik im Kontext



Übung: Primzahlen

- Formalisieren Sie das Konzept *Primzahl* und die Aussage „Zu jeder Primzahl gibt es eine größere Primzahl“.
- Beschreiben Sie ein geeignetes Modell für Ihre Formel(n)
- Geben Sie eine zweite Interpretation an, in der die Aussage „Zu jeder Primzahl gibt es eine größere Primzahl“ falsch ist.

Prädikatenlogik: Folgerung und Unerfüllbarkeit

- Wir betrachten nur geschlossene Formeln (bei denen alle Variablen gebunden sind)
  - Seien $F_1, \ldots, F_n, G \in For_\Sigma$
  - 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 Unerfüllbarkeit zeigen?

Prädikatenlogik: Negationsnormalform

- NNF: Wie in Aussagenlogik
  - Nur $\{\neg, \vee, \wedge\}$ (aber $\forall, \exists$ sind erlaubt)
  - $\neg$ nur direkt vor Atomen
- NNF Transformation:
  - Elimination von $\leftrightarrow, \rightarrow$ wie bei Aussagenlogik
  - Nach-innen-schieben von $\neg$
    - De-Morgan
    - $\neg(\forall x F) \equiv \exists x(\neg F)$
    - $\neg(\exists x F) \equiv \forall x(\neg F)$

Elimination von $\exists$: Skolemisierung

- Idee: Betrachte Formel der Form $\exists y F$
  - Wenn $\exists y F$ erfüllbar ist, dann gibt es eine Interpretation und (mindestens) einen Wert für y, mit der F wahr wird.
  - Wir können die Variable durch eine neue Konstante ersetzen und die Interpretation so anpassen, dass sie diese Konstante durch diesen Wert interpretiert.
- Idee: Betrachte $\forall x \exists y F$
  - Wenn ein solches y existiert, dann hängt y nur von x ab
  - Also: y kann durch $f(x)$ ersetzt werden, wobei f ein neues Funktionssymbol ist (und geeignet interpretiert wird).

Thoralf Skolem (1887–1963)

Ergebnis: Formel ohne Existenzquantor

Prädikatenlogische Literale

Definition (Literale): Sei $\Sigma = (P, F, V)$ eine prädikatenlogische Signatur und A_Σ die Menge von Atomen über Σ.
Sei $a \in A_\Sigma$. Dann sind a, $\neg a$ Literale.

Prädikatenlogische Klauseln

Definition: Eine Klausel ist eine Menge von Literalen.

- Beispiele
  - $C_1 = \{\neg mensch(X), sterblich(X)\}$
  - $C_2 = \{\neg lauter(sokrates, sokrates)\}$
  - $C_3 = \{\neg lauter(X, sokrates), \neg sterblich(X)\}$
- Literale einer Klausel sind (implizit) oder-verknüpft
- Schreibweise
  - $C_1 = \neg mensch(X) \vee sterblich(X)$
  - $C_2 = \neg lauter(sokrates, sokrates)$
  - $C_3 = \neg lauter(X, sokrates) \vee \neg sterblich(X)$
- Alle Variablen in Klauseln sind implizit $\forall$-quantifiziert

Klauselnormalform für Prädikatenlogik

Definition (Klauselnormalform):
Eine Formel in Klauselnormalform ist eine Menge von Klauseln.

- Die Klauseln werden als implizit „und“-verknüpft betrachtet
- Skolemisierte Formeln in NNF können durch Äquivalenzumformungen in KNF gebracht werden

Gesucht: Verfahren, die Unerfüllbarkeit einer Formel in Klauselnormalform zu zeigen

Herbrand-Universum (Jacques Herbrand (1908-1931))

Definition (Herbrand-Universum): Sei $\Sigma = (P, F, V)$ eine Signatur und sei $a/0 \in F$. Das Herbrand-Universum zu Σ ist die Menge aller Grundterme über F, also die Menge aller Terme über $F, \{\}$.

- Das Herbrand-Universum besteht aus allen variablenfreien Termen, die aus der Signatur gebildet werden können.
- Falls die Signatur keine Konstante enthält, so fügen wir eine beliebige Konstante hinzu.

Satz: Eine Formeln in Klauselnormalform ist genau dann erfüllbar, wenn es eine Modell $\langle U, I \rangle$ gibt, wobei U das Herbrand-Universum ist und I die Funktionen als Konstruktoren interpretiert.

Substitutionen

Definition: Eine Substitution ist eine Funktion $\sigma : V \to T_\Sigma$ mit der Eigenschaft, dass $\{X \in V | \sigma(X) \neq X\}$ endlich ist.

- σ weist Variablen Terme zu
  - Wir schreiben z.B.: $\sigma = \{X \leftarrow \mathit{sokrates}, Y \leftarrow \mathit{lehrer}(\mathit{aristoteles})\}$
- Anwendung auf Terme, Atome, Literale, Klauseln möglich!
- Beispiel:
  - $\sigma(Y) = \mathit{lehrer}(\mathit{aristoteles})$
  - $\sigma(\mathit{lauter}(X, \mathit{aristoteles})) = \mathit{lauter}(\mathit{sokrates}, \mathit{aristoteles})$
  - $\sigma(\neg \mathit{mensch}(X) \vee \mathit{sterblich}(X)) =$
    $\neg \mathit{mensch}(\mathit{sokrates}) \vee \mathit{sterblich}(\mathit{sokrates}))$

Übung: Substitutionen

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

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

Fällt Ihnen etwas auf?

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 die Instanz keine Variablen (mehr) enthält, so heißt sie auch eine Grundinstanz.

Satz von Herbrand

Satz: Eine Menge von Klauseln ist genau dann unerfüllbar, wenn die Menge aller ihrer Grundinstanzen (aussagenlogisch) unerfüllbar ist.

- Dabei betrachten wir die Grundatome als aussagenlogische Variablen
- Problem: Die Menge der Grundinstanzen ist in der Regel unendlich

Resolution für Prädikatenlogik kombiniert das Finden von Instanzen und das Herleiten der leeren Klausel.

Idee: Resolution für Prädikatenlogik

- Betrachte folgende Klauseln:
  - $C_1 = p(X, a)$
  - $C_2 = \neg p(f(b), Y)$
- Dann gilt: $\{C_1, C_2\}$ ist unerfüllbar
  - Betrachte $\sigma = \{X \leftarrow f(b), y \leftarrow a\}$
    - $\sigma(C_1) = p(f(b), a)$
    - $\sigma(C_2) = \neg p(f(b), a)$
  - Die beiden Instanzen sind jetzt direkt widersprüchlich.

Wie finden wir ein solches σ automatisch?

Unifikation

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

Definition (Unifikator):

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

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

- Fakt: (Allgemeinste) Unfikatoren können systematisch gefunden werden

Beobachtungen zum Finden von Unifikatoren

1. $\mathit{sterblich}(X), \mathit{mensch}(X)$ können nie unifizieren
   - Das erste Symbol unterscheidet sich immer
2. $X, \mathit{lehrer}(X)$ können nie unifizieren
   - Egal, was für X eingesetzt wird, ein *lehrer* bleibt immer über
   - "Occurs-Check"
3. $X, \mathit{lehrer}(\mathit{sokrates})$ unifizieren mit $\sigma = \{X \leftarrow \mathit{lehrer}(\mathit{sokrates})\}$
   - Variablen und die meisten Terme machen kein Problem
4. $\mathit{lauter}(X, \mathit{sokrates}), \mathit{lauter}(\mathit{aristoteles}, Y)$ unifizieren mit $\sigma = \{X \leftarrow \mathit{aristoteles}, Y \leftarrow \mathit{sokrates}\}$
   - 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: $\{s = t\}, \{\}$
  - Termination: $\{\}, \sigma$

Unifikation: Transformationssystem

$$\text{Binden:}\quad \frac{\{x = t\} \cup R, \sigma}{\{x \leftarrow t\}(R), \{x \leftarrow t\} \circ \sigma} \quad \text{falls } x \notin \text{var}(t)$$

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

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

$$\text{Occurs:}\quad \frac{\{x = t\} \cup R, \sigma}{\mathit{FAIL}} \quad \text{falls } x \in \text{var}(t)$$

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

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

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 $\sigma = \text{mgu}(p, q)$
- Faktorisieren: $\dfrac{C \vee p \vee q}{\sigma(C \vee p)}$ falls $\sigma = \text{mgu}(p, q)$

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

- Wenn $\square$ hergegleitet wird, so ist die Formelmenge unerfüllbar
- Wenn alle Möglichkeiten ausgeschöpft sind, ohne das $\square$ hergeleitet wurde, so is 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
  - $C_1 = \neg \mathit{mensch}(X) \vee \mathit{sterblich}(X)$
  - $C_2 = \mathit{mensch}(\mathit{sokrates})$
  - $C_3 = \neg \mathit{sterblich}(\mathit{sokrates})$
- Saturierung
  - Unifiziere $\mathit{sterblich}(\mathit{sokrates})$ (C_3) und $\mathit{sterblich}(X)$ (C_1)
    - $\sigma = \{X \leftarrow \mathit{sokrates}\}$
    - $\sigma(C_1) = \neg \mathit{mensch}(\mathit{sokrates}) \vee \mathit{sterblich}(\mathit{sokrates})$
    - $\sigma(C_3) = C_3$
    - Resolution zwischen $\sigma(C_1)$ und $\sigma(C_3)$: $C_4 = \neg \mathit{mensch}(\mathit{sokrates})$
  - Resolution zwischen C_2 und C_4: $\square$
- Also: $\{C_1, C_2, C_4\}$ ist unerfüllbar - es gibt keine Möglichkeit, das Sokrates gleichzeitig menschlich und unsterblich ist

Übung: Tierische Resolution

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

Übung: Tierische Resolution

- Axiome:
  - Alle Hunde heulen Nachts
    - $\neg h(X) \vee l(X)$
  - Wer Katzen hat, hat keine Mäuse
    - $\neg k(X) \vee \neg hat(Y, X) \vee \neg hat(Y, Z) \vee \neg m(Z)$
  - 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
    - $hat(john, tier) \wedge (h(tier) \vee k(tier))$
  - John ist empfindlich
    - $e(john)$
- Negierte 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.

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
  - ...

Zurück B

Zurück C

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!`	Setzt das `car` einer existierenden `cons`-Zelle
`set-cdr!`	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 `'()`

Material TINF14B

- Vorlesung 1
- Vorlesung 2
- Vorlesung 3
- Vorlesung 4
- Vorlesung 5
- Vorlesung 6
- Vorlesung 7
- Vorlesung 8
- Vorlesung 9
- Vorlesung 10
- Vorlesung 11
- Vorlesung 12
- Vorlesung 13
- Vorlesung 14
- Vorlesung 15
- Vorlesung 16
- Vorlesung 17
- Vorlesung 18
- Vorlesung 19

Ziele Vorlesung 1

- Gegenseitiges Kennenlernen
- Praktische Informationen
- Übersicht und Motivation

Kennenlernen

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

Praktische Informationen TINF14B

- Vorlesungszeiten
  - Dienstags, 12:15-**14:05**
    - Keine Vorlesung am 2.12., 6.1., 3.2.
  - Freitags, 12:45-15:15
  - Kurze Pausen nach Bedarf
  - 11 Wochen Vorlesung+Klausurwoche
  - Weihnachtspause: 22.12.-4.1.
- Klausur
  - KW10/2015 (2.-6.3.2015)
  - Voraussichtlich 90 Minuten
  - Genauer Termin wird vom Sekretariat koordiniert
- Webseite zur Vorlesung
  - `http://wwwlehre.dhbw-stuttgart.de/~sschulz/lgli2014.html`

Zur Vorlesung

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

- Kurzer Rückblick
- Mathematische Grundbegriffe
- Grundlagen der Mengenlehre
  - (eine) Konstruktion der natürlichen Zahlen
  - Mengenoperationen

Rückblick/Wiederholung

- Praktische Themen
  - Rechner mit Scheme-Umgebung!
- Vorlesungsziel: Vokabular und Methoden
- Ein erstes Formales System: Das MIU-Rätsel
- Einführung Logik
  - Beispiel: Äquivalenz von Spezifikation im Bereich ATM/ATC

Zur Vorlesung

Hausaufgabe

Bearbeiten Sie die Übung *Konstruktion der negativen Zahlen*.

Zusammenfassung

- Kurzer Rückblick
- Mathematische Grundbegriffe
- Grundlagen der Mengenlehre
  - (eine) Konstruktion der natürlichen Zahlen
  - Mengenoperationen

Feedback

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

Ziele Vorlesung 3

- Rückblick/Hausaufgabe Termalgebra
- Übung/Hausaufgabe Mengenoperationen
- Karthesisches Produkt, Potenzmenge
- Mengenalgebra

Rückblick/Wiederholung

- Grundbegriffe
  - Definition
  - Beweis
- Mengen
  - Definition per Aufzählung ($\{1, 2, 4, 8, \ldots\}$)
  - Beschreibend ($\{x | x = 2^n, n \in \mathbb{N}\}$ oder $\{2^n | n \in \mathbb{N}\}$)
  - Mengeneigenschaften
    - Ungeordnet (Gegensatz: z.B. Liste)
    - Keine Duplikate (Stichwort *Multimenge*)
  - Teilmengen ($\subseteq$ vs. $\subset$) und Obermengen
  - Mengengleichheit
  - Konkrete Mengen: $\emptyset, \mathbb{N}, \mathbb{N}^+, \mathbb{Z}, \mathbb{Q}, \mathbb{R}$
- Konstruktion der natürlichen Zahlen
  - Zahlen repräsentiert als $\emptyset, \{\emptyset\}, \{\{\emptyset\}\} \ldots$ oder äquivalent $0, s(0), s(s(0)) \ldots$
  - Rekursive Definition von a (Addition) und m (Multiplikation)
- Mengenoperationen ($\cup, \cap, \setminus, \overline{\ }$ (Komplement), $\triangle$)

Rückblick/Hausaufgabe (1)

- Idee: Wir interpretieren Mengen oder Terme als Zahlen

Mengenschreibweise	Term	Zahl
$\emptyset$ oder $\{\}$	0	0
$\{\emptyset\}$	$s(0)$	1
$\{\{\emptyset\}\}$	$s(s(0))$	2
$\{\{\{\emptyset\}\}\}$	$s(s(s(0)))$	3
$\{\{\{\{\emptyset\}\}\}\}$	$s(s(s(s(0))))$	4
$\ldots$	$\ldots$	$\ldots$

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

Rückblick/Hausaufgabe (2)

- Addition (a):
  (1) $a(X, 0) = X$
  (2) $a(X, s(Y)) = s(a(X, Y))$
- Multiplikation (m):
  (3) $m(X, 0) = 0$
  (4) $m(X, s(Y)) = a(X, m(X, Y))$
- Beispielrechnung: 2×2:

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

Rückblick/Hausaufgabe (3)

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

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

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

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

Erinnerung: Mengenoperationen

- Vereinigung:
  - $M_1 \cup M_2 = \{x | x \in M_1 \text{ oder } x \in M_2\}$
- Schnitt:
  - $M_1 \cap M_2 = \{x | x \in M_1 \text{ und } x \in M_2\}$
- Differenz:
  - $M_1 \backslash M_2 = \{x | x \in M_1 \text{ und } x \notin M_2\}$
- Symmetrische Differenz:
  - $M_1 \triangle M_2 = \{x | x \in M_1 \text{ und } x \notin M_2\} \cup \{x | x \in M_2 \text{ und } x \notin M_1\}$
- Komplement:
  - $\overline{M_1} = \{x | x \notin M_1\}$

Zur Vorlesung

Hausaufgabe

Bearbeiten Sie die Übung *Karthesisches Produkt und Potenzmenge*.

