

Kapitel 11:

Anwendungen der Logik in den Formalwissenschaften

Quellen:

© Inhetveen (Kap. 7),
Pereira/Shieber; Vorlesungsunterlagen
von FAU–Inf2/Schneider,
Inf8/Beckstein [jetzt FSU Jena]



Friedrich-Alexander-Universität Erlangen-Nürnberg
Department Informatik

Inhalt

- **Anwendungen der Logik in den Formalwissenschaften**

... anhand zweier ausgewählter Beispiele:

- Logik und Beweisen in der Mathematik
- Formale Linguistik

Zur Rolle der Logik in den Wissenschaften

- Frommer Wunsch?

Logic is crucial to the understanding of science, for every educated person is trained in logic, as well as in grammar, and these disciplines provide techniques of analysis and a technical vocabulary that permeate scientific writing. At the practical level, logic provides the training necessary for participation in the disputations that are a central feature of instruction, and whose structure - with arguments for and against a thesis, followed by a resolution - is reflected in many written works. The core of the logic curriculum provides the material for the study of types of predication, the analysis of simple propositions and their relations of inference and equivalence, the analysis of modal propositions, categorical and modal syllogisms, fallacies, dialectical topics, and scientific reasoning.

Quelle: Leicht paraphrasierter Einleitungstext zur mittelalterlichen Logik aus der "Routledge Encyclopedia of Philosophy"

Logik und Beweisen in der Mathematik

- Das Ableiten mathematischer Sätze aus Axiomen macht keineswegs den ganzen Sinn des Beweises solcher Sätze aus, sondern ein Beweis soll Gründe für die Wahrheit oder Gültigkeit des bewiesenen Satzes liefern.
(C. Thiel)
- Die Kenntnis von Symbolisierungstechniken und Dialogregeln kann das Verständnis mathematischer Texte erleichtern, insbesondere, wenn die Logik als ein Hilfsmittel verstanden wird, in geregelten Diskursen – Dialogen – über die Geltung von Behauptungen zu unterscheiden.
 - Zur Erinnerung: Gentzens "Natürliches Schließen", orientiert am mathematischen Beweisen.
- Illustration anhand eines Beispiels aus Inhetveen, Kap. 7.2 (dort s.a. weitere).

Logik und Beweisen in der Mathematik

- Beispiel aus einem Lehrbuch der Differential- und Integralrechnung:

„Wenn eine unendliche Folge von Zahlen $a_1, a_2, a_3, \dots, a_n, \dots$ gegeben ist und wenn es eine weitere Zahl g gibt, derart, dass in jedes noch so kleine um g abgegrenzte Intervall alle Zahlen a_n mit Ausnahme von höchstens endlich vielen (d.h. von einem gewissen Wert des Index n an alle) hineinfallen, so sagen wir: die Zahl g ist der Grenzwert der Zahlenfolge $a_1, a_2, a_3, \dots, a_n, \dots$, oder: die Zahlenfolge a_1, a_2, \dots konvergiert gegen g ; in Zeichen:

$$\lim_{n \rightarrow \infty} a_n = g. "$$

$$\Rightarrow \forall g \lim_{n \rightarrow \infty} a_n = g \Leftrightarrow \forall \epsilon > 0 \exists N \forall n > N |a_n - g| < \epsilon$$

**ξ hieß eben noch
g!**



Fortsetzung im Lehrbuchtext:

"Es ist nützlich, sich zu vergegenwärtigen, *dass jede konvergente Zahlenfolge beschränkt ist*, d.h., zu jeder Zahlenfolge a_1, a_2, a_3, \dots , für welche ein Grenzwert ξ existiert, gibt es eine von n unabhängige positive Zahl M , so dass für alle a_n der Folge $|a_n| < M$ gilt.

Aus unseren Definitionen folgt dieser Satz leicht: Sicher gibt es nämlich einen Index N , so dass für $n > N$ immer $|a_n - \xi| < 1$ ist. Unter den N Zahlen $|a_1 - \xi|, |a_2 - \xi|, \dots, |a_N - \xi|$ sei A die größte. Dann dürfen wir $M = |\xi| + A + 1$ setzen. Denn es ist sicher nach der Definition von A die Ungleichung $|a_n - \xi| < A + 1$ für $n = 1, 2, \dots, N$ erfüllt, während für $n > N$ gilt $|a_n - \xi| < 1 \leq A + 1$."

Beispiel aus der Infinitesimalrechnung (Forts.)

Unter der Voraussetzung der Existenz einer konvergenten Folge soll gelten:

$$\forall_{M>0} \exists_n (|a_n| < M)$$

Damit ist die Ausgangsstellung des zu führenden Dialoges klar. Sie lautet

$$\exists_{\varepsilon>0} \forall_N \exists_{n>N} (|a_n - g| < \varepsilon) \rightarrow \forall_{M>0} \exists_n (|a_n| < M)$$

Beispiel aus der Infinitesimalrechnung (Forts.)

- Für diese Ausgangsstellung ist eine Gewinnstrategie zu finden.
- Das **kann** nicht allein mit logischen Mitteln gelingen:

Im konkreten Fall müssen wir noch die Dreiecksungleichung zu Hilfe nehmen.

- Man könnte sie in geeigneter Form als zusätzliche Hypothese des Opponenten in die Anfangsstellung aufnehmen.
- Hier soll der Einfachheit halber der Proponent dort, wo er die Dreiecksungleichung braucht, einfach damit rechnen (s.u. Z. 6 und 8).

