1 Überblick

Formale Grundlagen der Informatik I

Herbstsemester 2012

Robert Marti

teilweise basierend auf Vorlagen von Prof. emer. Helmut Schauer

Theorie und Praxis

- Formale Grundlagen bedeutet
  - **Mathematik**
  - **Theorie**
  - **Modellbildung**
  - **Abstraktion**: Konzentrieren auf das Wesentliche, Verstecken von Details
  - aber: keine Theorie ohne Anwendung!

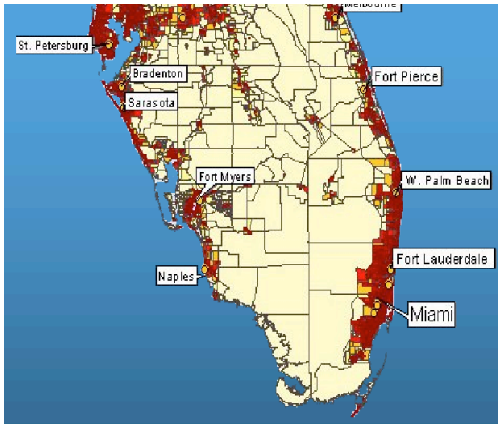
> "There is nothing more practical than a good theory."
> -- James C. Maxwell

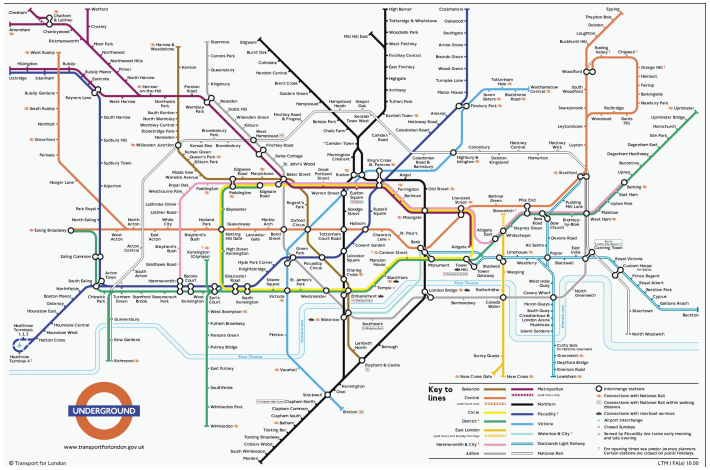
> "In theory, there is no difference between practice and theory.
> But in practice, there is."
> -- Jan L.A. van de Snepscheut

Modelle der Realität

- (Kinder-) Spielsachen
- Landkarten
- Baupläne
- Modelle in der Physik
- Modelle in der Oekonomie

- Diagramme von Computerprogrammen
- Datenmodelle





Inhalt (ohne 100% Gewähr, insbesondere bezügl. Reihenfolge)

1. Überblick
2. Informationstheorie
3. Boole'sche Algebra und Aussagenlogik
4. Prädikatenlogik
5. Eigenschaften binärer Relationen
6. Baumstrukturen
7. Graphen
8. Aufwandabschätzung und Komplexität von Algorithmen
9. Syntaxanalyse
10. Programmentwicklung und -verifikation

Literatur

Clemens Cap:

Theoretische Grundlagen der Informatik,

Springer-Verlag Wien, 1993.

(vergriffen, relevante Kapitel sind auf der Web Seite)

Folien der Vorlesung (auf Web Site)

Uwe Schöning:

Ideen der Informatik, Grundlegende Modelle und Konzepte der Theoretischen Informatik, 3. korrigierte Auflage,

Oldenbourg-Verlag München, 2008.

(Es wird nicht erwartet, dass Sie dieses Buch erwerben.)

Organisation

- Web Site: http://www.ifi.uzh.ch/ee/teaching/hs2012
  (Link auch auf OLAT-Startseite des Kurses)
- Dozent: Robert Marti, robert.marti@acm.org
- Assistentin: Viviane Cantaluppi, viviane@ifi.uzh.ch
- Tutor: Alessandro Rigamonti, alessandro.rigamonti@gmx.net

- Übungen: online Übungen mit OLAT
- Einschreibung: http://www.olat.uzh.ch
- - Es wird voraussichtlich 9 Übungen geben, die maximal je 10 Punkte ergeben.
  - Die Bearbeitungszeit beträgt in der Regel 60 min.
  - Sie haben maximal 3 Versuche, wobei der letzte Versuch zählt.
  - $^2/_3$ der Übungen je Hälfte der Punkte erreicht
    $\Rightarrow$ ¼ Note Bonus bei genügender Leistung (Note ≥ 4)

Assessment Prüfung

- Prüfung: **Mittwoch 19.12.2012** (genaue Zeit wird noch bekannt gegeben)

  Bem.: Dies ist während der letzten Semesterwoche und gilt für alle Studierenden.
  Am Montag 17.12.2012 findet keine Vorlesung mehr statt.

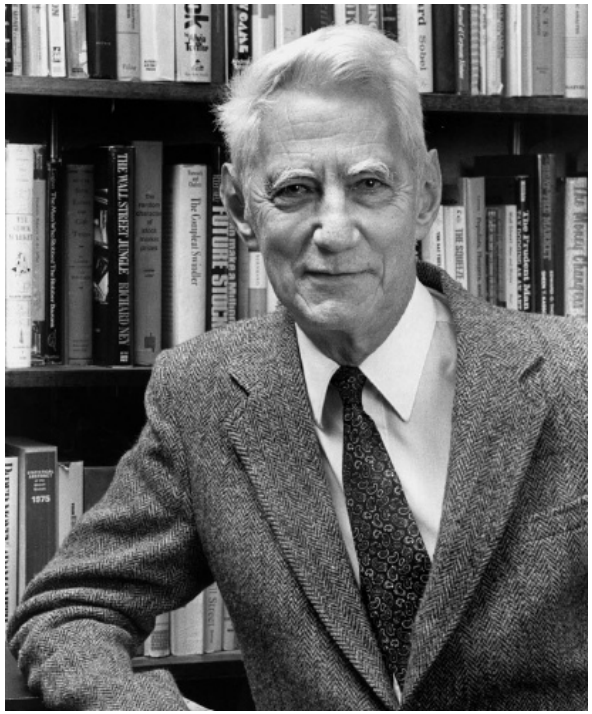
- Die Anmeldung zur Prüfung hat bis spätestens
  Freitag 12.10.2012
  im Modulbuchungstool zu erfolgen