Zusammenfassung

- Rückblick/Hausaufgabe Termalgebra
- Übung/Hausaufgabe Mengenoperationen
- Karthesisches Produkt, Potenzmenge
- Mengenalgebra

Feedback

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

Ziele Vorlesung 4

- Rückblick und Wiederholung
- Relationen
  - Grundkonzepte
  - Eigenschaften
  - Darstellung
  - Relationenalgebra (Anfang)

Rückblick und Wiederholung

- Termalgebra
- Mengenoperationen: $\cup, \cap, \setminus, \overline{}$ (Komplement), $\triangle$
- Karthesisches Produkt und Potenzmenge
- Mengenalgebra

Zur Vorlesung

Zusammenfassung

- Rückblick und Wiederholung
- Relationen
  - Grundkonzepte
  - Eigenschaften
  - Darstellung
  - Relationenalgebra (Anfang)

Feedback

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

Ziele Vorlesung 5

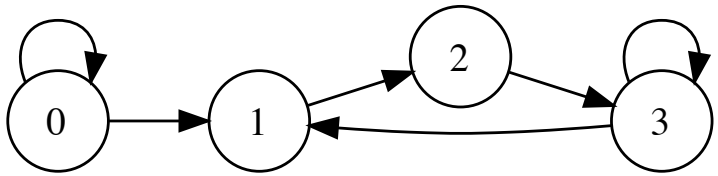
- Rückblick und Wiederholung
- Relationen (Teil 2)
- Funktionen
- Kardinalität
- Erste Schritte mit Scheme

Wiederholung: Relationen 1

- Relation über $M_1, M_2, \ldots, M_n$ ist $R \subseteq M_1 \times M_2 \times \ldots \times M_n$
  - N-Stellige Relation besteht aus N-Tupeln
- Spezialfälle:
  - Homogen: Alle Mengen sind gleich
  - Binär: $R \subseteq M_1 \times M_2$ besteht aus Paaren
  - Homogen und binär: Wie erwartet, häufiger Sonderfall (Z.B. $>, \geq, \neq, =, <, \leq \subseteq \mathbb{N} \times \mathbb{N}$)
- Eigenschaften von (homogenen) binären Relationen:
  - Reflexivität: $\forall x \in M : (x, x) \in R$
  - Symmetrie: $\forall x \in M : (x, y) \in R \rightsquigarrow (y, x) \in R$
  - Transitivität: $\forall x, y, z \in M : (x, y) \in R$ and $(y, z) \in R \rightsquigarrow (x, z) \in R$
  - Linkstotal: $\forall x \in M \exists y \in M : (x, y) \in R$
  - Rechtseindeutig: $\forall x, y, z \in M : (x, y) \in R$ and $(x, z) \in R \rightsquigarrow y = z$

Wiederholung: Relationen 2

- Relationendarstellung (z.T. nur bei endlichen Mengen praktikabel):
  - Als Menge von Paaren
    - Aufzählung: $R = \{(0,0),(0,1),(1,2),(2,3),(3,3),(3,1)\}$
    - Aber auch deskriptiv: $\leq = \{(x, x+i) | x, i \in \mathbb{N}\}$
  - Als Relationsgraph



  - Als Tabelle oder Matrix

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

bzw.

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 3 & 0 & 1 \end{pmatrix}$$

Wiederholung: Relationen 3

- Inverse Relation: $R^{-1} = \{(y, x) \mid (x, y) \in R\}$
- Relationsverknüpfung:
  $R_1 \circ R_2 = \{(x, y) \mid \exists z : (x, z) \in R_1 \text{ und } (z, y) \in R_2\}$
- Potenzierung einer Relation
  - $R^0 = \{(x, x) \mid x \in M\}$ (die Gleichheitsrelation oder Identität)
  - $R^n = R \circ R^{n-1}$ für $n \in \mathbb{N}^+$
- Übung:
  - $M = \{a, b, c, d\}$
  - $R = \{(a, b), (b, c), (c, d)\}$
  - $S = \{(a, a), (b, b), (c, c)\}$
  - Zu berechnen: R^{-1}, R^0, R^1, R^2, R^3, R^4, $S \circ S$, $R \circ S$

Zur Vorlesung

Hausaufgabe

Bearbeiten Sie Übung: *Relationen für Fortgeschrittene*.

Zusammenfassung

- Rückblick und Wiederholung
- Relationen (Teil 2)
- Funktionen
- Kardinalität
- Erste Schritte mit Scheme

Feedback

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

Schöne Pause!

Ziele Vorlesung 6

- Wiederholung/Hausaufgabe
- Einführung Scheme
  - “Hello World” - Starten und Ausführen von Programmen
  - Interaktives Arbeiten
  - Einfache Rekursion

Diskussion Hausaufgabe (1)

- Betrachten Sie die Menge $M = \{a, b, c\}$.
  - Wie viele (binäre homogene) Relationen über M gibt es?
  - 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 Teilemenge von $M \times M$.
    - $M \times M = \{(a, a), (a, b), (a, c), (b, a), (b, b), (b, c), (c, a), (c, b), (c, c)\}$
    - $|M \times M| = 3 \times 3 = 9$
    - $|2^{M \times M}| = 2^9 = 512$
  - Also: Es gibt 512 Teilmengen von $M \times M$, also auch 512 binäre homogene Relationen über M.

Diskussion Hausaufgabe (2)

- Wie viele dieser Relationen sind
  - Linkstotal?
    - Betrachte tabellarische Darstellung:

	a	b	c
a			
b			
c			

    - Linkstotal: Jede Zeile hat *mindestens eine* 1
    - 7 Möglichkeiten pro Zeile: 100, 010, 001, 110, 101, 011, 111
    - Insgesamt also $7^3 = 343$
  - Rechtseindeutig?
    - Jede Zeile hat *höchstens eine* 1
    - 4 Möglichkeiten: 000, 100, 010, 001
    - Also: $4^3 = 64$

Diskussion Hausaufgabe (3)

- Wie viele dieser Relationen sind
  - Reflexiv?
    - 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 gwählt werden
    - Also: $2^6 = 64$ reflexive Relationen
  - Symmetrisch?
    - Felder 1, 5, 9 können frei gewählt werden
    - 2 und 4 sind gekoppelt, 3 und 7 sind gekoppelt, 6 und 8 sind gekoppelt
    - Also: Ich kann die Werte von 3+3 Feldern wählen
    - Also: $2^6 = 64$ symmetrische Relationen

Diskussion Hausaufgabe (4)

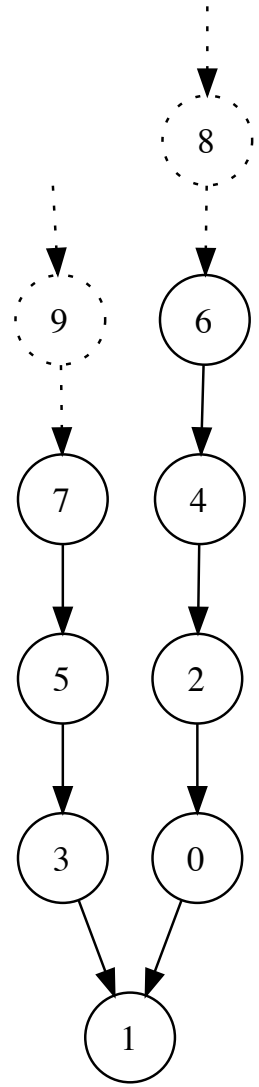
- Wie viele dieser Relationen sind
  - Transitiv?
    - Beobachtung: Wenn R transitiv ist, so gilt: Alles, wass ich in zwei Schritten erreichen kann, kann ich auch in einem Schritt erreichen! Also: $R^2 \subseteq R$
    - Kleines (Scheme-)Programm: 171 der 512 Relationen sind transitiv
  - Funktionen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

Diskussion Hausaufgabe (5)

- Betrachten Sie folgende Relation über $\mathbb{N}$: xRy gdw. $x = y + 2$
  - Was ist die transitive Hülle von R?
    - xR^+y gdw. $x \, mod \, 2 = y \, mod \, 2$ und $x > y$
  - Was ist die reflexive, symmetrische, transitive Hülle von$R$?
    - $(x, y) \in (R \cup R^{-1})^*$ gdw. $x \, mod \, 2 = y \, mod \, 2$ (alle geraden Zahlenstehen in Relation zueinander, und alle ungeraden Zahlen stehen in Relation zueinander)
  - Betrachten Sie $R' = R \cup \{(0, 1)\}$. Was ist die transitive Hülle von R'?
    - $R'^* = R^* \cup \{(x, 1) \mid x \in \mathbb{N}\}$
    - ... aber $(R' \cup R'^{-1})^* = \mathbb{N} \times \mathbb{N}$



Relation R'

Diskussion Hausaufgabe (6)

- Zeigen oder widerlegen (per Gegenbeispiel) Sie:
  - Jede homogene binäre symmetrische und transitive Relation ist eine Äquivalenzrelation
    - Behauptung ist falsch! Gegenbeispiel: Die leere Relation ist symmetrisch, transitiv, aber nicht reflexiv und also keine Äquivalenzrelation.
  - Jede linkstotale homogene binäre symmetrische und transitive Relation ist eine Äquivalenzrelation
    - Behauptung stimmt. Beweis:
      Sei R eine beliebige linkstotale homogene binäre symmetrische und transitive Relation über einer Menge M.
      Zu zeigen: R ist reflexiv
      Sei $a \in M$ beliebig. Da R linkstotal ist, existiert $b \in M$ mit aRb. Da R symmetrisch ist, gilt auch bRa. Da R auch transitiv ist, und aRb und bRa gilt, so gilt auch aRa. Da wir keine weiteren Annahmen gemacht haben, gilt dieses Argument für alle $a \in M$, also ist R reflexiv.

      q.e.d.

Zur Vorlesung

Hausaufgabe

- Bearbeiten Sie Übung: Fibonacci in Ruhe
- Können Sie eine Implementierung skizzieren oder sogar realisieren, die für große Zahlen schneller ist?

Zusammenfassung

- Wiederholung/Hausaufgabe
- Einführung Scheme
  - “Hello World” - Starten und Ausführen von Programmen
  - Interaktives Arbeiten
  - Einfache Rekursion

Feedback

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

Ziele Vorlesung 7

- Rückblick/Wiederholung
- Listenverarbeitung

Rückblick (1)

- Hintergund/Geschichte/Anwendungen
- Scheme Programme ausführen
  - Von der Komandozeile (z.B. `guile-2.0 file.scm`)
  - Interaktiv (read-eval-print loop)
  - Laden von Programmen in den Interpreter (z.B. `(load file.scm")`)
- Syntax (*s-expressions*)
  - Atome (Zahlen, Symbole, Strings, ...)
  - Listen: `(a1 a2 a3)`
- Beispiel *Fakultät*

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

Rückblick (2)

- Semantik
  - Strings, Zahlen, ... haben natürlichen Wert
  - Symbole haben nur dann einen Wert, wenn der (z.B. per `define`) festgelegt wird (sonst Fehler)
  - Listenauswertung (normal)
    - Werte alle Teilausdrücke (wenn nötig rekursiv) aus
    - Interpretiere den Wert des ersten Ausdrucks als Funktion, wende diese auf die Werte der anderen Ausdrücke an
  - *Special forms*
    - Z.B. `(if` tst $expr_1$ $expr_2$`)`
    - `(define pi 3.1415)`
    - `(define (plus3 x) (+ x 3))`
- Datentypen und definierte Funktionen
  - `equal?`
  - Zahlen (`=`, `>`, `+`, `-`, `*`, `/`)
- Übung/Hausaufgabe: Fibonacci

Diskussion: Fibonacci (1)

Direkte Umsetzung der Definition:

```
(define (fib x)
  (if (= x 0)
      0
      (if (= x 1)
          1
          (+ (fib (- x 1))
             (fib (- x 2))))))
```

Korrekt, aber für große x langsam!

Diskussion: Fibonacci (2)

- Idee: Vorwärts rechnen
  - Wir brauchen immer nur die letzten beiden Were!
  - Also Parameter
    - Letzter Wert
    - Vorletzter Wert
    - Wie viele Wert der Folge müssen noch gerechnet werden?

```
(define (fibhelp f1 f2 x)
  (if (= x 0)
      f2
      (fibhelp (+ f1 f2) f1 (- x 1))))

(define (fibfast x)
  (fibhelp 1 0 x))
```

Zur Vorlesung

Hausaufgabe

- Bearbeiten Sie Übung: Mengenlehre in Scheme, idealerweise komplett, mindestens bis einschließlich `intersection`.