Beispiel aus der Infinitesimalrechnung (Forts.)

- Zum Beweis:
 - **P** greift mit ε_0 (= 1, schematisch!?!) an (2).
 - Mit "Sicher gibt es..." wird die Opponentenbehauptung reformuliert.
 - In (4) folgen die Definitionen; dann erfolgt im Text ein Übergang zu (6) bzw. (8) ohne explizite Erwähnung – jedoch Verwendung ! – der Dreiecksungleichung und ... hört einfach auf (wieso geht das?).
 - Die zu zeigende letzte Ungleichung in (4) bleibt im Beweis einfach weg. Im Dialog kann der Beweis jedoch ohne weiteres vervollständigt werden (6) – in der Fallunterscheidung stehen die Verteidigungen, die **P** auf den Angriff in (5) noch schuldig ist.

0	$\bigwedge_{\varepsilon>0} \bigvee_N \bigwedge_{n>N} \cdot a_n - g < \varepsilon.$	$\bigvee_{M>0} \bigwedge_n \cdot a_n < M.$
1	?(0)	...
2	$\bigvee_N \bigwedge_{n>N} \cdot a_n - g < \varepsilon_0.$	$\varepsilon_0?(0)$
3	$\bigwedge_{n>N_0} \cdot a_n - g < \varepsilon_0.$?(2)
4		$A := \max_{1 \leq \nu \leq N_0} a_\nu - g ;$ $M := A + g + \varepsilon_0$ $\bigwedge_n \cdot a_n < M. \quad (\uparrow 1)$
5	$n_1?(4)$	$ a_{n_1} < M$
6	?(5)	<u>1. Fall:</u> $1 \leq n_1 \leq N_0 :$ $ a_{n_1} \leq a_{n_1} - g + g \leq A + g =$ $= M - \varepsilon_0 < M$
7		<u>2. Fall:</u> $n_1 > N_0 :$ $n_1?(3)$
8	$ a_{n_1} - g < \varepsilon_0$	$ a_{n_1} \leq a_{n_1} - g + g < \varepsilon_0 + g \leq$ $\leq A + g + \varepsilon_0 = M$

Rechnen
mit
Dreiecks-
ungleich.

hier
auch!

10

Ein großes Beispiel für das DiaLogic-Programm: Wurzel aus 2 ist irrational

$$\neg \forall_p \forall_q (2 = \frac{p^2}{q^2} \wedge \neg (\forall_s (2s = p) \wedge \forall_t (2t = q)))$$

falls gilt (Hypothesen):