- weitere Informationen folgen

► Informationstheorie

- Grundidee
  - Messen des Informationsgehalts von Daten bzw. Nachrichten
  - z.B. mittlerer Informationsgehalt einer Datenmenge, deren Elemente m verschiedene Werte annehmen können:

$$H = \sum_{i=1}^{m} (-p_i \cdot \log_2 p_i)$$



Claude Shannon
1916 – 2001

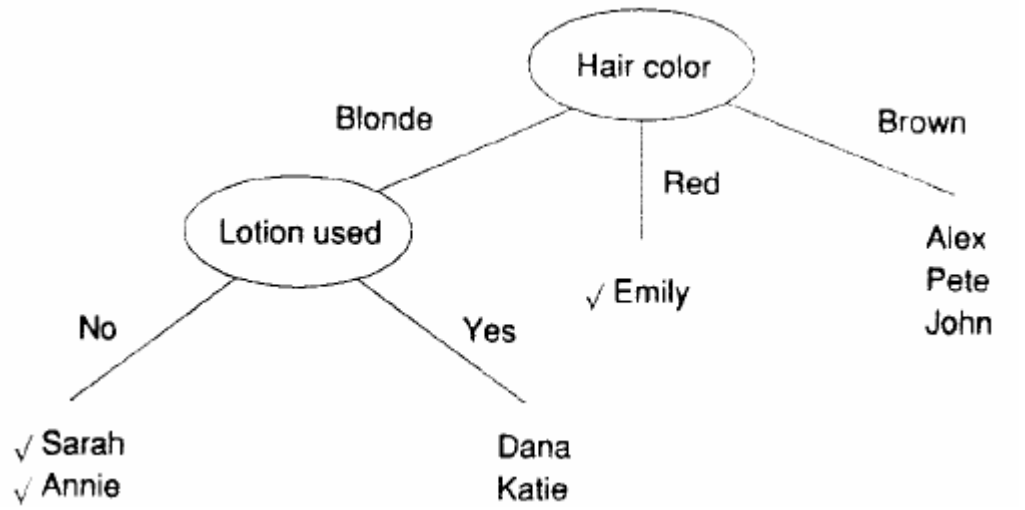
- Anwendungen
  - Codierung von Daten (z.B. Kompression)
  - Maschinelles Lernen:
    Konstruktion von Entscheidungsbäumen

Beispiel: Lernen von Entscheidungsbäumen aus Daten

- Daten

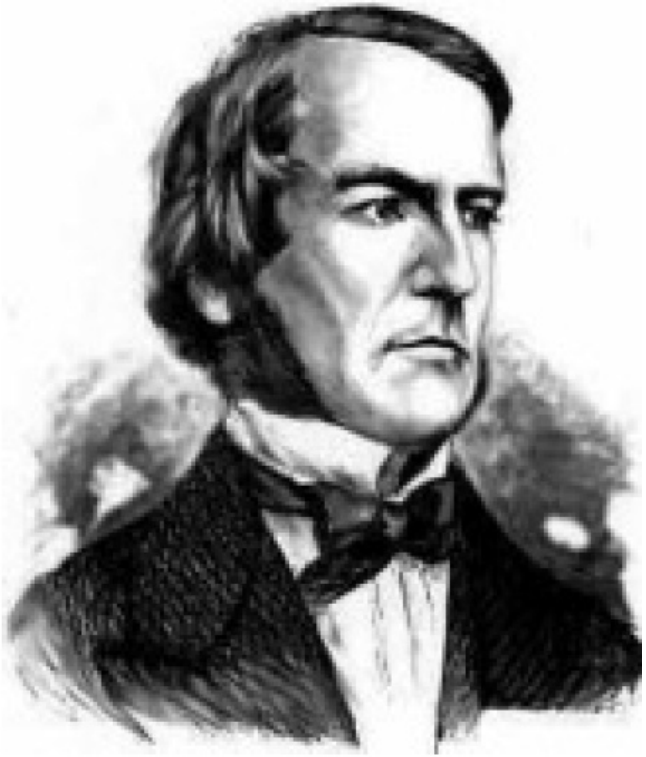
Name	Hair	Height	Weight	Lotion	Result
Sarah	blonde	average	light	no	sunburned
Dana	blonde	tall	average	yes	none
Alex	brown	short	average	yes	none
Annie	blonde	short	average	no	sunburned
Emily	red	average	heavy	no	sunburned
Pete	brown	tall	heavy	no	none
John	brown	average	heavy	no	none
Katie	blonde	short	light	yes	none