Zusammenfassung

- Rückblick/Wiederholung
- Listenverarbeitung

Feedback

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

Ziele Vorlesung 8

- Rückblick/Wiederholung
- Diskussion: Mengenlehre in Scheme
- Berechnungsmodel und Speichermodel
- Neue *special forms*
  - Sequenzen und `begin`
  - `cond`
  - `and/or/not`
  - `let/let*`
- Sortieren (Teil 1)

Rückblick

- Listenfunktionen
  - Leere liste `'()`
  - `null?`
  - `cons`, `car` und `cdr`
  - `append` und `list`
- Rekursion über Listen
  - Basisfall `(null? l)`
  - Rekursion:
    - Mach was mit `(car l)`
    - Rufe die Funktion mit `(cdr l)` rekursiv auf
    - Kombiniere die Ergebnisse
- Beispiel: `revert`

revert

```
;;; Einfach rekursiv
(define (revert l)
  (if (null? l)
      l
      (append (revert (cdr l)) (list (car l)))))

;;; Alternative: Mit Hilfsliste
(define (revert-help l1 l2)
  (if (null? l1)
      l2
      (revert-help (cdr l1) (cons (car l1) l2))))

(define (revert2 l)
  (revert-help l '()))
```

Übung: Mengenlehre in Scheme

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

Diskussion: Mengenlehre in Scheme (1)

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

(define (set-insert x set)
  (if (is-element? x set)
      set
      (cons x set)))

(define (set-union set1 set2)
  (if (null? set1)
      set2
      (set-insert (car set1)
                  (set-union  (cdr set1) set2))))
```

Diskussion: Mengenlehre in Scheme (2)

```
(define (set-intersection set1 set2)
  (if (null? set1)
      set1
      (if (is-element? (car set1) set2)
          (cons (car set1)
                (set-intersection (cdr set1) set2))
          (set-intersection (cdr set1) set2))))

(define (make-pairs m x)
  (if (null? m)
      m
      (cons (list (car m) x)
            (make-pairs (cdr m) x))))

(define (kart m1 m2)
  (if (null? m2)
      m2
      (append (make-pairs m1 (car m2))
              (kart m1 (cdr m2)))))
```

Diskussion: Mengenlehre in Scheme (3)

```
(define (add-to-sets sets element)
  (if (null? sets)
      sets
      (cons (cons element (car sets))
            (add-to-sets (cdr sets) element))))

(define (powerset set)
  (if (null? set)
      (list set)
      (append (powerset (cdr set))
              (add-to-sets (powerset (cdr set))
                           (car set)))))
```

Zur Vorlesung

Hausaufgabe

- Bearbeiten Sie Übung: InsertSort.

Zusammenfassung

- Rückblick/Wiederholung
- Diskussion: Mengenlehre in Scheme
- Berechnungsmodel und Speichermodel
- Neue *special forms*
  - Sequenzen und `begin`
  - `cond`
  - `and/or/not`
  - `let/let*`
- Sortieren (Teil 1)

Feedback

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

Ziele Vorlesung 9

- Rückblick/Wiederholung
- (Keine) Seiteneffekte
- Diskussion: Sortieren durch Einfügen
- Generisches Sortieren
- Mergesort

Rückblick

- Mengenlehre in Scheme
- Umgebungen und Funktionsaufrufe
- *Special Forms*
  - (Pseudo-) Boolsche Verknüpfungen
  - Lokale Variable mit `let/let*`
- Sortieren durch Einfügen

(Keine) Seiteneffekte

- Anmerkung: Bis jetzt programmieren wir (fast) rein funktional
  - Funktionen werden nur wegen ihres Wertes berechnet
- Einzige Ausnahmen bisher:
  - `define` fügt neue Namen zur globalen Umgebung hinzu
  - Output mit `display`, `newline`
- Insbesondere:
  - `(cdr l)` gibt den Rest von `l` zurück, aber verändert die Liste nicht
  - `(+ x 10)` gibt die Summe von `x` und 10 zurück, aber verändert nicht den Wert von `x` (vergleiche auch `(+ 10 x)` und `(+ x y)`)

```
> (define l '(1 2 3))
> l
==> (1 2 3)
> (cdr l)
==> (2 3)
> l
==> (1 2 3)
```

Wir verwenden (bisher) keine Funktionen, die Objekte verändern!

Diskussion: Sortieren durch Einfügen

?

Sortiertes Einfügen

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

> (insert 7 '(1 3 8 12))
==> (1 3 7 8 12)
> (insert 2 '())
==> (2)
> (insert 1 '(2 3))
==> (1 2 3)
> (insert 8 '(1 2 3))
==> (1 2 3 8)
```

Sortieren durch Einfügen

```
> (define (isort l)
        (if (null? l)
            l
            (insert (car l) (isort (cdr l)))))

>  (isort '(2 4 1 0 9))
==> (0 1 2 4 9)
> (isort '(1 2 4 1 0 9 1))
==> (0 1 1 1 2 4 9)
```

Alternative:

```
(define (isort l)
        (if (null? l)
            l
            (let* ((first (car l))
                   (rest  (isort (cdr l))))
              (insert first rest))))
```

Hausaufgabe

- Bearbeiten Sie Übung: Mergesort.

Zusammenfassung

- Rückblick/Wiederholung
- (Keine) Seiteneffekte
- Diskussion: Sortieren durch Einfügen
- Generisches Sortieren
- Mergesort

Feedback

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

Ziele Vorlesung 10

- Rückblick/Wiederholung
- Hausaufgabe/Mergesort
- Rekursion abstrakt
- Input/Output
- Die Wahrheit$^{\mathsf{TM}}$ über Listen

Rückblick

- Sortieren durch Einfügen
- Generisches Sortieren - Funktionen als Parameter
- Mergesort

Mergesort in Scheme

- Direkt Umsetzung
- Hilfsfunktionen
  - `split` teilt eine einzelne Liste in eine Liste von zwei etwa gleichgroße Listen auf
  - `merge` mischt zwei sortierte Listen sortiert zusammen

```
(define (mergesort l)
  (if (or (null? l)
          (null? (cdr l)))
      l
      (let* ((res (split l))
             (l1  (mergesort (car res)))
             (l2  (mergesort (car (cdr res)))))
        (merge l1 l2))))
```

Hilfsfunktion `merge`

- Ist eine der Listen leer, gib die andere zurück
- Ansonsten:
  - Extrahiere das kleinste Element, mische den Rest, hänge das kleinste vor das Ergebnis

```
(define (merge l1 l2)
        (cond ((null? l1)
               l2)
              ((null? l2)
               l1)
              ((< (car l1) (car l2))
               (cons (car l1) (merge (cdr l1) l2)))
              (else
               (cons (car l2) (merge l1 (cdr l2))))))
```

Hilfsfunktion `split`

```
(define (split l)
  (cond ((null? l)              ; Leeres l -> ('() '())
         (list '() '()))
        ((null? (cdr l))        ; Nur ein Element: ('() l)
         (list '() l))
        (else                   ; Sonst: Rest splitten
         (let* ((res (split (cdr (cdr l))))
                (l1  (car res))          ;; Erster Teil
                (l2  (car (cdr res)))    ;; Zweiter Teil
                )
           (list (cons (car l) l1)
                 (cons (car (cdr l)) l2))))))
```

Zur Vorlesung

Hausaufgabe

- Stellen Sie Übung: Komplexität von Sortierverfahren fertig. Wenn Sie die Aufgabe auch quantitativ bearbeiten wollen:
  - Wenn Sie Guile verwenden, dann liefert die Funktion `(gettimeofday)` ein Paar von ganzen Zahlen, dessen `car` die Zahl der Sekunden seit 1970 ist, während das `cdr` die Microsekunden der angefangenen nächsten Sekunde enthält.
  - In Racket/DrScheme gibt die Funktion `(current-inexact-milliseconds)` die Anzahl der Milisekunden seit 1970 als Gleitkommazahlzurück, alternativ akzeptiert die *Special Form* `(time` *`body`*`)` einen beliebigen Ausdruck, evaluiert ihn, und gibt die Laufzeit auf dem aktuellen Ausgabeport aus.

Zusammenfassung

- Rückblick/Wiederholung
- Hausaufgabe/Mergesort
- Rekursion abstrakt
- Input/Output
- Die Wahrheit$^{\mathsf{TM}}$ über Listen

Feedback

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

Ziele Vorlesung 11

- Rückblick/Wiederholung
- Programmieraufgabe: Türme von Hanoi

Es gibt jetzt einen Anhang mit einer Übersicht der von uns behandelten Scheme-Befehle.

Rückblick

- Rekursion als Fallunterscheidung
  - Einfache Fälle sofort
  - Komplexe Fälle zerlegen, kleinere Teilproblem rekursiv lösen, zusammensetzen
- Input/Output
  - `read`/`write`
  - Ports
  - `display`/`newline`
  - `read-char`/`write-char`/`peek-char`
- Kommandozeile und Programmende
- Listen als verkettete `cons`-Paare
- Assoc-Lists
- Hausaufgabe: Vergleich von Sortieralgorithmen

Erinnerung: Hausaufgabe

- Stellen Sie Übung: Komplexität von Sortierverfahren fertig. Wenn Sie die Aufgabe auch quantitativ bearbeiten wollen:
  - Wenn Sie Guile verwenden, dann liefert die Funktion `(gettimeofday)` ein Paar von ganzen Zahlen, dessen `car` die Zahl der Sekunden seit 1970 ist, während das `cdr` die Microsekunden der angefangenen nächsten Sekunde enthält.
  - In Racket/DrScheme gibt die Funktion `(current-inexact-milliseconds)` die Anzahl der Milisekunden seit 1970 als Gleitkommazahlzurück, alternativ akzeptiert die *Special Form* `(time` *`body`*`)` einen beliebigen Ausdruck, evaluiert ihn, und gibt die Laufzeit auf dem aktuellen Ausgabeport aus.

Diskussion: Komplexität von Sortierverfahren

```
Sorting list with 1000 elements
isort start:     0.025104045867919922 s
mergesort start: 0.0023880004882812 5 s

Sorting list with 2000 elements
isort start:     0.10665297508239746 s
mergesort start: 0.006776094436645508 s

Sorting list with 4000 elements
isort start:     0.4172329902648926 s
mergesort start: 0.012583017349243164 s

Sorting list with 8000 elements
isort start:     1.680817127227783 2 s
mergesort start: 0.028363227844238 28 s
```

Was heißt das?

Zur Vorlesung

Hausaufgabe

- Stellen Sie die Übung: Türme von Hanoi fertig

Zusammenfassung

- Rückblick/Wiederholung
- Programmieraufgabe: Türme von Hanoi

Feedback

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

Ziele Vorlesung 12

- Rückblick/Wiederholung
- Diskussion: Türme von Hanoi
- Funktionale Funktionen
- Zerstörung
- Typsystem in Scheme
- Lücken

Erinnerung: Am Dienstag, 3.2. (für 14B) und Mittwoch, 4.2. (für 14C) fällt die Vorlesung jeweils aus. Der Dozent muß was lernen...

Rückblick/Wiederholung

- Aufgabe: Türme von Hanoi

Diskussion: Türme von Hanoi

Zur Vorlesung

Hausaufgabe

- Stellen Sie die Übung: Höfliche Damen fertig.

Zusammenfassung

- Rückblick/Wiederholung
- Diskussion: Türme von Hanoi
- Funktionale Funktionen
- Zerstörung
- Typsystem in Scheme
- Lücken

Haben Sie in den letzen 3 Wochen etwas gelernt?

Feedback

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

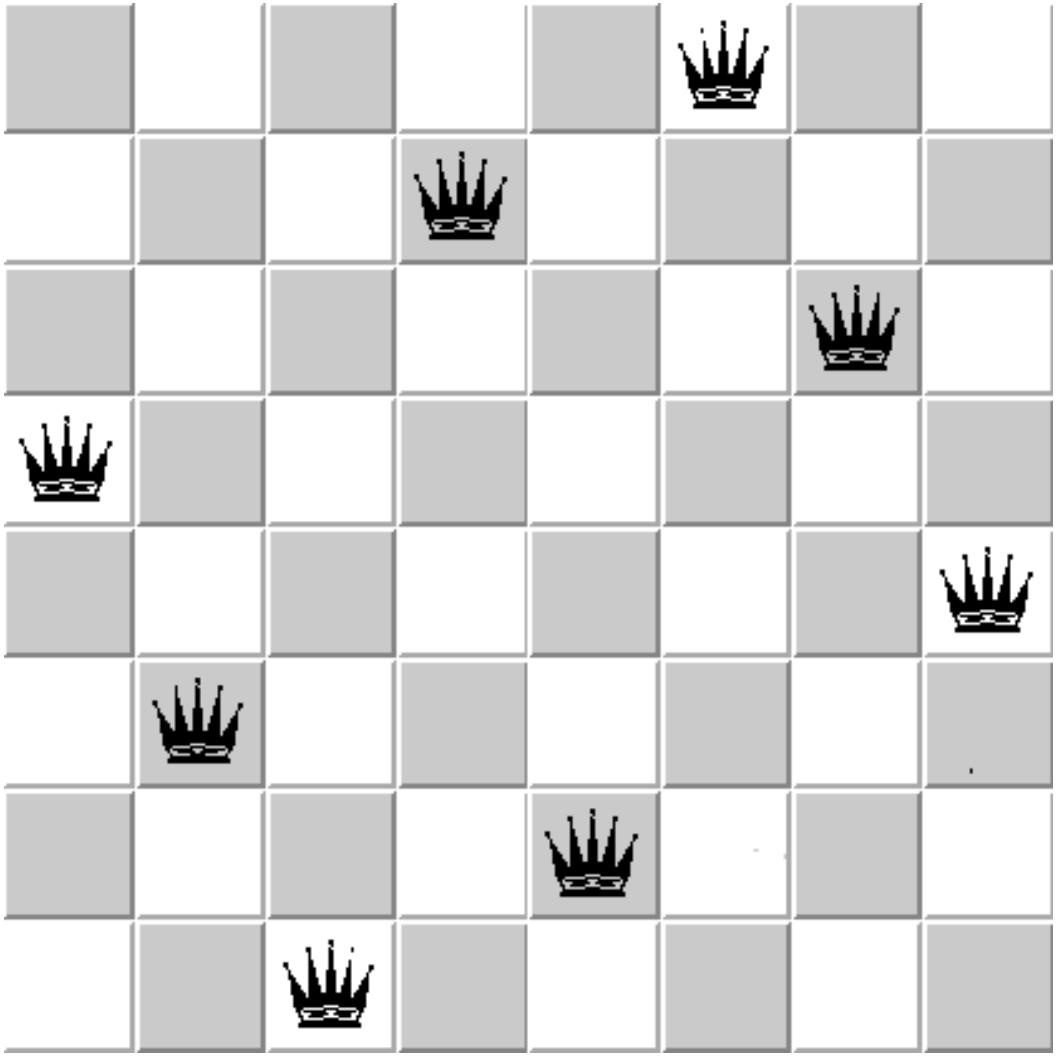
Ziele Vorlesung 13

- Rückblick/Wiederholung
- Hausaufgabe: Damen plazieren
- Aussagenlogik
  - Einführung
  - Syntax
  - Semantik
  - Formalisieren