$$\forall_u (\forall_t (2t = u^2) \rightarrow \forall_s (2s = u))$$

$$\forall_x \forall_y \forall_z (2x = y^2 \wedge 2z = y \rightarrow 2z^2 = x)$$

Wurzel aus 2 ist irrational

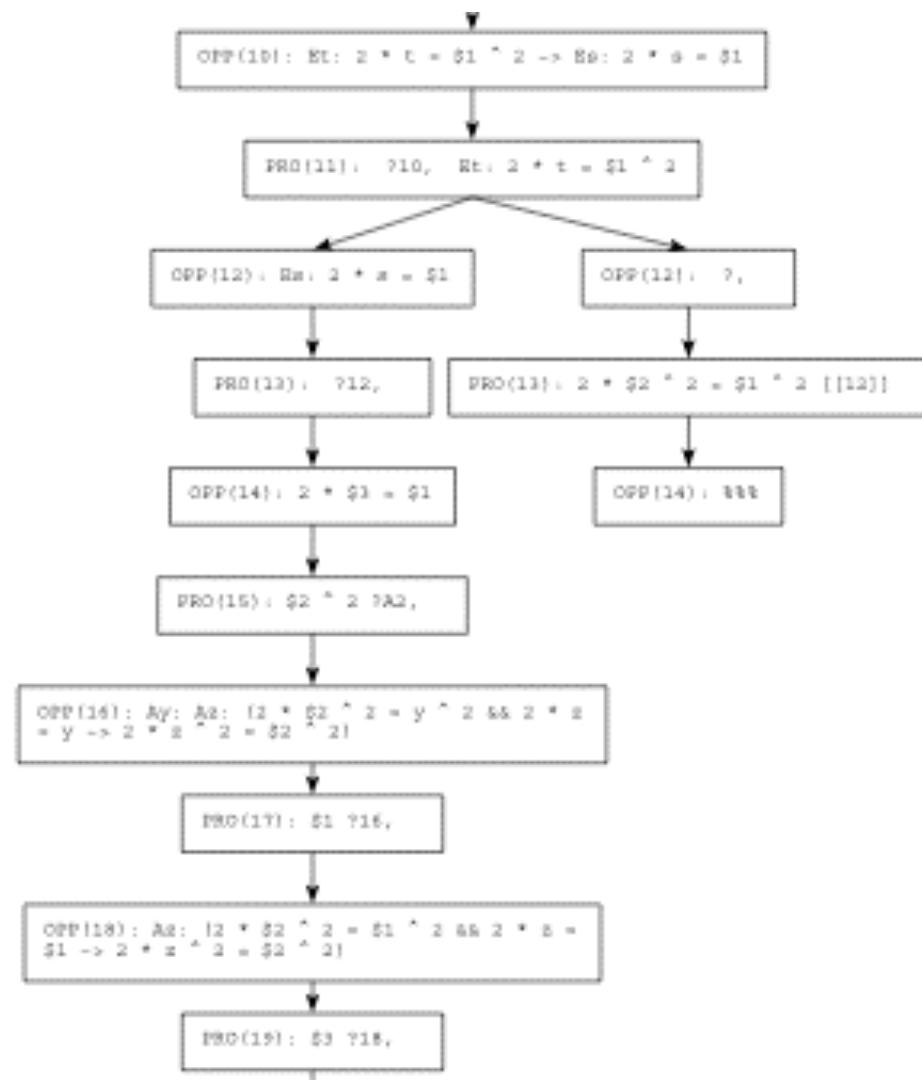
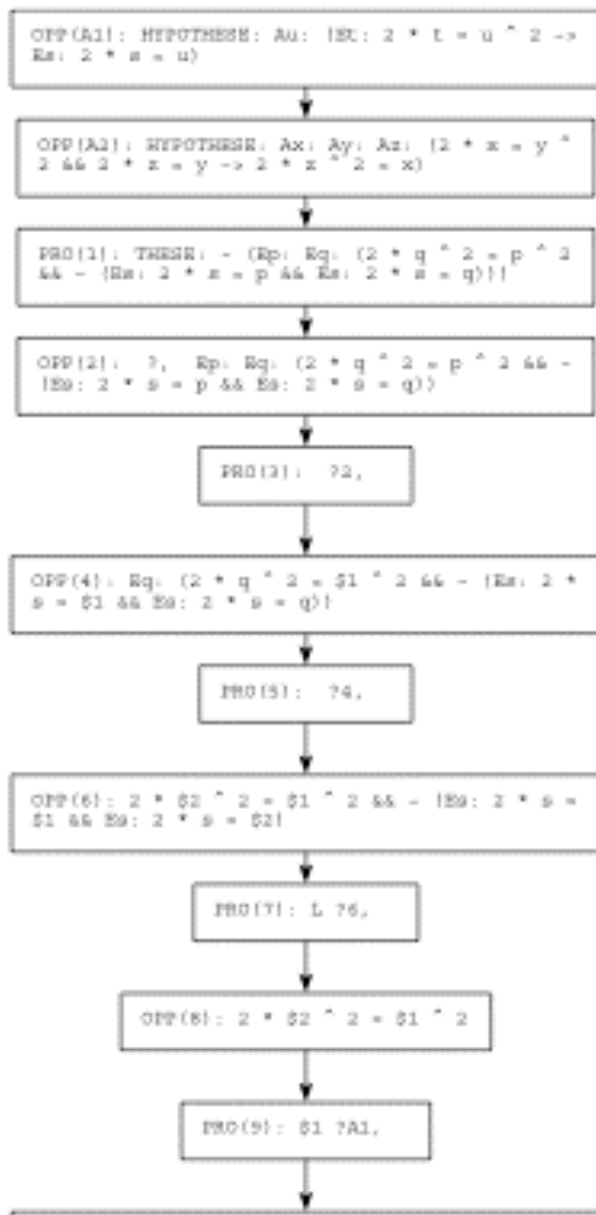
```
; *** Irrationality of sqrt 2
declare sort nat
declare constant 2 nat

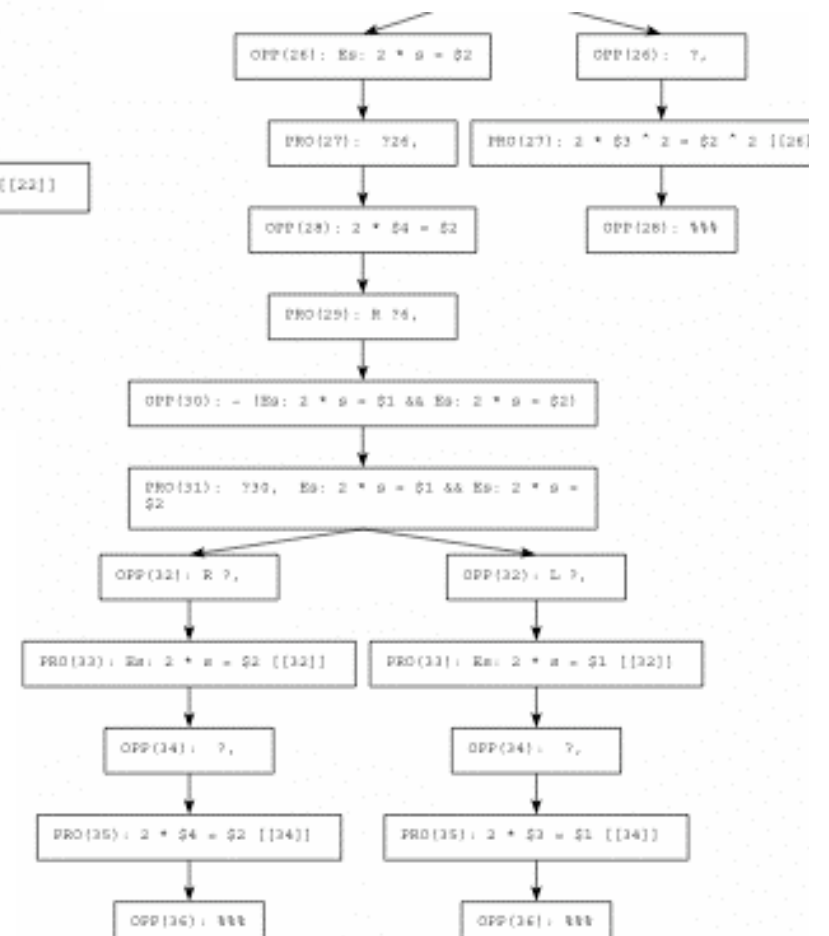
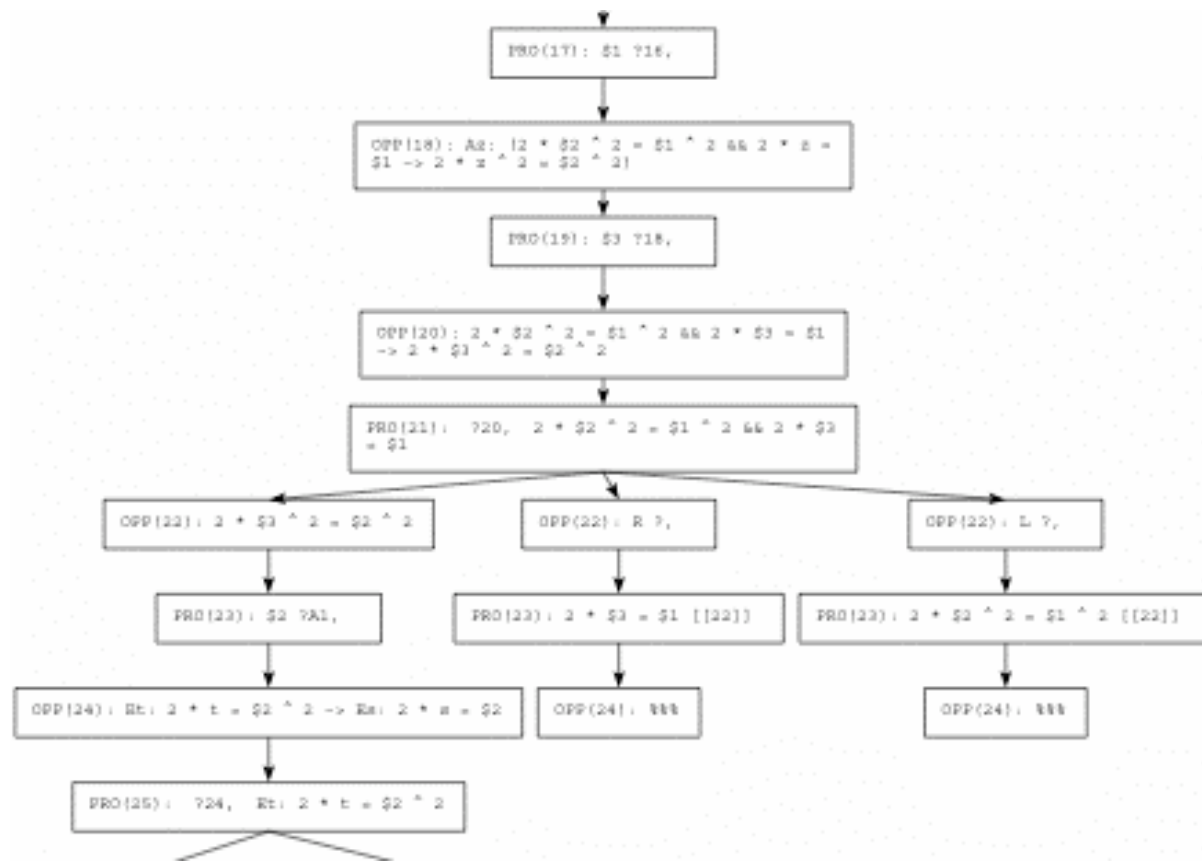
declare variable p nat \ declare variable q nat \ declare variable r nat
declare variable s nat \ declare variable t nat \ declare variable u nat
declare variable x nat \ declare variable y nat \ declare variable z nat

declare function * (nat nat) -> nat infix 3 4 *
declare function ^ (nat nat) -> nat infix 6 5 ^
declare predicate = (nat nat) infix =

hypothesis A1 ALL u ((EXI t 2 * t = u ^ 2) SUB (EXI s 2 * s = u))
hypothesis A2 ALL x ALL y ALL z ((2 * x = y ^ 2 AND 2 * z = y)
SUB 2 * z ^ 2 = x)

thesis NOT EXI p EXI q (2 * q ^ 2 = p ^ 2 AND NOT ((EXI s 2 * s = p)
AND (EXI s 2 * s = q)))
set attack limit 2
```