- Baum



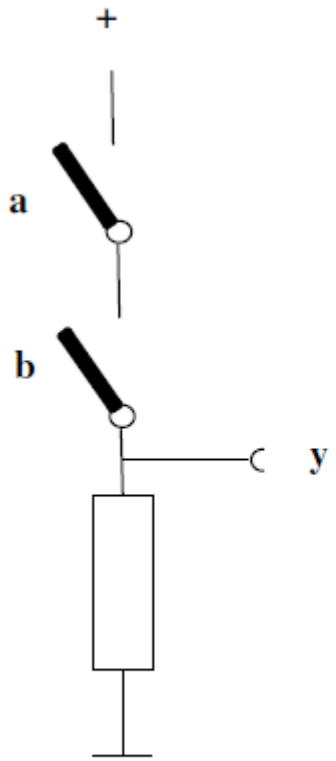
► Boole'sche Algebra und Aussagenlogik

- Grundidee
  - Rechnen mit den logischen Werten "falsch"/"false" und "wahr"/"true" bzw. mit binären Ziffern 0 und 1 (**bi**nary digi**t**s = **bit**s)
- Anwendungen
  - Elektrische Schaltungen (Hardware)
  - Programmsteuerung (Software)
  - Kryptographie (z.B. Klartext und Chiffrierschlüssel als Bitstrings, auf denen Operationen wie XOR angewendet werden)



George Boole
1815 – 1864

Konjunktion (AND)



Serienschaltung

$$y = a \wedge b$$

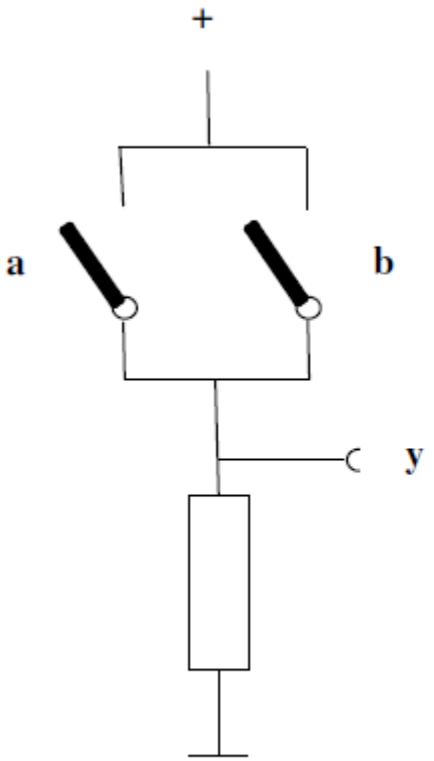
a	b	y
0	0	0
0	1	0
1	0	0
1	1	1

$0 \wedge 0 = 0$
$0 \wedge 1 = 0$
$1 \wedge 0 = 0$
$1 \wedge 1 = 1$

$x \wedge 0 = 0$
$x \wedge 1 = x$ (neutrales Element 1)
$x \wedge x = x$ (Idempotenz)

a, b und y stehen für atomare Aussagen

Disjunktion (OR)



Parallelschaltung

a	b	y
0	0	0
0	1	1
1	0	1
1	1	1

$$y = a \vee b$$

$0 \vee 0 = 0$
$0 \vee 1 = 1$
$1 \vee 0 = 1$
$1 \vee 1 = 1$

$x \vee 0 = x$ (neutrales Element 0)
$x \vee 1 = 1$
$x \vee x = x$ (Idempotenz)

a, b und y stehen für atomare Aussagen

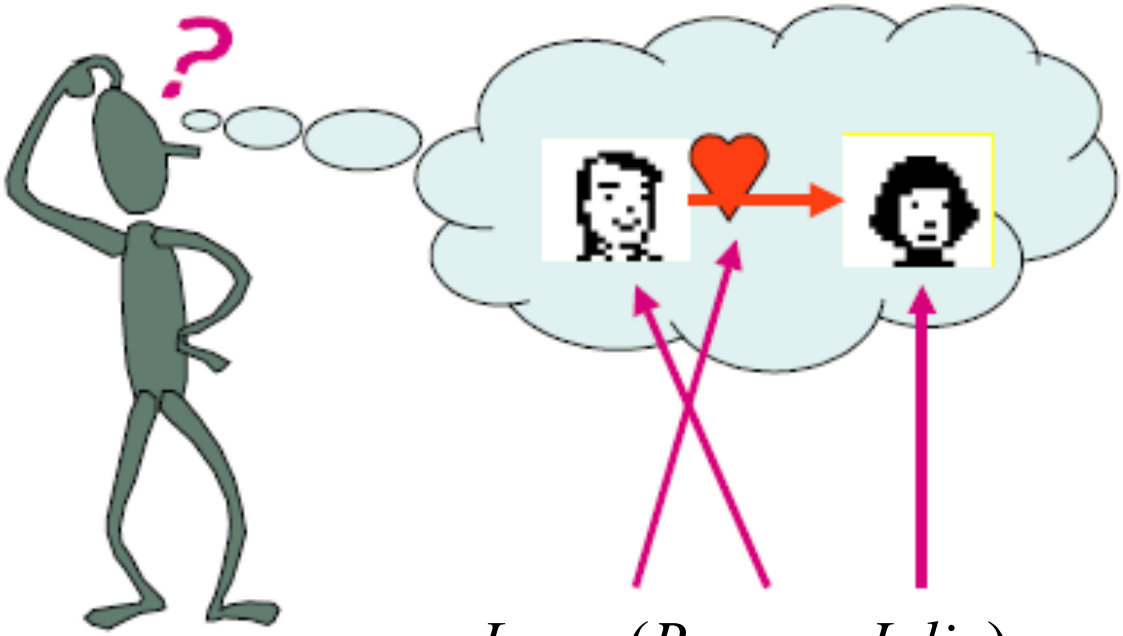
► Prädikatenlogik

- Grundidee
  - Eine Erweiterung der Aussagenlogik, wobei ein Prädikat als "parametrisierbare" atomare Aussage aufgefasst werden kann
- Anwendungen
  - Datenbank-Anfragespachen (Quel, Datalog)
  - Deklarative Programmierspachen (Prolog und weniger bekannte Derivate)
    Deklaratives Programmieren bedeutet, dass nur das "Was" spezifiziert wird, nicht aber das "Wie"