Rückblick/Wiederholung

- Funktionale Features
  - `lambda, map, apply, eval`
- Destruktive Funktionen:
  - `set!, set-car!, set-cdr!`
- Typsystem
- Lücken
  - Zahlen
  - Strings
  - Vektoren
  - Variadische Funktionen
  - Macros

Diskussion: Damen



Hausaufgabe

- Bearbeiten Sie Übung: Formalisierung von Raubüberfällen

Zusammenfassung

- Rückblick/Wiederholung
- Hausaufgabe: Damen plazieren
- Aussagenlogik
  - Einführung
  - Syntax
  - Semantik
  - Formalisieren

Feedback

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

Ziele Vorlesung 14

- Rückblick/Wiederholung
- Unerfüllbarkeit
- Logisches Folgern
- Wahrheitstafeln
- Auf dem Weg zum Tableaux-Kalkül: Bäume
- Einführung: Analytische Tableaux

Rückblick/Wiederholung

- Beispiel: Craig 1
- Aussagenlogik
  - Einführung
  - Syntax
  - Semantik (Interpretationen, Modelle)
  - Allgemeingültigkeit/Tautologie

Erinnerung: Formalisierung von Raubüberfällen

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

1. Jeder der Männer A, B und C war am Tag des Geschehens in dem Laden gewesen, und kein anderer hatte den Laden an dem Tag betreten.
2. Wenn A schuldig ist, so hat er genau einen Komplizen.
3. Wenn B unschuldig ist, so ist auch C unschuldig.
4. Wenn genau zwei schuldig sind, dann ist A einer von ihnen.
5. Wenn C unschuldig ist, so ist auch B unschuldig.

- Beschreiben Sie die Ermittlungsergebnisse als logische Formeln.
- Lösen sie das Verbrechen!

nach R. Smullyan: "Wie heißt dieses Buch?"

Lösung: Raubüberfälle (Formalisierung)

- Jeder der Männer A, B und C war am Tag des Geschehens in dem Laden gewesen, und kein anderer hatte den Laden an dem Tag betreten.
  - $A \vee B \vee C$
- 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!

Zur Vorlesung

Hausaufgabe

- Bearbeiten Sie Übung: Aussagenlogik in Scheme

Zusammenfassung

- Rückblick/Wiederholung
- Unerfüllbarkeit
- Logisches Folgern
- Wahrheitstafeln
- Auf dem Weg zum Tableaux-Kalkül: Bäume
- Einführung: Analytische Tableaux

Feedback

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

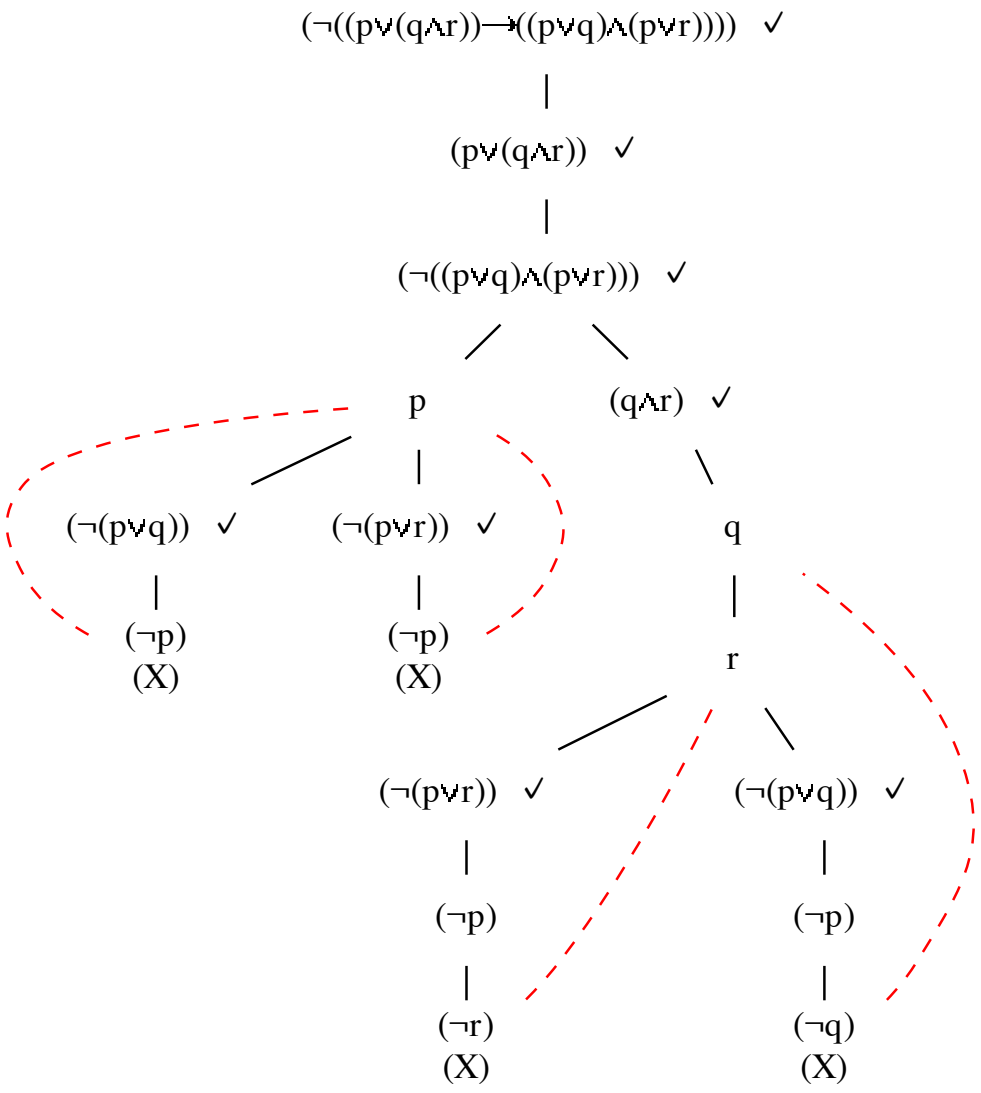
Ziele Vorlesung 15

- Rückblick/Wiederholung
- Hausaufgabe: Aussagenlogik in Scheme
- Tableaux-Kalkül für die Aussagenlogik
  - Üben!
  - Korrektheit und Vollständigkeit
  - Strategien
- Logische Äquivalenz

Rückblick/Wiederholung

- F ist allgemeingültig: $I(F) = 1$ für alle I
- F ist unerfüllbar: $I(F) = 0$ für alle I
- F allgemeingültig gdw. $\neg F$ unerfüllbar (aber es gibt viele Formeln, die weder-noch sind!)
- Logisches Folgern
  - $KB \models F$: Alle Modelle von KB sind auch Modelle von F
  - Wahrheitstafelmethode
  - Deduktionstheorem: $F_1, \ldots, F_n \models G$ gdw. $\models (F_1 \wedge \ldots \wedge F_n) \rightarrow G$
- Bäume
  - Knoten, Kanten, Wurzel, Pfade, Äste
- Einführung: Tableaux-Kalkül für die Aussagenlogik
  - α- und β-Zerlegung

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

Aussagenlogik in Scheme (1)

- Meine Version:
  - Interpretationen sind Scheme-Funktionen, die Atome auf Scheme-Wahrheitswerte abbilden
  - Formeln sind Scheme-Ausdrücke mt Operatoren: `'not`, `'or`, `'and`, `'implies`, `'equiv`
- Beispielformel: $a \rightarrow (b \wedge \neg c) \vee (\neg b \wedge c)$

```
(define g2
   '(implies a (or (and b (not c)) (and (not b) c))))
```

- Beispielinterpretation:

```
(define (make-var-interpretation true-vars)
    (lambda (x) (if (member x true-vars) #t #f)))

(define I1 (make-var-interpretation '(a b)))
(define I2 (make-var-interpretation '(a b c)))
```

Aussagenlogik in Scheme(2)

```
(define (prop-eval form var-interpretation)
  (if (symbol? form)
      (var-interpretation form)
      (let ((op   (car form))
            (args (map (lambda (x)
                         (prop-eval x var-interpretation))
                       (cdr form))))
        (cond ((equal? op 'not)
               (not (car args)))
              ((equal? op 'or)
               (or (car args) (cadr args)))
              ((equal? op 'and)
               (and (car args) (cadr args)))
              ((equal? op 'implies)
               (or (not (car args)) (cadr args)))
              ((equal? op 'equiv)
               (equal?  (car args) (cadr args)))
              (else
               (display "Error in formula")
               (newline))))))
```

Aussagenlogik in Scheme(3)

```
> (define g2
    '(implies a (or (and b (not c)) (and (not b) c))))

> (define I1 (make-var-interpretation '(a b)))
> (define I2 (make-var-interpretation '(a b c)))

> (prop-eval g2 I1)
==> #t
> (prop-eval g2 I2)
==> #f
```

Zur Vorlesung

Hausaufgabe

- Bearbeiten Sie Übung: Tableaux für Inspektor Craig

Zusammenfassung

- Rückblick/Wiederholung
- Hausaufgabe: Aussagenlogik in Scheme
- Tableaux-Kalkül für die Aussagenlogik
  - Üben!
  - Korrektheit und Vollständigkeit
  - Strategien
- Logische Äquivalenz

Feedback

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

Ziele Vorlesung 16

- Rückblick/Wiederholung
- Basen der Aussagenlogik
- Äquivalenzumformungen (kurz)
- Normalformen (NNF/KNF)

Rückblick/Wiederholung

- Abschluß des Tableaux-Kalküls
  - Korrektheit und Vollständigkeit
  - Modellextraktion
  - Strategien
- Logische Äquivalenz
  - Zwei Formeln sind logisch äquivalent, wenn sie unter allen Interpretationen gleich ausgewertet werden