Formale Linguistik

- Theorie der formalen Sprachen und Automaten
 - vielfältige Anwendungen in der Informatik, z.B. Programmiersprachen-Interpretation und -Übersetzung.
- Anwendung auf "natürliche" Sprachen
 - Ansatz: "English as a Formal Language" (Richard Montague).
- **Linguistische Abstraktionsebenen**
 - **Ordnung (Phonologie, Morphologie, Syntax)** – grammatische Struktur
 - **Inhalt (Semantik)** – textuelle Bedeutung
 - **Gebrauch (Pragmatik und Diskurs)** --- kontextuelle, situierte Bedeutung, u.a.

Beherrschung der Komplexität durch Modularisierung.

- Fundamentales Problem: **Ambiguität** auf allen linguistischen Ebenen.
- Im Folgenden: Fokussierung auf **grammatische Analyse!**

Formale Sprachen

Definitionen:

- Eine **formale Sprache** ist ein System von endlich vielen Zeichen und endlich vielen Regeln, wobei die Regeln in irgendeiner Weise festlegen, welche Zeichenfolgen zur Sprache gehören sollen und welche nicht.
- Eine **Chomsky-Grammatik** ist gegeben durch **$G = (T, N, P, S)$** . Dabei sind
 - T, N endliche Mengen ($T \cap N = \emptyset, V := T \cup N$) [von Zeichen]
 - P eine endliche Teilmenge von $V^* \times V^*$
 - $S \in N$
- Bezeichnungen:
 - T **terminales Alphabet**
 - N **nichtterminales Alphabet**
 - V Alphabet
 - V^* Menge aller aus Elementen von V durch Konkatenation erzeugbaren Zeichenketten (freie Halbgruppe mit ϵ als neutralem Element)
 - P **Produktionen**
 - S **Startsymbol**, Axiom

Noam
Chomsky
(geb. 1928)

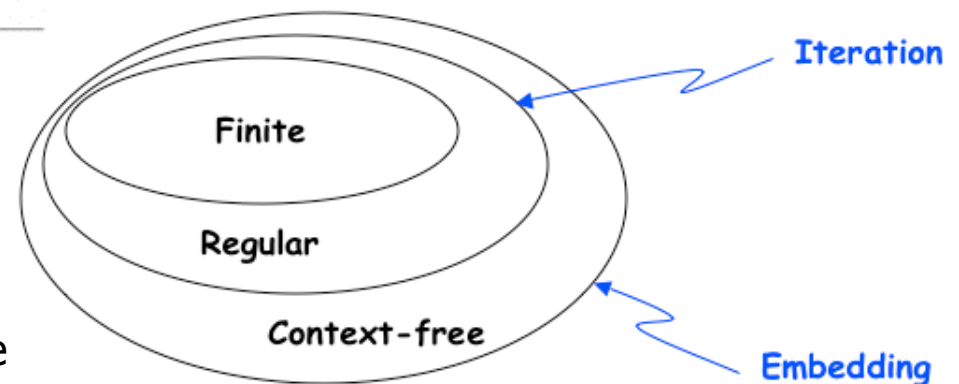


Kontextfreie Grammatik

- Eine allgemeine Chomsky-Grammatik erlaubt, dass mehrelementige Zeichenketten ersetzt werden.
- Eine Chomsky-Grammatik $G = (T, N, P, S)$ heißt **kontextfrei**, wenn auf der linken Seite der Produktionen stets genau ein (nichtterminales) Symbol steht: $P \subset N \times V^*$
- Die kontextfreien Grammatiken entsprechen der **Backus-Naur-Form:**

<i>BNF</i>	<i>Chomsky</i>	<i>oder</i>
$A ::= BCD$	$(A, BCD) \in P$	$A \rightarrow BCD$
$A ::= BC \mid DE$	$(A, BC) \in P$ $(A, DE) \in P$	$A \rightarrow BC$ $A \rightarrow DE$

Ausschnitt aus der
Chomsky-Hierarchie



Beispiel: Kontextfreie Grammatik

- Kontextfreie ("Phrasenstruktur"-) Grammatik für einfache englische Sätze

- Grammatik:

Sentence ::= NounPhrase VerbPhrase

NounPhrase ::= Det Noun OptRel | PropNoun

OptRel ::= that VerbPhrase | ϵ

VerbPhrase ::= Trans Verb NounPhrase | IntransVerb

PropNoun ::= terry | john | mary | shrdlu | ...

Det ::= the | its | a | every | ...

Noun ::= program | fish | woman | man | student | ...

IntransVerb ::= halts | walks | ...

TransVerb ::= writes | eats | sees | loves | ...

← ab hier:

"Lexikalische Regeln"

- Terminal sind in diesem Beispiel alle mit einem Kleinbuchstaben beginnenden Wörter.
- **Beispielsatz:** "every student writes a program that halts"

Ableitbarkeit

Definitionen:

- y heißt **direkt ableitbar** aus x mittels $p \in P$: $x \xrightarrow[p]{}$ y genau dann,

$$(\forall_{w, u, v, z \in V^*}) (x = wuz \wedge y = wvz \wedge p = (u, v))$$

- y heißt direkt ableitbar aus x in G : $x \xrightarrow[G]{}$ y genau dann, wenn

$$(\forall_{p \in P}) (x \xrightarrow[p]{} y)$$

- $x \rightarrow y$, wenn G klar ist.

Ableitbarkeit

Definitionen:

- y heißt **ableitbar** aus x in G : $x \xrightarrow[G]{*} y$, wenn gilt:

$$\bigvee_{n \geq 0} \bigvee_{w_0, w_1, \dots, w_n \in V^*} (x = w_0$$
$$\wedge \bigwedge_{1 \leq i \leq n} (w_{i-1} \xrightarrow[G]{*} w_i)$$
$$\wedge y = w_n)$$

- **Beispiel:**

$S \rightarrow NP \quad VP \rightarrow PN \quad VP \rightarrow \text{terry} \quad VP \rightarrow \text{terry TV NP}$

$\rightarrow \text{terry writes NP} \rightarrow \text{terry writes Det N OptRel}$

$\rightarrow \text{terry writes a N OptRel} \rightarrow \text{terry writes a program OptRel}$

Ableitbarkeit

- Menge der erzeugten Zeichenreihen: $L(G) := \{w \mid w \in T^* \wedge S \xrightarrow[G]{*} w\}$
- $\xrightarrow[G]{*}$ ist **entscheidbar**, $\xrightarrow[G]{*}$ ist für allgemeine Chomsky-Grammatiken **nicht entscheidbar**.
- $\xrightarrow[G]{*}$ ist für kontextfreie Grammatiken effizient entscheidbar.
- **Beispiel 1:**
 $S \rightarrow NP VP \rightarrow PN VP \rightarrow terry VP \rightarrow terry TV NP$
 $\rightarrow terry writes NP \rightarrow terry writes Det N OptRel$