Gottlob Frege
1848 – 1925

  - Wissensrepräsentation, z.B. Web Ontology Language OWL im Semantic Web
  - automatisches Beweisen mathematischer Sätze (Theorem Proving)
  - Repräsentation natürlicher Sprache (Natural Language Processing)

Atomare Aussagen und Quantoren



atomare Aussage mit Parametern

Existenzquantor

Allquantor $\forall x\,(\,Man(x) \rightarrow \exists y\,(\,Woman(y) \wedge Loves(x, y)\,)\,)$

bzw. $\textbf{all}\ x\,(\,Man(x) \rightarrow \textbf{exists}\ y\,(\,Woman(y)\ \textbf{and}\ Loves(x, y)\,)\,)$

► Eigenschaften binärer Relationen

- Grundidee
  - Eigenschaften binärer Relationen,
    z.B. =, <, ≤, ⊂, ⊆, "*PreconditionFor*", "*Loves*", …

    z.B. Symmetrie oder Asymmetrie, Transitivität,
    Existenz teilweiser oder vollständiger Ordnung

  Bem.:
  binäre Relationen können auch als Prädikate mit 2 Argumenten aufgefasst werden

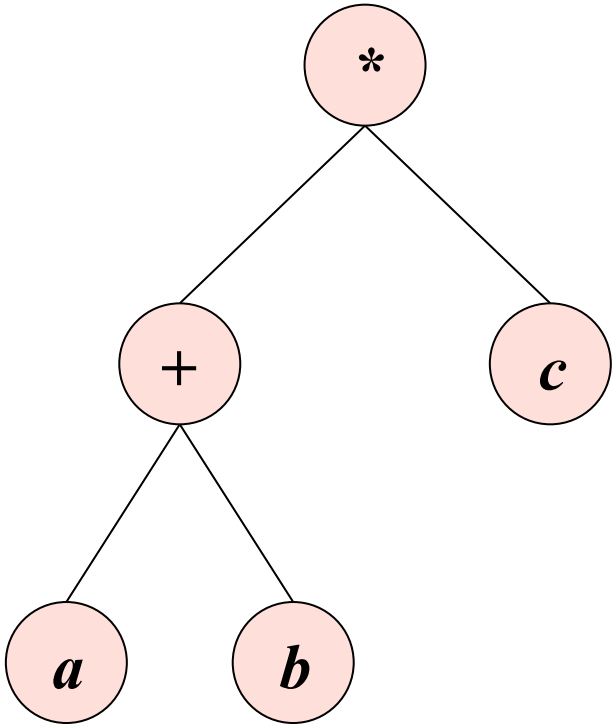
- Anwendungen
  - z.B. bei Beweisen mathematischer Sätze

► Baumstrukturen

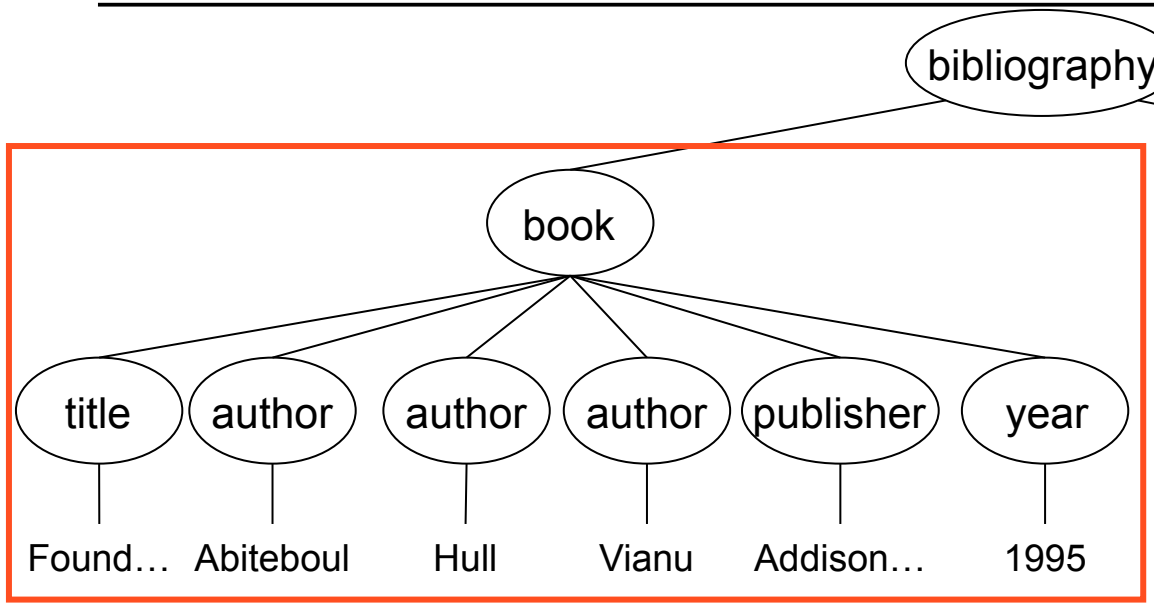
- Grundidee
  - Abstraktion von Hierarchien
- Anwendungen
  - biologische Arten
  - Stammbaum
  - Organisationen
  - Operatorbäume
  - binäre Suchbäume, Mehrweg-Suchbäume
  - Entscheidungsbäume (siehe frühere Folie)

Arithmetischer Ausdruck als Baum

- Beispiel: $(a + b) \cdot c$ bzw. $(a + b) * c$
- Darstellung als Baum
  - Variablen sind Knoten
  - Operatoren sind ebenfalls Knoten, deren Unterbäume sind Operanden
- Traversieren des Baums
  - preorder: $* + a\ b\ c$
    in der Sprache Lisp: `(* (+ a b) c)`
  - inorder: $(a + b) * c$
  - postorder: $a\ b + c\ *$
    z.B. HP Calculators (UPN bzw. RPN), Sprachen Forth, Postscript