Tableau: Craig 1

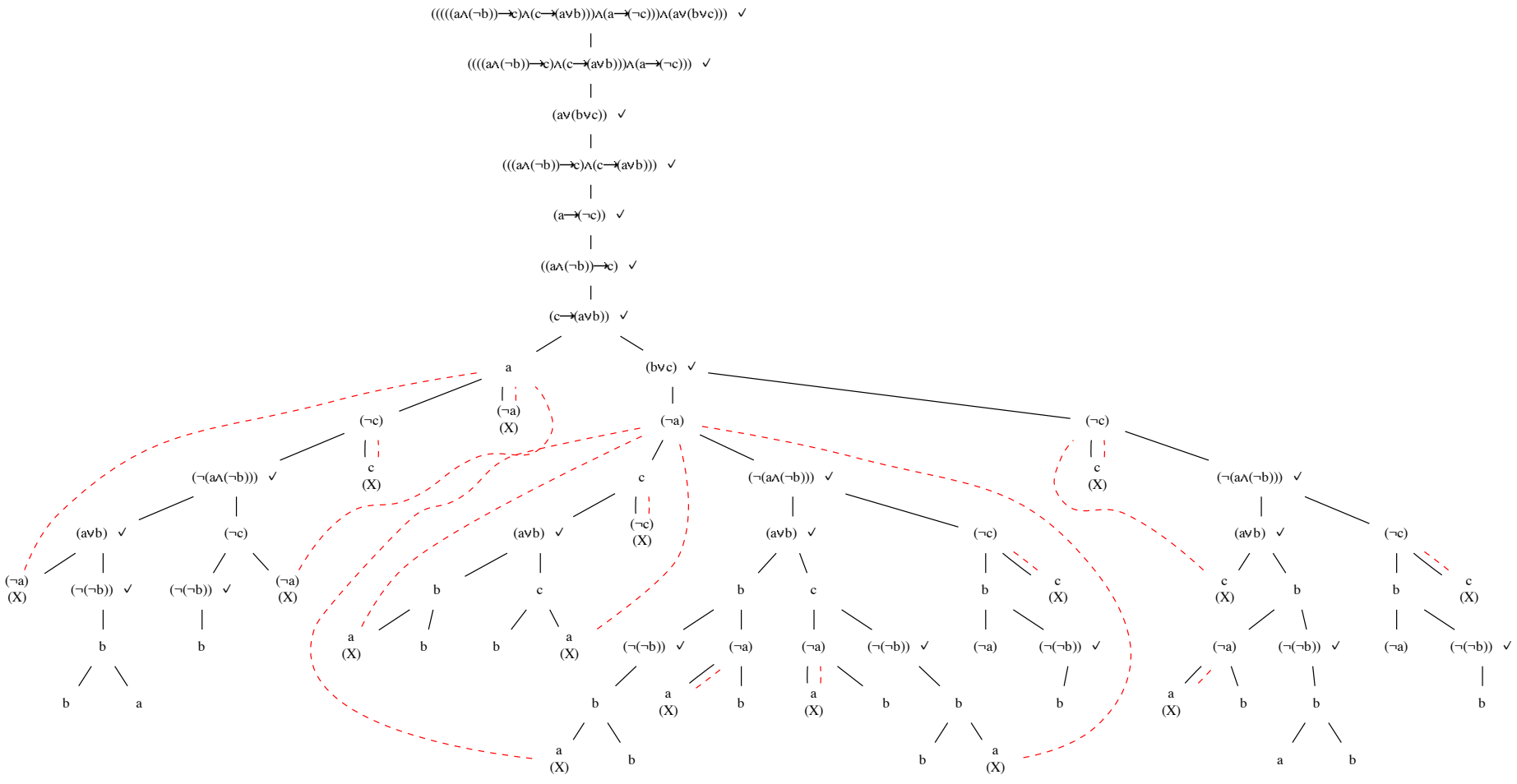
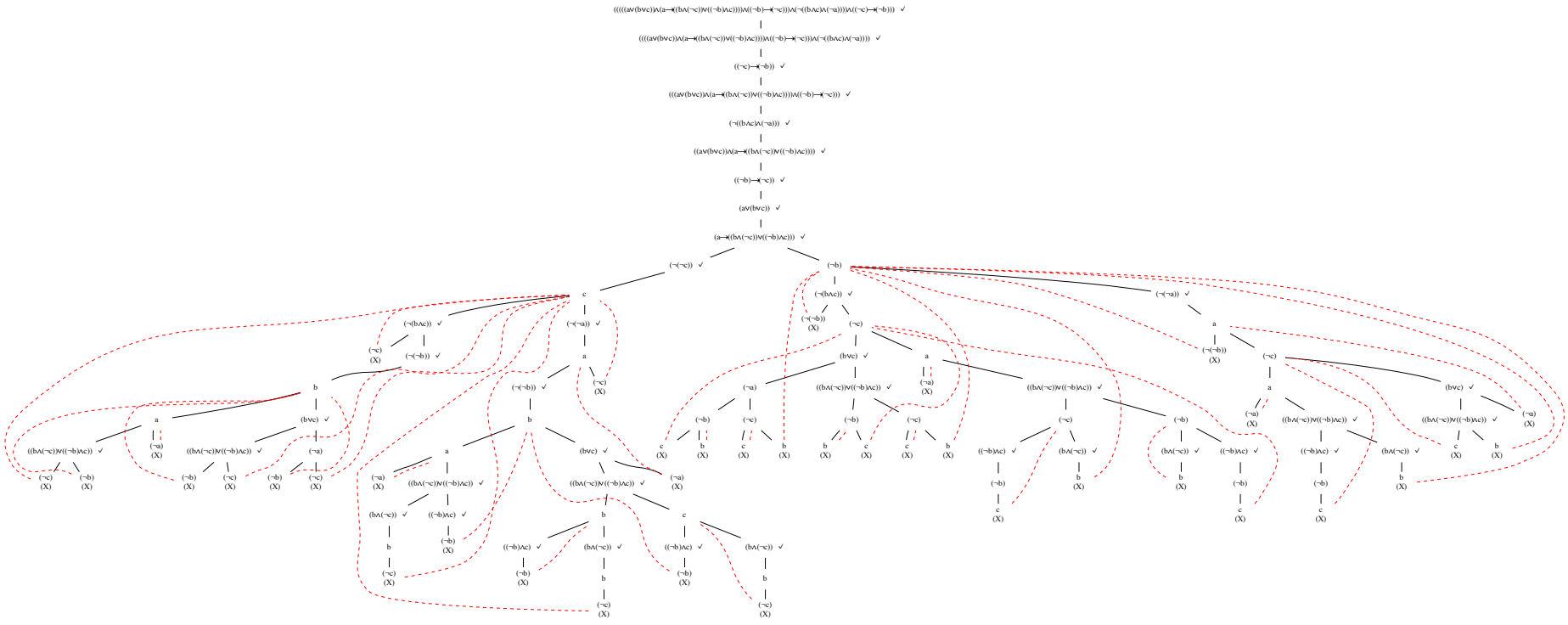


Tableau: Craig 2



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:

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

Beweisprinzip: Strukturelle Induktion

- Die strukturelle Induktion oder Induktion über den Aufbau kann verwendet werden, um Eigenschaften von rekursiv definierten Objekten zu zeigen.
- Idee:
  - Zeige die Eigenschaft für die elementaren Objekte.
  - Zeige die Eigenschaft für die zusammengesetzten Objekte unter der Annahme, dass die Teile bereits die Eigenschaft haben
- Konkret für aussagenlogische Formeln:
  - Basisfälle sind die Aussagenlogischen Variablen oder $\top$, $\bot$.
  - 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:
  - $\{\rightarrow, \neg\}$ 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 enhält 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 entält 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.

Zur Vorlesung

Hausaufgabe

- Bearbeiten Sie Übung: KNF Transformation.

Zusammenfassung

- Rückblick/Wiederholung
- Basen der Aussagenlogik
- Äquivalenzumformungen (kurz)
- Normalformen (NNF/KNF)

Feedback

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

Ziele Vorlesung 17

- Rückblick/Wiederholung
- Konjunktive/Klauselnormalform
- Resolution für Aussagenlogik
- Klausellogik erster Stufe
  - Unifikation
  - (Resolution)

Rückblick/Wiederholung

- Strukturelle Induktion
- Basen der Aussagenlogik
- Äquivalenzumformungen
- NNF/KNF (Kurz)

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 $\wedge$

Lösung: KNF Transformation (Craig)

$(A \vee B \vee C)$		$(A \vee B \vee C)$
$\wedge(A \rightarrow ((B \wedge \neg C) \vee (\neg B \wedge C)))$	$\wedge$	$(\neg A \vee \neg B \vee \neg C) \wedge (\neg A \vee B \vee C)$
		(siehe unten)
$\wedge(\neg B \rightarrow \neg C)$	$\wedge$	$(B \vee \neg C)$
$\wedge\neg(B \wedge C \wedge \neg A)$	$\wedge$	$(\neg B \vee \neg C \vee A)$
$\wedge(\neg C \rightarrow \neg B)$	$\wedge$	$(C \vee \neg B)$

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

Lösung: KNF Transformation (Abstrakte Kunst)

$$
\begin{aligned}
&\neg(a \vee b) \leftrightarrow (b \wedge (a \rightarrow \neg c)) \\
&= (\neg(a \vee b) \rightarrow (b \wedge (a \rightarrow \neg c))) \wedge ((b \wedge (a \rightarrow \neg c)) \rightarrow \neg(a \vee b)) \\
&= (a \vee b \vee (b \wedge (\neg a \vee \neg c))) \wedge (\neg(b \wedge (\neg a \vee \neg c)) \vee \neg(a \vee b)) \\
&= (a \vee b \vee (b \wedge (\neg a \vee \neg c))) \wedge (\neg b \vee \neg(\neg a \vee \neg c) \vee (\neg a \wedge \neg b)) \\
&= (a \vee b \vee (b \wedge (\neg a \vee \neg c))) \wedge (\neg b \vee (a \wedge c) \vee (\neg a \wedge \neg b)) \text{ (NNF)} \\
&= (b \vee a \vee b) \wedge (\neg a \vee \neg c \vee a \vee b) \wedge (\neg a \vee a \vee \neg b) \\
&\wedge (\neg b \vee a \vee \neg b) \wedge (\neg a \vee c \vee \neg b) \wedge (\neg b \vee c \vee \neg b) \\
&= (b \vee a) \wedge (\neg b \vee a) \wedge (\neg a \vee c \vee \neg b) \wedge (\neg b \vee c)
\end{aligned}
$$

Zur Vorlesung

Zeitraffer

- Formeln der Prädikatenlogik werden aus Atomen mit den üblichen Operatoren und den Quantoren $\forall, \exists$ gebaut
- Fakten:
  - Wir können prädikatenlogische Formeln in Klausellogik erster Stufe übersetzen
  - Wir können logische Folgerung in Prädikatenlogik auf die Unerfüllbarkeit von Klauselmengen zurückführen
- Jetzt: Klausellogik erster Stufe

Zur Vorlesung

Hausaufgabe

- Bearbeiten Sie einige der Aufgaben aus der Aufgabensammlung.

Zusammenfassung

- Rückblick/Wiederholung
- Konjunktive/Klauselnormalform
- Resolution für Aussagenlogik
- Klausellogik erster Stufe
  - Unifikation
  - Resolution

Feedback

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

Ziele Vorlesung 18

- Rückblick/Wiederholung
- Unifikation
- Übungen/Aufgaben

Rückblick/Wiederholung

- Konjunktive/Klauselnormalform
- Resolution für Aussagenlogik
- Klausellogik erster Stufe
  - Syntax
    - Signatur: (P, F, V)
    - Terme (aus F, V)
    - Atome (aus P und Termen)
    - Literale und Klauseln wie bei Aussagenlogik
    - Variablen sind all-quantifiziert!
  - Unifikation
  - (Resolution)

Unifikation

Problem: Finde geeignete Ersetzungen für Variablen, so dass zwei Terme/Atome gleich werden.

Andere Formulierung: Finde gemeinsame Instanzen von zwei Atomen! Also: Terme, für die beide Atome ein Aussage machen!

Definition (Unifikator):

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

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

- Fakt: (Allgemeinste) Unfikatoren können systematisch gefunden werden

Beobachtungen zum Finden von Unifikatoren

1. $\mathit{sterblich}(X), \mathit{mensch}(X)$ können nie unifizieren
   - Das erste Symbol unterscheidet sich immer
2. $X, \mathit{lehrer}(X)$ können nie unifizieren
   - Egal, was für X eingesetzt wird, ein *lehrer* bleibt immer über
   - "Occurs-Check"
3. $X, \mathit{lehrer}(\mathit{sokrates})$ unifizieren mit $\sigma = \{X \leftarrow \mathit{lehrer}(\mathit{sokrates})\}$
   - Variablen und die meisten Terme machen kein Problem
4. $\mathit{lauter}(X, \mathit{sokrates}), \mathit{lauter}(\mathit{aristoteles}, Y)$ unifizieren mit $\sigma = \{X \leftarrow \mathit{aristoteles}, Y \leftarrow \mathit{sokrates}\}$
   - 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: $\{s = t\}, \{\}$
  - Termination: $\{\}, \sigma$

Unifikation: Transformationssystem

$$\text{Binden:}\quad \frac{\{x = t\} \cup R, \sigma}{\{x \leftarrow t\}(R), \{x \leftarrow t\} \circ \sigma} \quad \text{falls } x \notin \text{var}(t)$$

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

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

$$\text{Occurs:}\quad \frac{\{x = t\} \cup R, \sigma}{\mathit{FAIL}} \quad \text{falls } x \in \text{var}(t)$$

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

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

Übungen

- Unifikation
- Syntax für Logik erster Stufe
- *Quodlibet*

Hausaufgabe

Zusammenfassung

- Rückblick/Wiederholung
- Unifikation
- Übungen/Aufgaben

Feedback

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

Ziele Vorlesung 19

Übungsklausur

Feedback

- Gerne direkt wie immer
- Auch gerne anonym per Web-System

Viel Glück für die Klausuren!

Ende

Material TINF14C

- Vorlesung 1
- Vorlesung 2
- Vorlesung 3
- Vorlesung 4
- Vorlesung 5
- Vorlesung 6
- Vorlesung 7
- Vorlesung 8
- Vorlesung 9
- Vorlesung 10
- Vorlesung 11
- Vorlesung 12
- Vorlesung 13
- Vorlesung 14
- Vorlesung 15
- Vorlesung 16
- Vorlesung 17
- Vorlesung 18
- Vorlesung 19
- Vorlesung 20
- Vorlesung 21

Ziele Vorlesung 1

- Gegenseitiges Kennenlernen
- Praktische Informationen
- Übersicht und Motivation

Kennenlernen

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

Praktische Informationen TINF14C

- Vorlesungszeiten
  - Montags, 13:00-15:15
  - Freitags, 9:00-11:15
  - Kurze Pausen nach Bedarf
  - 11 Wochen Vorlesung+Klausurwoche
  - Weihnachtspause: 22.12.-4.1.
- Klausur
  - KW10/2015 (2.-6.3.2015)
  - Voraussichtlich 90 Minuten
  - Genauer Termin wird vom Sekretariat koordiniert
- Webseite zur Vorlesung
  - `http://wwwlehre.dhbw-stuttgart.de/~sschulz/lgli2014.html`

Zur Vorlesung

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

- Kurzer Rückblick
- Mathematische Grundbegriffe
- Grundlagen der Mengenlehre
  - (eine) Konstruktion der natürlichen Zahlen

Rückblick/Wiederholung

- Praktische Themen
  - Rechner mit Scheme-Umgebung!
- Vorlesungsziel: Vokabular und Methoden
- Ein erstes Formales System: Das MIU-Rätsel
- Einführung Logik
  - Beispiel: Äquivalenz von Spezifikation im Bereich ATM/ATC

Zur Vorlesung

Hausaufgabe

Bearbeiten Sie die Übung *Konstruktion der negativen Zahlen*

Zusammenfassung

- Kurzer Rückblick
- Mathematische Grundbegriffe
- Grundlagen der Mengenlehre
  - (eine) Konstruktion der natürlichen Zahlen

Feedback

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

Ziele Vorlesung 3

- Kurzer Rückblick
- Grundlagen der Mengenlehre
  - Venn-Diagramme
  - Mengenoperationen
  - Potenzmengen

Rückblick/Wiederholung

- Grundbegriffe
  - Definition
  - Beweis
- Mengen
  - Definition per Aufzählung ($\{1, 2, 4, 8, \ldots\}$)
  - Beschreibend ($\{x | x = 2^n, n \in \mathbb{N}\}$ oder $\{2^n | n \in \mathbb{N}\}$)
  - Mengeneigenschaften
    - Ungeordnet (Gegensatz: z.B. Liste)
    - Keine Duplikate (Stichwort *Multimenge*)
  - Teilmengen ($\subseteq$ vs. $\subset$) und Obermengen
  - Mengengleichheit
  - Konkrete Mengen: $\emptyset, \mathbb{N}, \mathbb{N}^+, \mathbb{Z}, \mathbb{Q}, \mathbb{R}$
- Konstruktion der natürlichen Zahlen
  - Zahlen repräsentiert als $\emptyset, \{\emptyset\}, \{\{\emptyset\}\} \ldots$ oder äquivalent $0, s(0), s(s(0)) \ldots$
  - Rekursive Definition von a (Addition) und m (Multiplikation)

Rückblick/Hausaufgabe (1) TINF14C

- Idee: Wir interpretieren Mengen oder Terme als Zahlen

Mengenschreibweise	Term	Zahl
$\emptyset$ oder $\{\}$	0	0
$\{\emptyset\}$	$s(0)$	1
$\{\{\emptyset\}\}$	$s(s(0))$	2
$\{\{\{\emptyset\}\}\}$	$s(s(s(0)))$	3
$\{\{\{\{\emptyset\}\}\}\}$	$s(s(s(s(0))))$	4
...	...	...