Ableitbarkeit

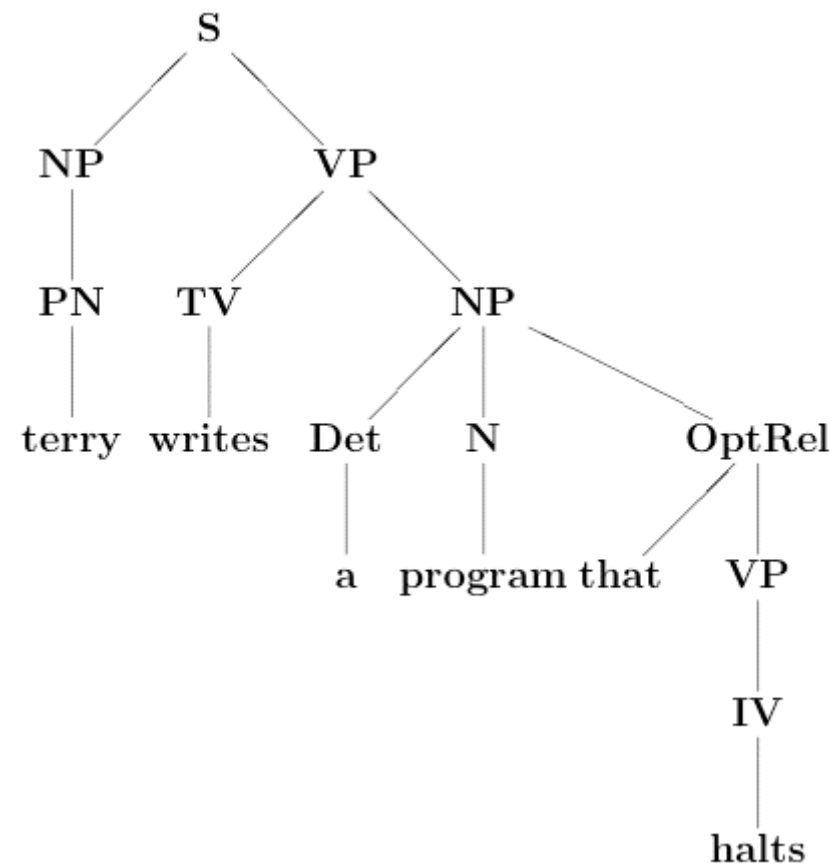
- **Beispiel 2:**

$S \rightarrow NP VP \rightarrow PN VP \rightarrow PN TV NP$

\rightarrow terry TV NP \rightarrow terry writes NP \rightarrow terry writes Det N OptRel

- Eine Ableitung heißt **kanonisch**, wenn immer das am weitesten links stehende nichtterminale Symbol ersetzt wird.
- Ein **Ableitungsbaum** ist eine graphische Darstellung einer Ableitung, wobei die unwesentlichen Mehrdeutigkeiten entfallen.

Ableitungsbaum



Syntaxanalyse mit Prolog

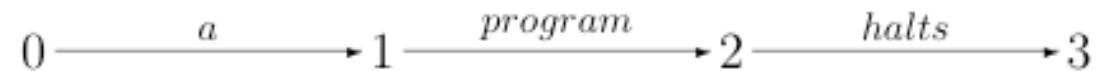
$$NP(p_0, p_1) \wedge VP(p_1, p_2) \rightarrow S(p_0, p_2)$$

$$p_0 \xrightarrow{NP} p_1 \xrightarrow{VP} p_2$$

$$p_0 \xrightarrow{S} p_2$$

Syntaxanalyse mit Prolog

- Beispielsatz 1:



- Codierung mittels connects:

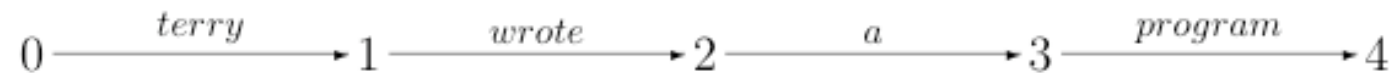
```
connects(a, 0, 1).  
connects(program, 1, 2).  
connects(halts, 2, 3).
```

- Gewünschtes Ergebnis:

```
?- s(0,3).  
Yes  
?- s(0,2).  
No
```

Syntaxanalyse mit Prolog

- Beispielsatz 2:



- Codierung mittels connects:

```
connects(terry, 0, 1).  
connects(wrote, 1, 2).  
connects(a, 2, 3).  
connects(program, 3, 4).
```

- Gewünschtes Ergebnis:

```
?- s(0,4).  
Yes
```

Syntaxanalyse mit Prolog

- Der Satz ist als Sammlung von connects-Fakten in die Datenbasis einzutragen.
Aber: Die beiden Beispiele dürfen **nicht gleichzeitig** dort stehen!
- Regeln zum Satzbau (Pereira/Shieber):

```
s(P0, P2) :- np(P0, P1), vp(P1, P2).
```

- Weitere Regeln (zu diesem einfachen Beispiel):

```
np(P0, P2) :- det(P0, P1), n(P1, P2).
```

```
np(P0, P1) :- pn(P0, P1).
```

```
vp(P0, P2) :- tv(P0, P1), np(P1, P2).
```

```
vp(P0, P1) :- iv(P0, P1).
```

Syntaxanalyse mit Prolog

- Lexikalische Regeln:

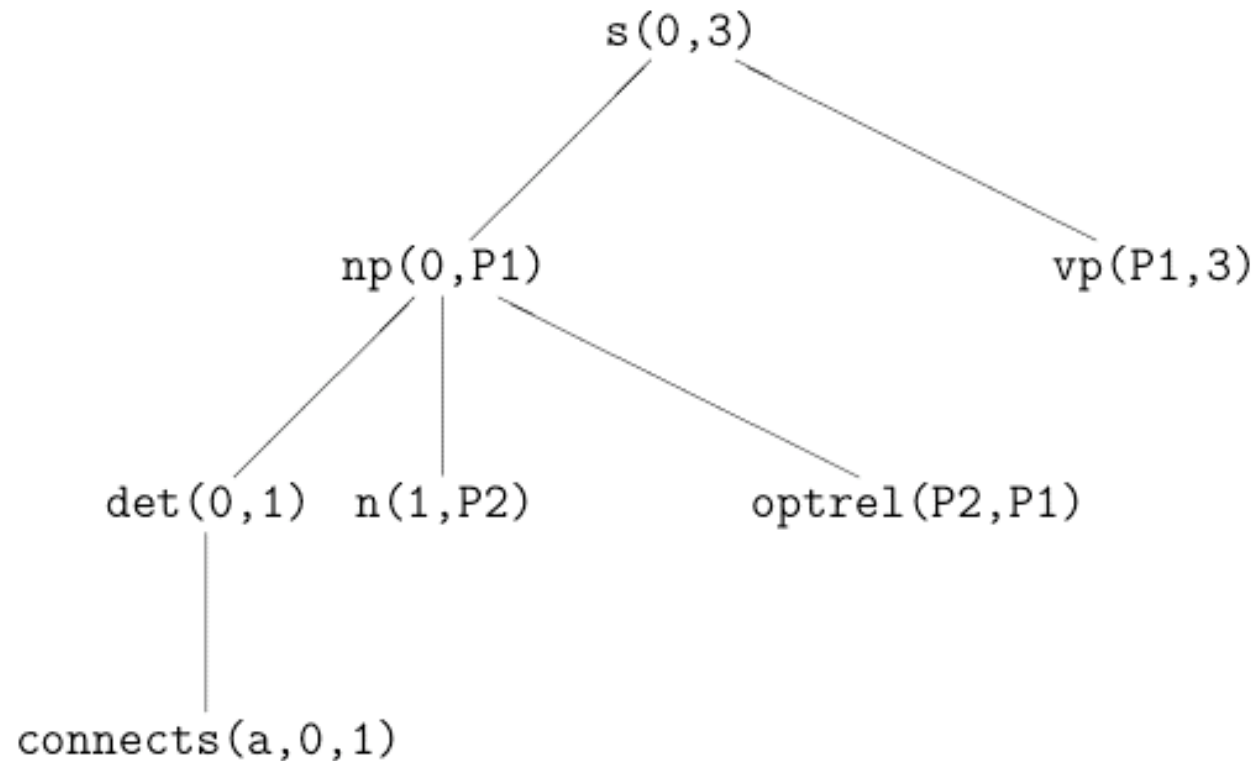
```
pn(P0, P1)      :- connects(terry, P0, P1).  
pn(P0, P1)      :- connects(shrdlu, P0, P1).  
det(P0, P1)     :- connects(a, P0, P1).  
n(P0, P1)       :- connects(program, P0, P1).  
iv(P0, P1)      :- connects(halts, P0, P1).  
tv(P0, P1)      :- connects(wrote, P0, P1).  
tv(P0, P1)      :- connects(writes, P0, P1).
```

Syntaxanalyse mit Prolog

- Anfrage:

```
?- s(0,3).
```

- Ableitungsbaum nach einigen Schritten:



Syntaxanalyse mit Prolog

- Chomsky-Grammatik (BNF):

$$A ::= B C$$

- Prolog:

```
a(P0, P2) :- b(P0, P1), c(P1, P2).
```

- **Neue Interpretation:**

- P0, P1, P2 sind Wortlisten
- Bei Erreichen der Position noch zu verarbeitender Teil des Satzes
- Beispiel:

P0 vorgegebene Liste

P1 Restliste nach Abarbeitung von B

P2 Rest nach Abarbeitung von B und C

- **Zu ändern ist nur das connects-Prädikat:**

```
connects(Word, [Word | Rest], Rest).
```

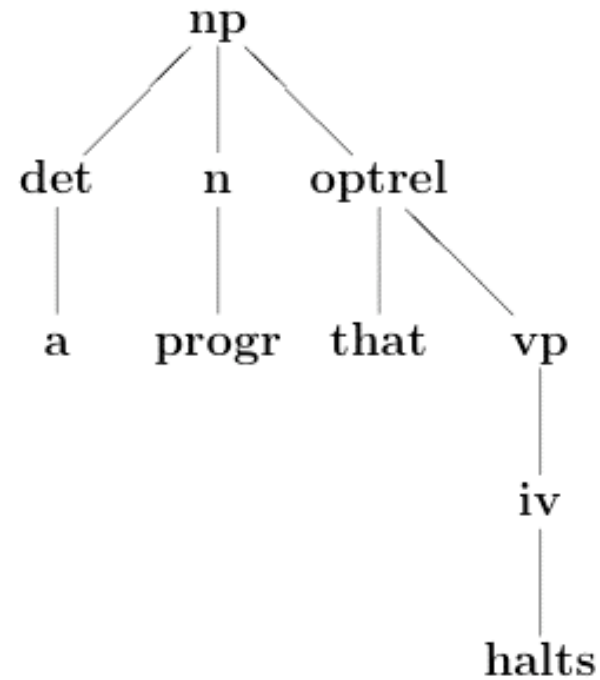
Syntaxanalyse mit Prolog

Beispiel:

- `?- s([a, program, halts], []).`
`s(P0, P2) :- np(P0, P1), vp(P1, P2).`
`P0 = [a, program, halts] P2 = []`
- `?- np([a, program, halts], P1), vp(P1, []).`
`np(P0, P3) :- det(P0, P1'), n(P1', P2), optrel(P2,P3).`
`P0 = [a, program, halts] P3 = P1`
- `?- det([a, program, halts], P1'), n(P1',P2), optrel(P2,P1), vp(P1, []).`
`det(P0,P1'') :- connects(X, P0, P1''), det(X).`
`P0 = [a, program, halts] P1'' = P1'`
- `?- connects(X, [a, program, halts], P1'), det(X), n(P1', P2), optrel(P2, P1), vp(P1, []).`
`connects(Word, [Word | Rest], Rest).`
`X = Word = a Rest = P1' = [program, halts]`
- `?- det(a), n([program, halts], P2), optrel(P2, P1), vp(P1, []).`

Syntaxanalyse mit Prolog

... Ergebnisse als
Strukturbäume:



```
np(det(a),  
    n(program),  
    optrel(that,  
            vp(iv(halts)))  
    )  
)
```

Syntaxanalyse: Kongruenzprüfung

- Grenzen der bisherigen Lösung: Die Grammatik akzeptiert auch "terry write a program that halts"
⇒ Kongruenzprüfung (Kasus, **Numerus**, Genus) !!

- **Wir führen eine weitere Parameterposition ein, die den Numerus angibt.**

```
n(program, sing).      n(programs, plur).
pn(terry, sing).      pn(they, plur).
iv(halts, sing).      iv(halt, plur).
tv(writes, sing).      tv(write, plur).
tv(wrote, _).
det(a, sing).          det(the, _).
```

- **Weitergabe des Numerus an den syntaktischen Regelteil:**

```
pn(N, P0, P1) :- connects(X, P0, P1), pn(X, N).
iv(N, P0, P1) :- connects(X, P0, P1), iv(X, N).
det(N, P0, P1) :- connects(X, P0, P1), det(X, N).
n(N, P0, P1)   :- connects(X, P0, P1), n(X, N).
tv(N, P0, P1)  :- connects(X, P0, P1), tv(X, N).
```

Hier ist auch die Generalisierung bei den lexikalischen Regeln realisiert.

pn/2 ≠ pn/1 !

Syntaxanalyse: Kongruenzprüfung

- **Ergänzung der Syntaxregeln:**

```
s(P0, P2)      :- np(N, P0, P1), vp(N, P1, P2).  
np(N, P0, P3)  :- det(N, P0, P1), n(N, P1, P2), optrel(N, P2, P3).  
np(N, P0, P1)  :- pn(N, P0, P1).  
vp(N, P0, P2)  :- tv(N, P0, P1), np(_, P1, P2).  
vp(N, P0, P1)  :- iv(N, P0, P1).  
optrel(_, P, P).  
optrel(N, P0, P2) :- connects(that, P0, P1), vp(N, P1, P2).
```

Definite Clause Grammar (DCG)

- Das Hinschreiben der Positionsparameter kann automatisiert werden.
- DCG-Notation (wird übersetzt) : Infixoperator ' \rightarrow '

```
s      --> np(N), vp(N).  
np(N)  --> det(N), n(N), optrel(N).  
np(N)  --> pn(N).  
vp(N)  --> tv(N), np(_).  
vp(N)  --> iv(N).
```

Definite Clause Grammar (DCG)

- **Regeln mit Scan-Schritten:**

```
optrel(N, P0, P2 ):- connects(that, P0, P1), vp(N, P1, P2).  
optrel(_, P, P).
```

In der DCG-Schreibweise werden Scan-Schritte durch das entsprechende terminale Symbol in eckigen Klammern dargestellt:

```
optrel(N) --> [that], vp(N).  
optrel(_) --> [].
```

- **Kombination mit Prädikaten, die nicht expandiert werden sollen:**

```
pn(N, P0, P1)      :- connects(X, P0, P1), pn(X, N).  
iv(N, P0, P1)     :- connects(X, P0, P1), iv(X, N).
```

Solche Prädikate werden in geschweifte Klammern eingeschlossen:

```
pn(N) --> [X], {pn(X, N)}.  
iv(N) --> [X], {iv(X, N)}.
```

Computerlinguistik...

- Erweiterung um **grammatische Phänomene**:
 - Verbformen, Hilfsverben
 - Fragesätze: Ja-/Nein-Fragen, wh-Fragen
 - Passiv
 - Allgemeine Relativsätze
 - Lückenkonstruktionen ...
- **Semantische Analyse**: "Übersetzungssemantik" NL → FOL
 - Grundannahme: Kompositionalität der "**logischen Form**" ("Frege-Prinzip").
 - Zuordnung von (funktionalen) Regeln zur Konstruktion logischer Formen zu den Regeln der Grammatik, ausgehend von basalen logischen Formen im Lexikon ("semantischen Lexikoneinträgen").
 - **Sinn und Bedeutung** ("meaning and reference"):
Semantische Auswertung durch **logische Inferenz**.