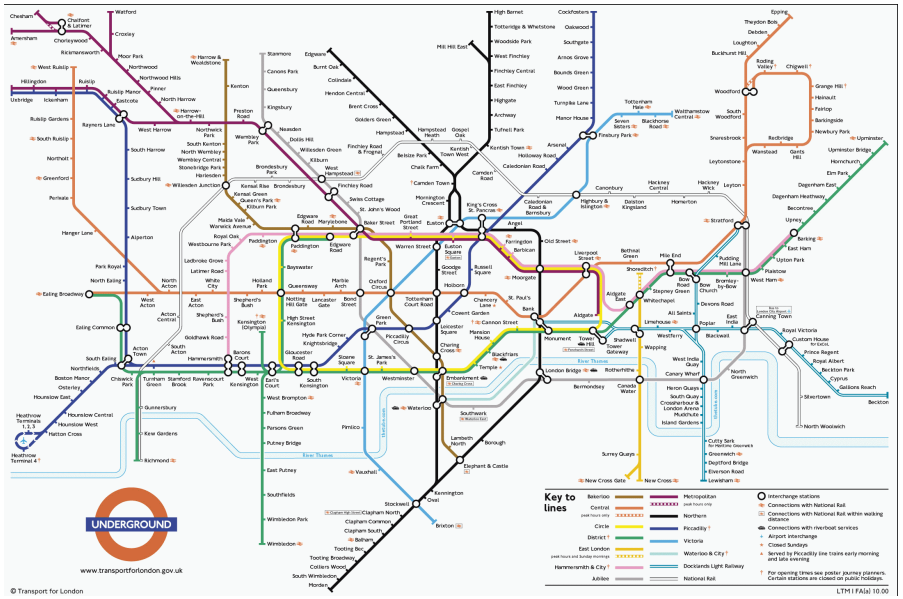
XML (Daten-) Dokument als Baum




```
<bibliography>
  <book>
    <title>
       Foundations of Databases
     </title>
    <author>Abiteboul</author>
    <author>Hull</author>
    <author>Vianu</author>
    <publisher>
       Addison Wesley
     </publisher>
    <year>1995</year>
  </book>
  <book>
    ...
   </book>
</bibliography>
```

► Graphen

- Grundidee
  - Abstraktion von Netzwerken
  - Beantworten von Fragen wie Verbundenheit, Zyklen, Distanzen, ...

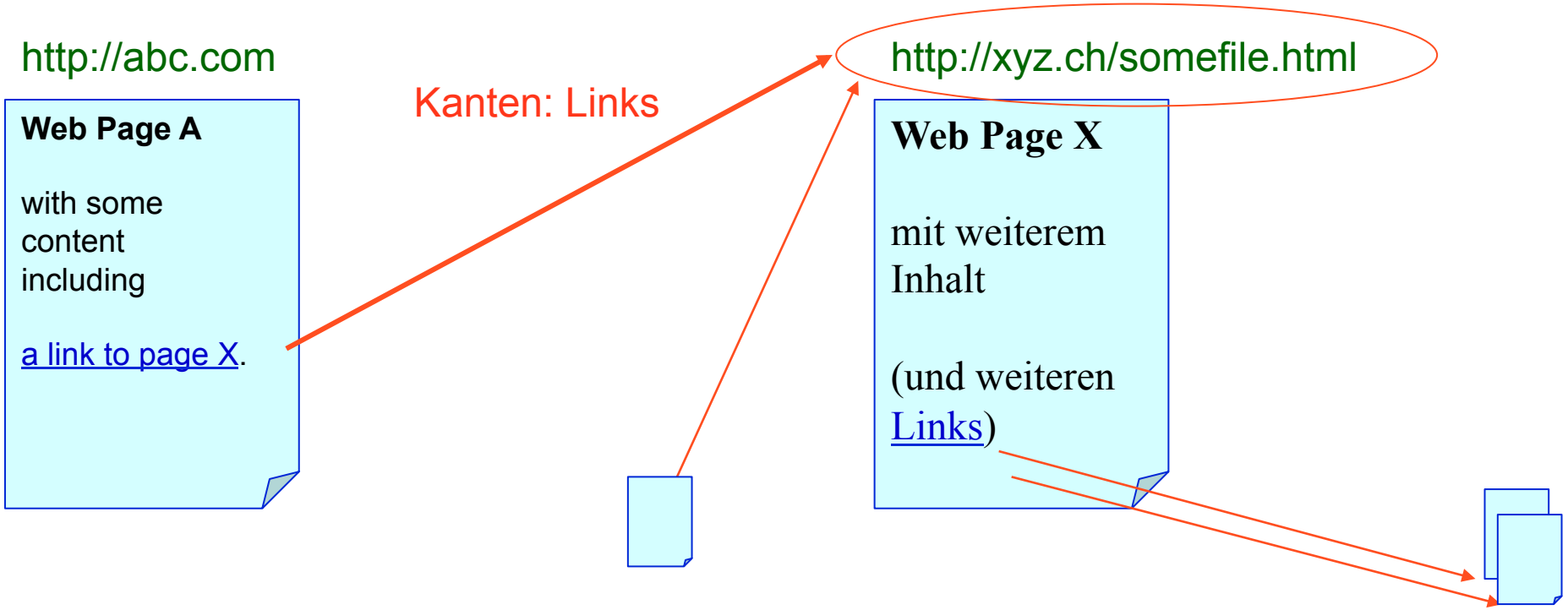


- Anwendungen
  - Verkehrsnetze / Fahrpläne
  - Abhängigkeitsgraphen, z.B. "cause-effect", Arbeitspläne, Lehrpläne
  - Physische Computernetzwerke (z.B. Maschinen als Knoten)
  - Logische Computernetzwerke (z.B. Web Seiten als Knoten)
  - soziale Netzwerke (bis hin zu facebook)