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

Rückblick/Hausaufgabe (2) TINF14C

- 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:

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

Rückblick/Hausaufgabe (3) TINF14C

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

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

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

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

Zur Vorlesung

Zusammenfassung

- Kurzer Rückblick
- Grundlagen der Mengenlehre
  - Venn-Diagramme
  - Mengenoperationen
  - Potenzmengen und karthesische Produkte

Feedback

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

Ziele Vorlesung 4

- Rückblick und Wiederholung
- Relationen und ihre Eigenschaften

Rückblick und Wiederholung TINF14C

- Venn-Diagramme
- Mengenoperationen: $\cup, \cap, \setminus, \overline{\ }$ (Komplement), $\triangle$
- Übung zu Mengenoperationen
- Mengenalgebra
- Karthesisches Produkt und Potenzmenge

Diskussion: Übung Mengenoperationen

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

Zusammenfassung

- Rückblick und Wiederholung
- Relationen und ihre Eigenschaften

Feedback

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

Ziele Vorlesung 4

- Rückblick
- Relationen (Teil 2)
- Funktionen
- Kardinalität

Rückblick

- Mengenalgebra
- Karthesisches Produkt
- Relationen: $R \subseteq M_1 \times \ldots \times M_n$
- Spezieller Relationen
  - Binäre Relationen: $R \subseteq M_1 \times M_2$
  - Homogene Relationen: $R \subseteq M^n$
  - Binäre homogene Relationen: $R \subseteq M^2$
- Relationseigenschaften:
  - Reflexivität: $\forall x \in M : (x, x) \in R$
  - Symmetrie: $\forall x, y \in M : (x, y) \in R \rightsquigarrow (y, x) \in R$
  - Transitivität: $\forall x, y, z \in M : (x, y) \in R$ and $(y, z) \in R \rightsquigarrow (x, z) \in R$
  - Linkstotal: $\forall x \in M \exists y \in M : (x, y) \in R$
  - Rechtseindeutig: $\forall x, y, z \in M : (x, y) \in R$ and $(x, z) \in R \rightsquigarrow y = z$
- Äquivalenzrelationen

Zur Vorlesung

Hausaufgabe

Bearbeiten Sie die Übung *Kardinalität*.

Zusammenfassung

- Rückblick
- Relationen (Teil 2)
- Funktionen
- Kardinalität

Feedback

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

Ziele Vorlesung 6

- Einführung in Scheme
- “Hello World” - Starten und Ausführen von Programmen
- Interaktives Arbeiten
- Einfache Rekursion
- Listenverarbeitung

Zur Vorlesung

Hausaufgabe

Bearbeiten Sie Übung: *Mengenlehre in Scheme*.

Zusammenfassung

- Einführung in Scheme
- “Hello World” - Starten und Ausführen von Programmen
- Interaktives Arbeiten
- Einfache Rekursion
- Listenverarbeitung

Feedback

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

Schöne Pause!

Ziele Vorlesung 7

- Rückblick/Wiederholung
- Hausaufgabe (Mengenlehre in Scheme)
- Funktionale Programmierung und Seiteneffekte
- Speichermodell, Variablen, Werte, Umgebungen
- Wichtige *Special Forms*
  - Sequenzen, `begin`
  - Konditionale: `cond`, `and`, `or`
  - `let`, `let*`

Rückblick (1)

- Hintergund/Geschichte/Anwendungen
- Scheme Programme ausführen
  - Von der Komandozeile (z.B. `guile-2.0 file.scm`)
  - Interaktiv (read-eval-print loop)
  - Laden von Programmen in den Interpreter (z.B. `(load file.scm")`)
- Syntax (*s-expressions*)
  - Atome (Zahlen, Symbole, Strings, ...)
  - Listen: `(a1 a2 a3)`
- Beispiel *Fakultät*

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

Rückblick (2)

- Semantik
  - Strings, Zahlen, ... haben natürlichen Wert
  - Symbole haben nur dann einen Wert, wenn der (z.B. per `define`) festgelegt wird (sonst Fehler)
  - Listenauswertung (normal)
    - Werte alle Teilausdrücke (wenn nötig rekursiv) aus
    - Interpretiere den Wert des ersten Ausdrucks als Funktion, wende diese auf die Werte der anderen Ausdrücke an
  - *Special forms*
    - Z.B. (`if` tst $expr_1$ $expr_2$)
    - `(define pi 3.1415)`
    - `(define (plus3 x) (+ x 3))`
- Datentypen und definierte Funktionen
  - `equal?`
  - Zahlen (`=`, `>`, `+`, `-`, `*`, `/`)
  - Listen `'()`, `car`, `cdr`, `cons`, `append`, `null?`, `list`)

Übung: Mengenlehre in Scheme

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

Diskussion: Mengenlehre in Scheme (1)

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

(define (set-insert x set)
  (if (is-element? x set)
      set
      (cons x set)))

(define (set-union set1 set2)
  (if (null? set1)
      set2
      (set-insert (car set1)
                  (set-union  (cdr set1) set2))))
```

Diskussion: Mengenlehre in Scheme (2)

```
(define (set-intersection set1 set2)
  (if (null? set1)
      set1
      (if (is-element? (car set1) set2)
          (cons (car set1)
                (set-intersection (cdr set1) set2))
          (set-intersection (cdr set1) set2))))

(define (make-pairs m x)
  (if (null? m)
      m
      (cons (list (car m) x)
            (make-pairs (cdr m) x))))

(define (kart m1 m2)
  (if (null? m2)
      m2
      (append (make-pairs m1 (car m2))
              (kart m1 (cdr m2)))))
```

Diskussion: Mengenlehre in Scheme (3)

```
(define (add-to-sets sets element)
  (if (null? sets)
      sets
      (cons (cons element (car sets))
            (add-to-sets (cdr sets) element))))

(define (powerset set)
  (if (null? set)
      (list set)
      (append (powerset (cdr set))
              (add-to-sets (powerset (cdr set))
                           (car set)))))
```

Zur Vorlesung

Hausaufgabe

- Bearbeiten Sie Übung: Fibonacci und Potenzmenge

Zusammenfassung

- Rückblick/Wiederholung
- Hausaufgabe (Mengenlehre in Scheme)
- Funktionale Programmierung und Seiteneffekte
- Speichermodell, Variablen, Werte, Umgebungen
- Wichtige *Special Forms*
  - Sequenzen, `begin`
  - Konditionale: `cond`, `and`, `or`
  - `let`, `let*`

Feedback

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

Ziele Vorlesung 8

- Rückblick und Wiederholung (kurz!)
- Sortieren durch Einfügen
- Generische Programmierung
- Vorstellung Mergesort

Rückblick/Wiederholung

- Auswertung vs. Seiteneffekte
- Variablen, Umgebungen
- Scheme-Konstrukte:
  - Sequenzen und `begin`
  - `cond`
  - `and`, `or`, `not`
  - `let`, `let*`

Hausaufgabe (Fibonacci)

Fibonacci mit `if`

```
(define (fib x)
  (if (= x 0)
      0
      (if (= x 1)
          1
          (+ (fib (- x 1))
             (fib (- x 2))))))
```

Fibonacci mit `cond`

```
(define (fib2 x)
  (cond ((= x 0)
         0)
        ((= x 1)
         1)
        (else
         (+ (fib2 (- x 1))
            (fib2 (- x 2))))))
```

Hausaufgabe: Potenzmenge (Hilfsfunktion v1)

Hilfsfunktion: `add-to-sets`

```
;; Eingabe:
;;   sets: Eine Liste von Listen (interpretiert als
;;         Menge von Mengen)
;;   element: Ein einzelnes Element
;; Ausgabe:
;;   Liste von Listen, wobei jede der Originalliste
;;   um das einzelne Element erweitert wird

(define (add-to-sets sets element)
  (if (null? sets)
      sets
      (cons (cons element (car sets))
            (add-to-sets (cdr sets) element))))
```

Hausaufgabe: Potenzmenge (Hilfsfunktion v2)

Hilfsfunktion: `add-to-sets`

```
;; Eingabe:
;;   sets: Eine Liste von Listen (interpretiert als
;;         Menge von Mengen)
;;   element: Ein einzelnes Element
;; Ausgabe:
;;   Liste von Listen, wobei jede der Originalliste
;;   um das einzelne Element erweitert wird

(define (add-to-sets sets element)
  (if (null? sets)
      sets
      (let* ((first-set (car sets))
             (rest-sets (cdr sets))
             (first-new (cons element first-set))
             (rest-new  (add-to-sets rest-sets element)))
        (cons first-new rest-new))))
```

Hausaufgabe: Potenzmenge (Hilfsfunktion Beispiele)

```
> (add-to-sets '(() (1) (1 2) (1 2 3)) 5)
```

$\Longrightarrow$ `((5) (5 1) (5 1 2) (5 1 2 3))`

```
> (add-to-sets '((1 2) (2 3) (3 4)) 5)
```

$\Longrightarrow$ `((5 1 2) (5 2 3) (5 3 4))`

```
> (add-to-sets '(()) 5)
```

$\Longrightarrow$ `((5))`

```
> (add-to-sets '() 5)
```

$\Longrightarrow$ `()`

Hausaufgabe: Potenzmenge

```
(define (powerset set)
  (if (null? set)
      (list set)
      (append (powerset (cdr set))
              (add-to-sets (powerset (cdr set))
                           (car set)))))
;; Berechne Potenzmenge von l
(define (powerset2 l)
  (cond ((null? l)
         (list l))
        (else
         (let* ((sub-ps  (powerset2 (cdr l)))
                (new     (car l))
                (sub-new (add-to-sets sub-ps new))
                (result  (append sub-ps sub-new)))
           result))))
```

Zur Vorlesung

Hausaufgabe

- Bearbeiten Sie die Übung: Mergesort

Zusammenfassung

- Rückblick und Wiederholung (kurz!)
- Sortieren durch Einfügen
- Generische Programmierung
- Vorstellung Mergesort

Feedback

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

Ziele Vorlesung 9

- Hausaufgabe/Mergesort
- Funktionen und Rekursion
- Input/Output

Rückblick/Wiederholung

- Mengenlehre in Scheme
- Sortieren durch Einfügen
- . . . in Scheme
- Sortieren mit Vergleichsfunktion als Parameter
- Vorstellung Mergesort
  - Teile Sortierproblem in zwei kleinere Probleme
  - Sortiere diese rekursiv
  - Füge die sortierten Teillisten sortiert zusammen

Mergesort in Scheme

- Direkt Umsetzung
- Hilfsfunktionen
  - `split` teilt eine einzelne Liste in eine Liste von zwei etwa gleichgroße Listen auf
  - `merge` mischt zwei sortierte Listen sortiert zusammen

```
(define (mergesort l)
  (if (or (null? l)
          (null? (cdr l)))
      l
      (let* ((res (split l))
             (l1  (mergesort (car res)))
             (l2  (mergesort (car (cdr res)))))
        (merge l1 l2))))
```

Hilfsfunktion `merge`

- Ist eine der Listen leer, gib die andere zurück
- Ansonsten:
  - Extrahiere das kleinste Element, mische den Rest, hänge das kleinste vor das Ergebnis

```
(define (merge l1 l2)
        (cond ((null? l1)
               l2)
              ((null? l2)
               l1)
              ((< (car l1) (car l2))
               (cons (car l1) (merge (cdr l1) l2)))
              (else
               (cons (car l2) (merge l1 (cdr l2))))))
```

Hilfsfunktion `split`

```
(define (split l)
  (cond ((null? l)              ; Leeres l -> ('() '())
         (list '() '()))
        ((null? (cdr l))        ; Nur ein Element: ('() l)
         (list '() l))
        (else                   ; Sonst: Rest splitten
         (let* ((res (split (cdr (cdr l))))
                (l1  (car res))          ;; Erster Teil
                (l2  (car (cdr res)))    ;; Zweiter Teil
                )
           (list (cons (car l) l1)
                 (cons (car (cdr l)) l2))))))
```

Zur Vorlesung

Hausaufgabe

- Stellen Sie die Übung: Türme von Hanoi fertig

Zusammenfassung

- Hausaufgabe/Mergesort
- Funktionen und Rekursion
- Input/Output

Feedback

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

Ziele Vorlesung 10

- Rückblick/Wiederholung
- Hausaufgabe: Vergleich von Sortierverfahren
- Die Wahrheit$^{\mathsf{TM}}$ über Listen
- Ein nicht-triviales Beispiel: Türme von Hanoi

Rückblick/Wiederholung

- Semantik von Funktionsaufrufen
- Rekursion im allgemeinen
- Input/Output
  - `read`/`write`
  - Ports
  - `display`/`newline`
  - `read-char`/`write-char`/`peek-char`
- Hausaufgabe: Vergleich von Sortieralgorithmen

Diskussion: Komplexität von Sortierverfahren

```
Sorting list with 1000 elements
isort start:     0.025104045867919922 s
mergesort start: 0.0023880004882812 5 s

Sorting list with 2000 elements
isort start:     0.10665297508239746 s
mergesort start: 0.006776094436645508 s

Sorting list with 4000 elements
isort start:     0.4172329902648926 s
mergesort start: 0.012583017349243164 s

Sorting list with 8000 elements
isort start:     1.680817127227783 2 s
mergesort start: 0.028363227844238 28 s
```

Was heißt das?

Zur Vorlesung

Hausaufgabe

- Stellen Sie die Übung: Türme von Hanoi fertig

Zusammenfassung

- Rückblick/Wiederholung
- Hausaufgabe: Vergleich von Sortierverfahren
- Die Wahrheit$^{\mathsf{TM}}$ über Listen
- Ein nicht-triviales Beispiel: Türme von Hanoi

Feedback

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

Ziele Vorlesung 11

- Rückblick/Wiederholung
- Diskussion: Türme von Hanoi
- Funktionale Funktionen
- Zerstörung
- Lücken

Rückblick/Wiederholung

- Listenstruktur
- Listenbefehle
- Aufgabe: Türme von Hanoi

Diskussion: Türme von Hanoi

Zur Vorlesung

Hausaufgabe

- Stellen Sie die Übung: Höfliche Damen fertig.

Zusammenfassung

- Rückblick/Wiederholung
- Diskussion: Türme von Hanoi
- Funktionale Funktionen
- Zerstörung
- Lücken

Haben Sie in den letzen 3 Wochen etwas gelernt?

Feedback

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

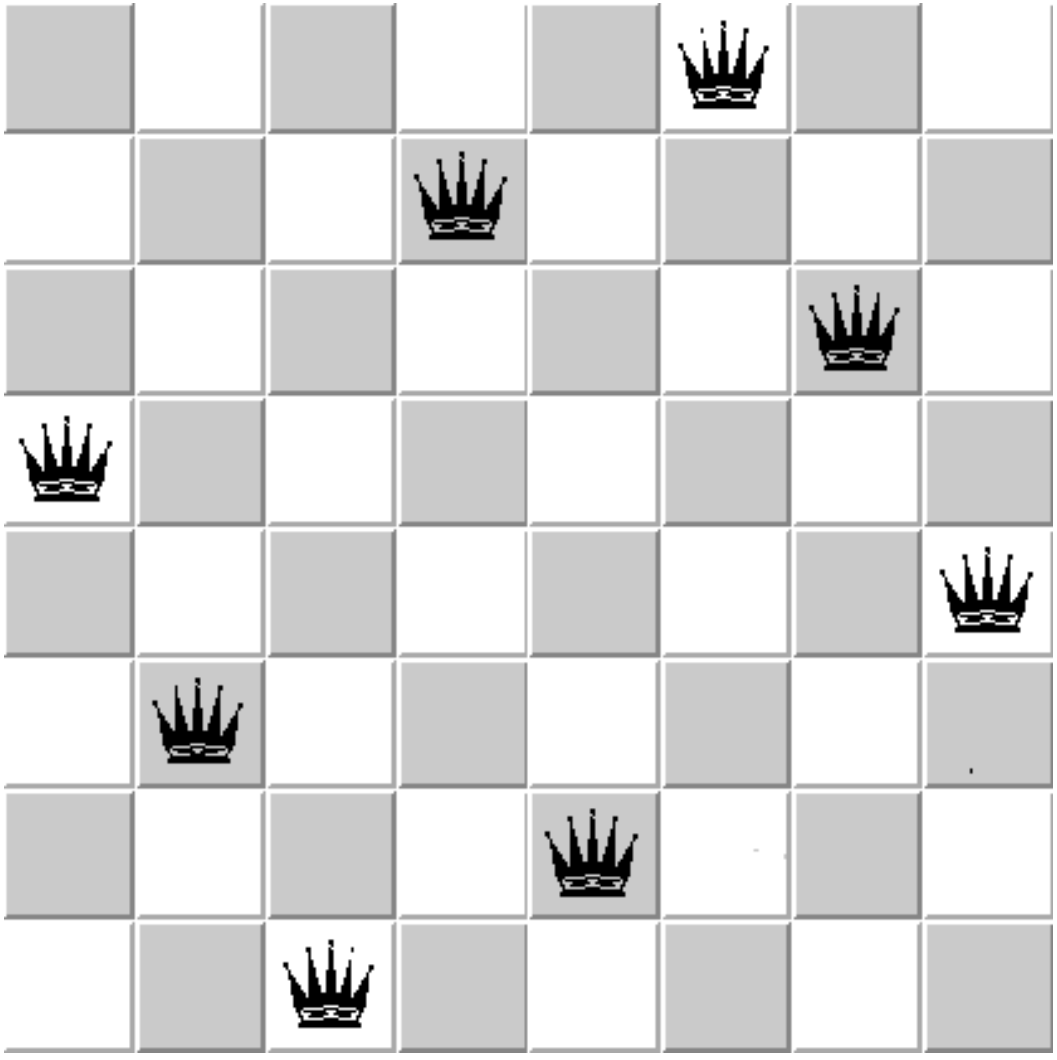
Ziele Vorlesung 12

- Rückblick/Wiederholung
- Hausaufgabe: Damen plazieren
- Aussagenlogik
  - Einführung
  - Syntax
  - Semantik
  - Formalisieren

Rückblick/Wiederholung

- Diskussion: Türme von Hanoi (mal zwei)
- Funktionale Features
  - `lambda, map, apply, eval`
- Destruktive Funktionen:
  - `set!, set-car!, set-cdr!`
- Lücken
  - Zahlen
  - Strings
  - Vektoren
  - Variadische Funktionen
  - Macros

Diskussion: Damen



Hausaufgabe

- Bearbeiten Sie Übung: Formalisierung von Raubüberfällen

Zusammenfassung

- Rückblick/Wiederholung
- Hausaufgabe: Damen plazieren
- Aussagenlogik
  - Einführung
  - Syntax
  - Semantik
  - Formalisieren

Feedback

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

Ziele Vorlesung 13

- Rückblick/Wiederholung
- Allgemeingültigkeit und Unerfüllbarkeit
- Logisches Folgern
- Wahrheitstafeln
- Bäume

Rückblick/Wiederholung

- Inspektor Craig
- Logik: Syntax/Semantik/Kalkül
- Syntax der Aussagenlogik
  - Atomare Aussagen ("Variablen")
  - Operatoren: $\top, \bot, \neg, \wedge, \vee, \rightarrow, \leftrightarrow$
  - Klammern (und wie man sie vermeidet)
    - Präzedenz und Assoziativität
- Semantik der Aussagenlogik
  - Interpretationen auf Atomen
  - Fortsetzung auf Formeln
  - Modellbegriff

Es gibt jetzt einen Anhang mit einer Übersicht der von uns behandelten Scheme-Befehle.

Erinnerung: Formalisierung von Raubüberfällen

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

1. Jeder der Männer A, B und C war am Tag des Geschehens in dem Laden gewesen, und kein anderer hatte den Laden an dem Tag betreten.
2. Wenn A schuldig ist, so hat er genau einen Komplizen.
3. Wenn B unschuldig ist, so ist auch C unschuldig.
4. Wenn genau zwei schuldig sind, dann ist A einer von ihnen.
5. Wenn C unschuldig ist, so ist auch B unschuldig.

- Beschreiben Sie die Ermittlungsergebnisse als logische Formeln.
- Lösen sie das Verbrechen!

nach R. Smullyan: "Wie heißt dieses Buch?"

Lösung: Raubüberfälle (Formalisierung)

- Jeder der Männer A, B und C war am Tag des Geschehens in dem Laden gewesen, und kein anderer hatte den Laden an dem Tag betreten.
  - $A \vee B \vee C$
- 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!

Zur Vorlesung

Hausaufgabe

- Bearbeiten Sie Übung: Aussagenlogik in Scheme

Zusammenfassung

- Rückblick/Wiederholung
- Allgemeingültigkeit und Unerfüllbarkeit
- Logisches Folgern
- Wahrheitstafeln
- Bäume

Feedback

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

Ziele Vorlesung 14

- Rückblick/Wiederholung
- Tableaux-Kalkül für die Aussagenlogik
  - Vorstellung und Übung

Erinnerung: Am Dienstag, 3.2. (für 14B) und Mittwoch, 4.2. (für 14C) fällt die Vorlesung jeweils aus. Der Dozent muß was lernen...

Rückblick/Wiederholung

- F ist allgemeingültig: $I(F) = 1$ für alle I
- F ist unerfüllbar: $I(F) = 0$ für alle I
- Logisches Folgern
  - $KB \models F$: Alle Modelle von KB sind auch Modelle von F
  - Wahrheitstafelmethode (gleich)
- Bäume
  - Knoten, Kanten, Wurzel, Pfade, Äste

Die Wahrheitstafelmethode

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

Folgerungsproblem von Craig

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

Schritt 1: Aufzählung aller möglichen Welten

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

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

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

Vermutung 1: A ist schuldig!

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

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

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

Vermutung 2: B ist schuldig!

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

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) $KB \models (A \vee B)$

Aussagenlogik in Scheme (1)

- Meine Version:
  - Interpretationen sind Scheme-Funktionen, die Atome auf Scheme-Wahrheitswerte abbilden
  - Formeln sind Scheme-Ausdrücke mt Operatoren: `'not`, `'or`, `'and`, `'implies`, `'equiv`
- Beispielformel: $a \rightarrow (b \wedge \neg c) \vee (\neg b \wedge c)$

```
(define g2
   '(implies a (or (and b (not c)) (and (not b) c))))
```

- Beispielinterpretation:

```
(define (make-var-interpretation true-vars)
    (lambda (x) (if (member x true-vars) #t #f)))

(define I1 (make-var-interpretation '(a b)))
(define I2 (make-var-interpretation '(a b c)))
```

Aussagenlogik in Scheme(2)

```
(define (prop-eval form var-interpretation)
  (if (symbol? form)
      (var-interpretation form)
      (let ((op   (car form))
            (args (map (lambda (x)
                         (prop-eval x var-interpretation))
                       (cdr form))))
        (cond ((equal? op 'not)
               (not (car args)))
              ((equal? op 'or)
               (or (car args) (cadr args)))
              ((equal? op 'and)
               (and (car args) (cadr args)))
              ((equal? op 'implies)
               (or (not (car args)) (cadr args)))
              ((equal? op 'equiv)
               (equal?  (car args) (cadr args)))
              (else
               (display "Error in formula")
               (newline))))))
```

Aussagenlogik in Scheme(3)

```
> (define g2
    '(implies a (or (and b (not c)) (and (not b) c))))

> (define I1 (make-var-interpretation '(a b)))
> (define I2 (make-var-interpretation '(a b c)))