Beispiel: London Underground

Das World Wide Web als Graph

Knoten: Web Seiten identifiziert durch eindeutige Adressen (URLs)

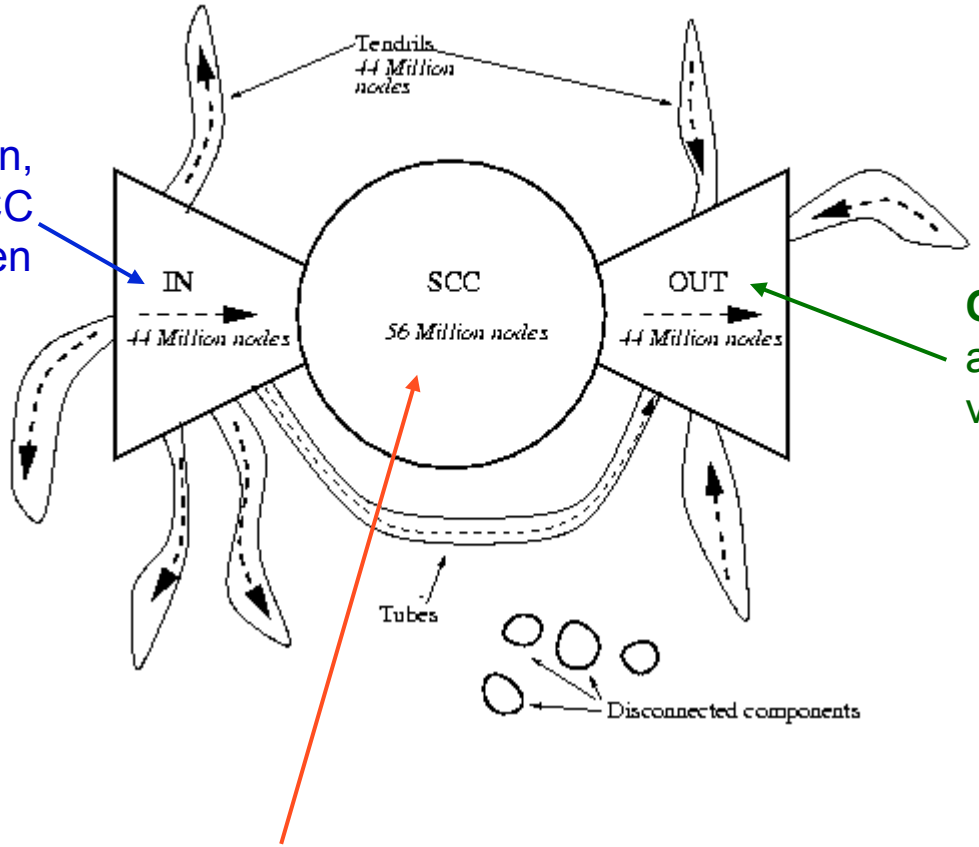


HTML (HyperText Markup Language): Sprache zur Beschreibung von Web Seiten

Grob-Struktur des Web

gemäss Broder 2000

IN: Menge von Knoten, die auf Knoten im SCC verweisen



OUT: Menge von Knoten, auf die von Knoten im SCC verwiesen wird

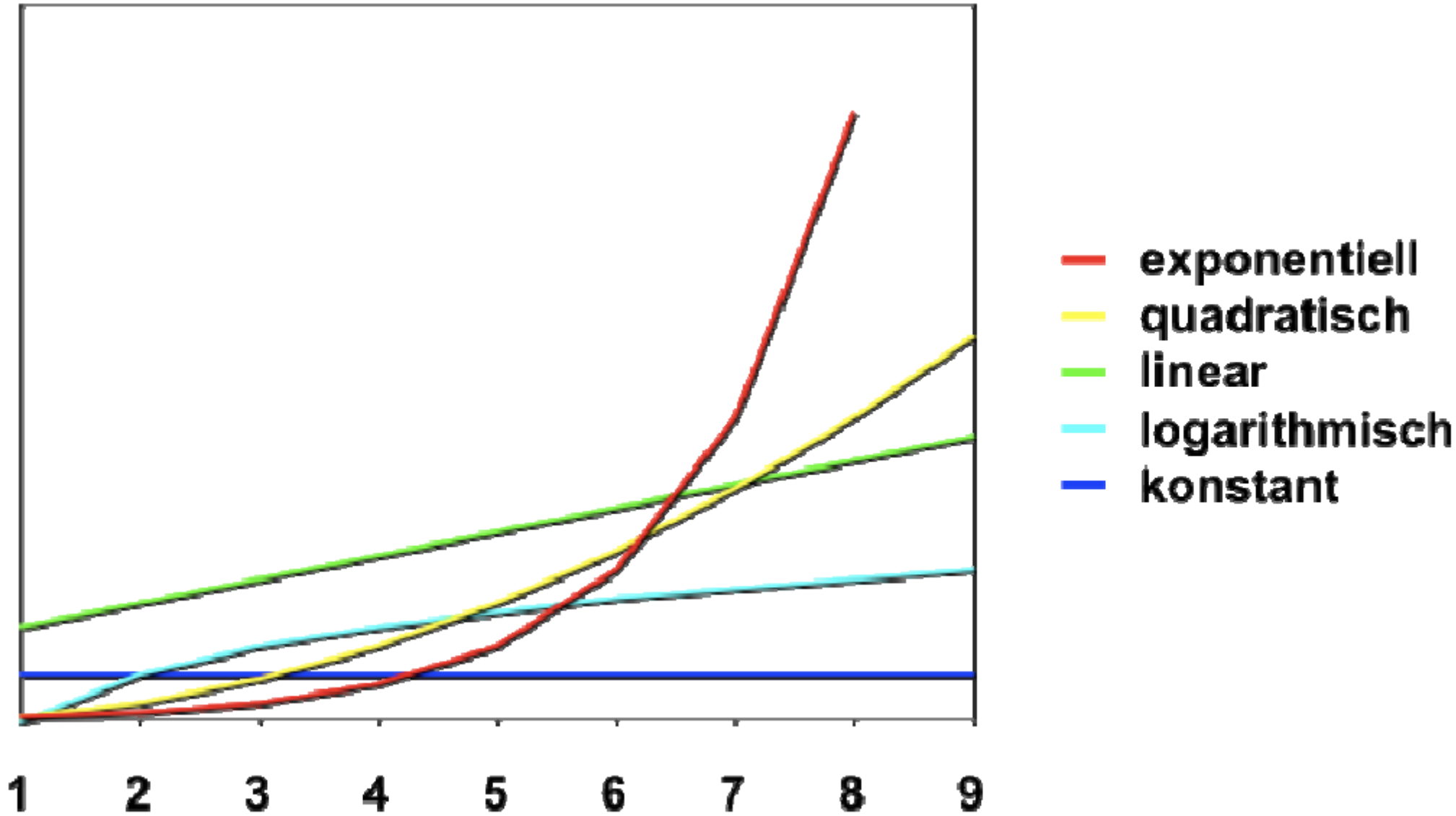
SCC: **S**trongly **C**onnected **C**omponent

Ein gerichteter Graph $G = <V, E>$, wobei V die Menge der Knoten (**v**ertices) und E die Menge der Kanten (**e**dges) ist, heisst **stark zusammenhängend** (strongly connected) falls jeder Knoten $a \in V$ von jedem anderen Knoten $b \in V,\ b \neq a$, erreicht werden kann.

► Aufwandabschätzung / Komplexität von Algorithmen

- Grundidee
  - Analyse der Laufzeit (time complexity)
    und des Platzbedarfs (space complexity)
    von Algorithmen in Abhängigkeit der
    Grösse des Problems
    (will heissen: des Inputs für das Problem)
  - O-Notation
    z.B. quadratisch: $O(n^2)$, linear: $O(n)$, logarithmisch: $O(\log n)$, konstant: $O(1)$
- Anwendungen
  - Normalfall:
    - Entwicklung möglichst effizienter Algorithmen
    - Wahl effizienter Datenstrukturen
  - Kryptologie: Entwicklung von Verschlüsselungsalgorithmen,
    deren Knacken möglichst *ineffizient* (=aufwendig) ist

Komplexitätsklassen

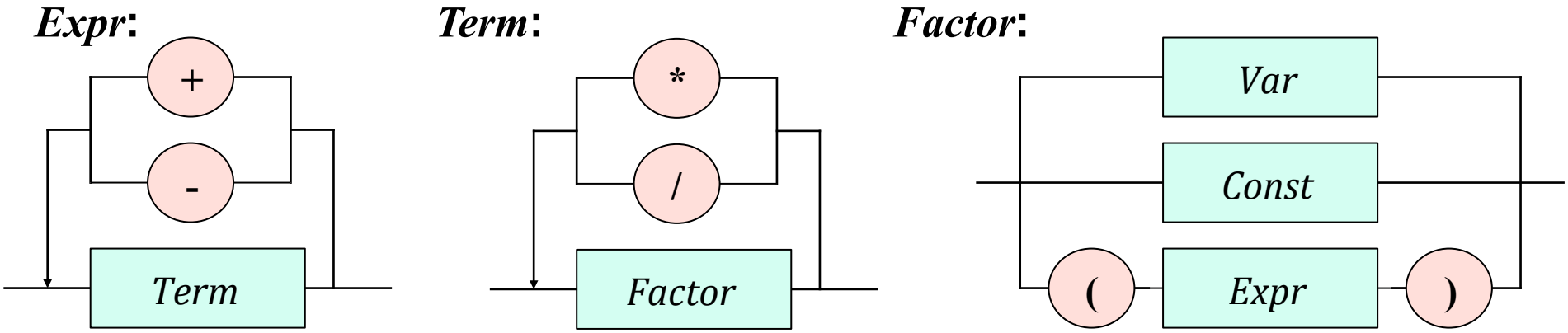


► Syntaxanalyse

- Grundidee
  - Analysieren formaler Sprachen (Programmiersprachen), um Programme in den Befehlssatz eines Computers zu übersetzen (und auszuführen)
  - Analysieren natürlicher Sprache, um Sätze
    . in eine Datenbankanfragesprache zu übersetzen
    . in eine andere natürliche Sprache zu übersetzen
    . den Inhalt zu "verstehen"
- Anwendungen
  - siehe oben