> (prop-eval g2 I1)
==> #t
> (prop-eval g2 I2)
==> #f
```

Zur Vorlesung

Hausaufgabe

- Bearbeiten Sie Übung: Tableaux für Inspektor Craig

Zusammenfassung

- Rückblick/Wiederholung
- Tableaux-Kalkül für die Aussagenlogik
  - Vorstellung und Übung

Feedback

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

Ziele Vorlesung 15

- Rückblick/Wiederholung
- Scheme: Typen und Symbole
- Tableaux-Kalkül für die Aussagenlogik
  - Korrektheit und Vollständigkeit
  - Strategien
- Logische Äquivalenz

Erinnerung: Am Dienstag, 3.2. (für 14B) und Mittwoch, 4.2. (für 14C) fällt die Vorlesung jeweils aus. Der Dozent muß was lernen...

Rückblick/Wiederholung

- Besprechung: Logik in Scheme
- Tableaux-Kalkül
  - $\alpha - /\beta$-Zerlegungen
  - Tableau-Konstruktion
  - Geschlossenes Tableau = Unerfüllbare Wurzel

Tableau: Craig 1

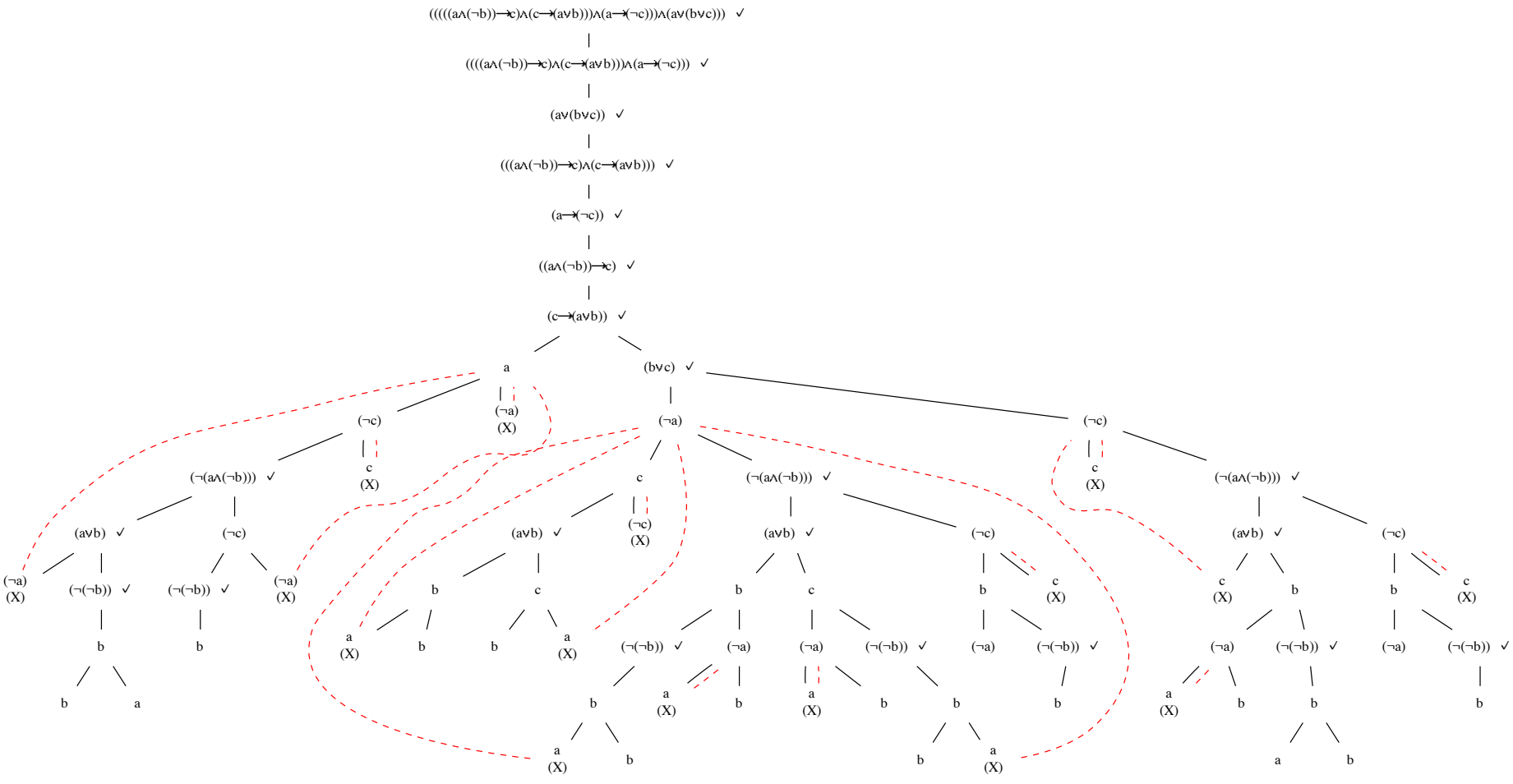
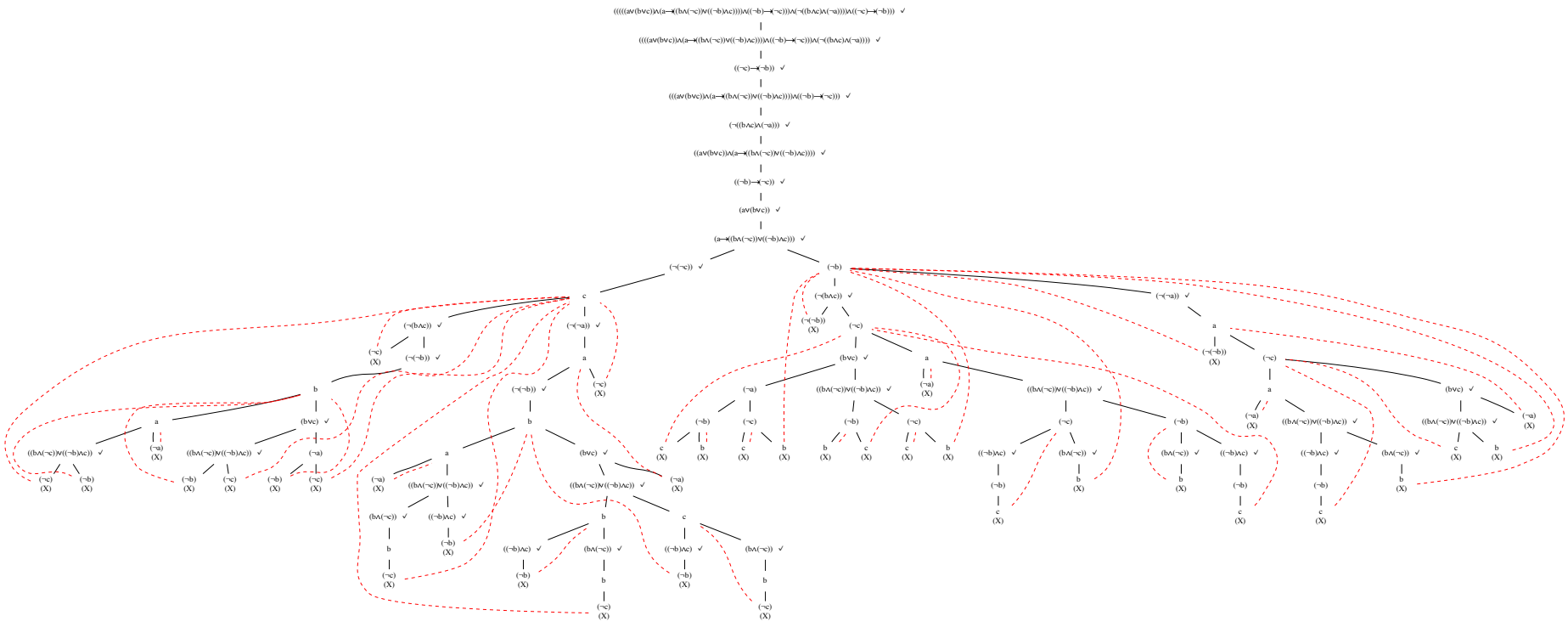
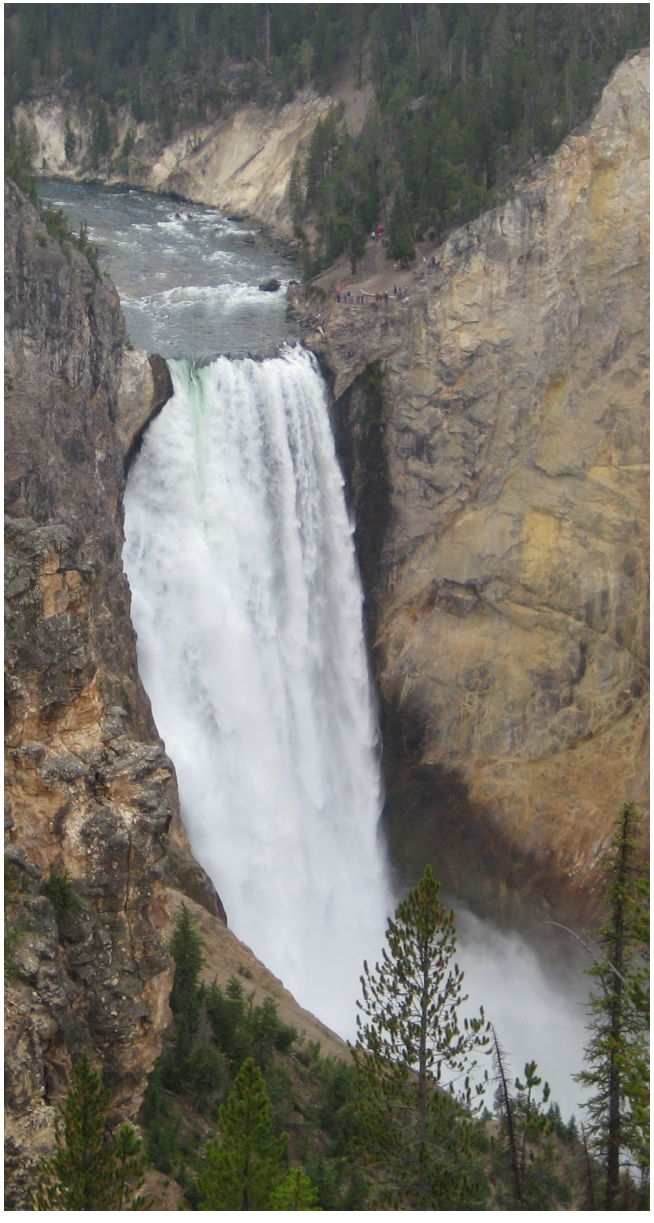


Tableau: Craig 2



Das Typsystem von Scheme



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

Die Typen von Scheme

- Typprädikate (jedes Objekt hat genau einen dieser Typen):

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

Symbole als Werte

> *Symbols are objects whose usefulness rests on the fact that two symbols are identical (in the sense of eqv?) if and only if their names are spelled the same way.*
>
> *R5RS (6.3.3)*

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

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

Zur Vorlesung

Hausaufgabe

- Bearbeiten Sie Übung: Basis der Aussagenlogik

Zusammenfassung

- Rückblick/Wiederholung
- Tableaux-Kalkül für die Aussagenlogik
  - Korrektheit und Vollständigkeit
  - Strategien
- Logische Äquivalenz

Feedback

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

Ziele Vorlesung 16

- Rückblick/Wiederholung
- Hausaufgabe: Basis der Aussagenlogik
- Substitutionstheorem
- Kalkül der Äquivalenzumformungen

Rückblick/Wiederholung

- Scheme-Reste (Typen und Symbole)
- Abschluß des Tableaux-Kalküls
  - Korrektheit und Vollständigkeit
  - Modellextraktion
  - Strategien
- Logische Äquivalenz
  - Zwei Formeln sind logisch äquivalent, wenn sie unter allen Interpretationen gleich ausgewertet werden
- Teilformeln
- Beweisprinzip: Strukturelle Induktion

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:

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

Beweisprinzip: Strukturelle Induktion

- Die strukturelle Induktion oder Induktion über den Aufbau kann verwendet werden, um Eigenschaften von rekursiv definierten Objekten zu zeigen.
- Idee:
  - Zeige die Eigenschaft für die elementaren Objekte.
  - Zeige die Eigenschaft für die zusammengesetzten Objekte unter der Annahme, dass die Teile bereits die Eigenschaft haben
- Konkret für aussagenlogische Formeln:
  - Basisfälle sind die Aussagenlogischen Variablen oder $\top$, $\bot$.
  - 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.

$\{\rightarrow, \neg\}$ 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 enhält 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 entält 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.

Übung: Basis der Aussagenlogik

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

Zur Vorlesung

Hausaufgabe

- Vollziehen Sie den Beweis zum Substitutionstheorem nach.

Zusammenfassung

- Rückblick/Wiederholung
- Hausaufgabe: Basis der Aussagenlogik
- Substitutionstheorem
- Kalkül der Äquivalenzumformungen

Feedback

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

Ziele Vorlesung 17

- Rückblick/Wiederholung
- Normalformen
  - NNF
  - KNF
- Resolution für Aussagenlogik

Rückblick/Wiederholung

- Basen der Aussagenlogik
- Strukturelle Induktion
- Äquivalenzumformungen
- Kalkül $\vdash_{LU}$

Zur Vorlesung

Hausaufgabe

- Bearbeiten Sie Übung: Resolution von Jane

Zusammenfassung

- Rückblick/Wiederholung
- Normalformen
  - NNF
  - KNF
- Resolution für Aussagenlogik

Feedback

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

Ziele Vorlesung 18

- Rückblick/Wiederholung
- Prädikatenlogik 1. Stufe
  - Syntax
  - Semantik

Rückblick/Wiederholung

- Negations-Normalform
- Konjunktive Normalform
- Unerfüllbarkeit mit Resolution

Lösung: KNF Transformation

$$
\begin{aligned}
& \neg(a \vee b) \leftrightarrow (b \wedge (a \rightarrow \neg c)) \\
= & (\neg(a \vee b) \rightarrow (b \wedge (a \rightarrow \neg c))) \wedge ((b \wedge (a \rightarrow \neg c)) \rightarrow \neg(a \vee b)) \\
= & (a \vee b \vee (b \wedge (\neg a \vee \neg c))) \wedge (\neg(b \wedge (\neg a \vee \neg c)) \vee \neg(a \vee b)) \\
= & (a \vee b \vee (b \wedge (\neg a \vee \neg c))) \wedge (\neg b \vee \neg(\neg a \vee \neg c) \vee (\neg a \wedge \neg b)) \\
= & (a \vee b \vee (b \wedge (\neg a \vee \neg c))) \wedge (\neg b \vee (a \wedge c) \vee (\neg a \wedge \neg b)) \text{ (NNF)} \\
= & (b \vee a \vee b) \wedge (\neg a \vee \neg c \vee a \vee b) \wedge (\neg a \vee a \vee \neg b) \\
& \wedge (\neg b \vee a \vee \neg b) \wedge (\neg a \vee c \vee \neg b) \wedge (\neg b \vee c \vee \neg b) \\
= & (b \vee a) \wedge (\neg b \vee a) \wedge (\neg a \vee c \vee \neg b) \wedge (\neg b \vee c)
\end{aligned}
$$

Hausaufgabe: Resolution von Jane

Axiome:

1. $\neg K \wedge E \rightarrow M$
2. $B \rightarrow E$
3. $\neg B \rightarrow F$
4. $\neg M$
5. $\neg K$

Hypothese:

6. F

Resolutionsbeweis:

7. B (aus 3, 6)
8. E (aus 2, 7)
9. $K \vee M$ (aus 1, 8)
10. K (aus 4, 9)
11. $\square$ (aus 5, 10)

Axiome KNF:

1. $K \vee \neg E \vee M$
2. $\neg B \vee E$
3. $B \vee F$
4. $\neg M$
5. $\neg K$

Hypothese negiert, KNF:

6. $\neg F$

Hinweis: Set-of-Support Strategie
Mindestens eine Klausel in jedem Schritt stammt von der Hypothese ab.

Zur Vorlesung

Hausaufgabe

- Bearbeiten Sie Übung: Primzahlen

Zusammenfassung

- Rückblick/Wiederholung
- Prädikatenlogik 1. Stufe
  - Syntax
  - Semantik

Feedback

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

Ziele Vorlesung 19

- Rückblick/Wiederholung
- Unerfüllbarkeit für Prädikatenlogik
- Unifikation
- Resolution

Rückblick/Wiederholung

- Prädikatenlogik 1. Stufe
  - Syntax
    - Terme
    - Atome
    - Formeln (mit $\forall, \exists$)
  - Semantik
    - Interpretation: $\langle U, I \rangle$
    - Universum
    - Funktionssymbole werden als Funktionen interpretiert
    - Prädikatssymbole werden als Prädikate interpretiert

Zur Vorlesung

Hausaufgabe

- Bearbeiten Sie einige der Aufgaben aus der Aufgabensammlung.

Zusammenfassung

- Rückblick/Wiederholung
- Unerfüllbarkeit für Prädikatenlogik
- Unifikation
- Resolution

Feedback

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

Ziele Vorlesung 19

- Rückblick/Wiederholung
- Übung

Rückblick/Wiederholung

- Unerfüllbarkeit für Prädikatenlogik
- Klausellogik
- Unifikation
- 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. $m(maus)$

- 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, 5)
  14. $\neg h(tier)$ (1,13)
  15. $k(tier)$ (5,14)
  16. $\square$ (15,11)

Hausaufgabe

Zusammenfassung

- Rückblick/Wiederholung

Feedback

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

Ziele Vorlesung 21

Übungsklausur

Feedback

- Gerne direkt wie immer
- Auch gerne anonym per Web-System

Viel Glück für die Klausuren!

Ende