Beispiel: Syntaxdiagramme für arithmetische Ausdrücke

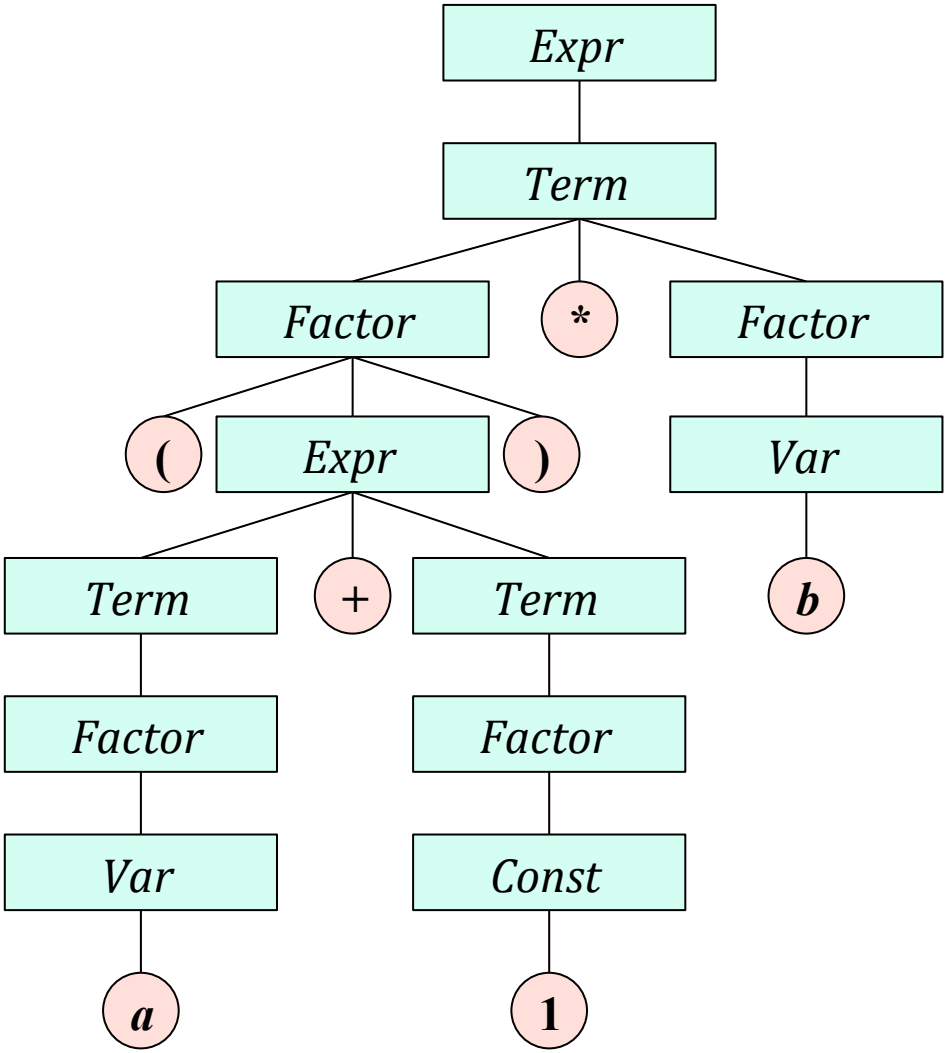
- Ausdruck = expression (***Expr***)



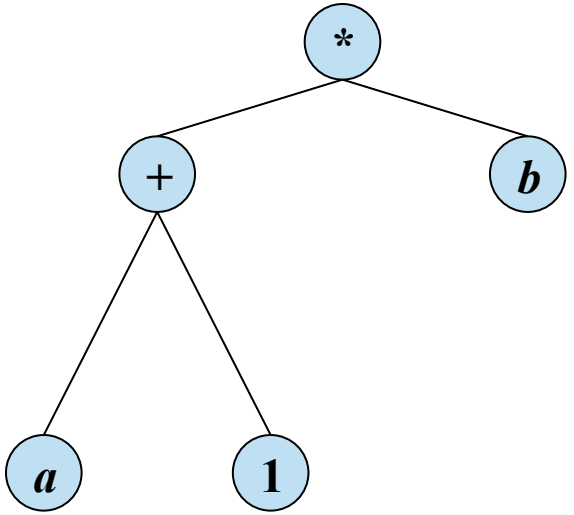
z.B. $(a+1)*b$

Beispiel der Übersetzung eines Ausdrucks

Parsebaum für $(a+1)*b$



Operatorbaum für $(a+1)*b$



SQL Anfragen (in EBNF Notation)

Query =
    Subquery { (`UNION` | `INTERSECT` | `EXCEPT`) [`ALL`] *Subquery* }
    [`ORDER BY` *OrderItem* { "," *OrderItem* }] .

Subquery =
    `SELECT` [`DISTINCT`] (*ColumnSpec* { "," *ColumnSpec* } | "*")
    `FROM` *TableExpr* { "," *TableExpr* }
    [`WHERE` *SearchCondition*]
    [`GROUP BY` *ColumnRef* { "," *ColumnRef* }]
    [`HAVING` *SearchCondition*].

ColumnSpec = *ScalarExpr* [[`AS`] *ColumnName*].

ScalarExpr = *ColumnRef* | *Literal* | … .

ColumnRef = [*TableName* "."] *ColumnName* .

TableExpr = *TableName* [[`AS`] *CorrelationVar*] .

OrderItem = { *ColumnRef* | *Integer* } [`ASC` | `DESC`] .

Beispiel: Übersetzung Englisch → Prädikatenlogik

Beispiel einer stark vereinfachten Grammatik für Englisch: (in DCG Notation)

$s \rightarrow np, vp.$
Ein Satz (s) besteht aus einer noun phrase (np), gefolgt von einer verb phrase (vp).

$np \rightarrow det, noun.$

$vp \rightarrow verb, np.$
$vp \rightarrow verb.$

etc.

Übersetzung eines englischen Satzes in eine prädikatenlogische Formel:

```
| ?- s(F, [every,man,loves,a,woman],[]).

F = all(_14,man(_14)->exists(_31,woman(_31)&loves(_14,_31)))
```

► Programmentwicklung und -verifikation

- Grundidee
  - Spezifikation von Vorbedingungen (preconditions) für Programme und Herleitung der gültigen Nachbedingungen (postconditions)
- Anwendungen
  - Unterstützung bei der Entwicklung bzw. der Überprüfung der Korrektheit von Programmen

Edsger Dijkstra
1930 – 2002

Herausforderung:
Der Beweis der Korrektheit von Programmen inbezug auf Spezifikationen ist oft einfacher als der Beweis der Korrektheit der Spezifikationen.
Ausserdem: Wie sieht's bezüglich **Vollständigkeit** aus?