Karl Heinz Wagner

Logikgrammatik[1]

0. Vorbemerkung

Mit **Logikgrammatik** ist hier gemeint die Anwendung der Prinzipien der Logikprogrammierung auf die Verarbeitung formaler und natürlicher Sprachen.[2] Solche Anwendungen finden sich mindestens seit 1972 (vgl. Abramson/Dahl, 1989: 1ff) und verbinden sich insbesondere mit der Entwicklung der Programmiersprache Prolog (prOgrammation en logique) durch Alain Colmerauer und seinen Mitarbeitern. Den ersten Schritt machte Colmerauer selbst mit der Einführung der Metamorphose-Grammatiken (Colmerauer, 1978). Ein Variante davon wurde vor allem durch einen Aufsatz von F. Pereira und D. Warren — "Definite Clause Grammars for Language Analysis" — bekannt, der 1980 in der Zeitschrift *Artificial Intelligence* erschien (Pereira/Warren, 1980) und den Ansatz der Logikgrammatik mit dem der ATN-Grammatiken verglich.[3] *Definite Clause* ist eine andere Bezeichnung für *Horn Clause* (Hornklausel) und bezieht sich auf eine besondere Form von Prädikatenlogischen Ausdrücken. *Definite Clause Grammar* (*DCG*) kann also mit *Hornklausel-Grammatik* wiedergegeben werden. Um diesen Grammatikformalismus wird es im folgenden vorrangig gehen. Er hat sich als so fruchtbar erwiesen, daß ein DCG-Compiler, der DCG-Regeln in Prologklauseln übersetzt, fester Bestandteil der meisten Prolog-Implementierungen ist.

1. Kontexfreie Phrasenstruktur-Grammatik: KF-PSG

Wir gehen zunächst aus von einem Fragment einer kontextfreien Phrasenstruktur-Grammatik. Die allgemeine Form einer kontextfreien PS-Regel ist bekanntermaßen $A \rightarrow B$, wobei A ein nichtterminales Symbol aus einem gegebenen Inventar von Kategorialsymbolen und B eine nichtleere Kette von terminalen und/oder nichtterminalen Symbolen ist.

(1.1.) Gegeben sei folgendes Grammatikfragment:

---

[1] Dieser Beitrag ist die erweiterte Fassung des ersten Teils eines zweiteiligen Vortrags über Unifikationsgrammatiken im Rahmen des *Bremer linguistischen Kolloquiums.* Eine überarbeitete Fassung des zweiten Teils, der sich mit der *Lexical Functional Grammar* befaßte, erscheint zu einem späteren Zeitpunkt.

[2] Zu den logischen Grundlagen vgl. Lloyd (1984). Eine zusammenfassende einführende Darstellung der Logikgrammatik ist Abramson/Dahl (1989). Dort findet sich auch weiterführende Literatur.

[3] ATN = *Augmented Transition Network*; vgl. dazu Woods (1973) sowie Bates (1978).

(1) Satz → NP⌢VP
(2) NP → Det⌢N
(3) NP → Name
(4) VP → Vt⌢NP
(5) VP → Vi
(6) Det → *the*
(7) N → *boy*
(8) N → *girl*
(9) N → *ball*
(10) Name → *John*
(11) Name → *Mary*
(12) Vt → *loves*
(13) Vt → *kicked*
(14) Vi → *jumped*
(15) Vi → *laughed*

Solche PS-Regeln können auf unterschiedliche Weise interpretiert werden. Eine Regel wie $Satz \rightarrow NP \frown VP$ kann beispielsweise als Ersetzungsregel verstanden werden, die besagt, daß in einer Symbolkette jedes Vorkommen des Symbols $Satz$ durch die Symbolfolge $NP \frown VP$ ersetzt werden kann. Eine Kette aus terminalen Symbolen wie $John \frown jumped$ gehört dann zu der durch die Grammatik definierten Sprache, wenn sie sich durch sukzessive Ersetzung aus dem Anfangssymbol $Satz$ ableiten läßt, z.B.

$Satz$
$NP \frown VP$ Regel (1)
$Name \frown VP$ Regel (3)
$John \frown VP$ Regel (10)
$John \frown Vi$ Regel (5)
$John \frown jumped$ Regel (14)

Inhaltsreicher ist die Regel $Satz \rightarrow NP \frown VP$ etwa so zu interpretieren: Eine Wortkette z ist ein Satz gdw. sie sich so in zwei Teilketten x und y zerlegen läßt (d.h. $z = x \frown y$), daß x eine Nominalphrase (NP) und y eine Verbalphrase (VP) ist. Oder mit etwas anderer Perspektive: ist x eine Nominalphrase ($NP(x)$) und y eine Verbalphrase ($VP(y)$), dann ($\Rightarrow$) ist die Verkettung von x und y ein Satz ($Satz(x \frown y)$). Diese Aussage läßt sich prädikatenlogisch unmittelbar wie folgt ausdrücken:

$$\bigwedge x \bigwedge y \; (NP(x) \wedge VP(y) \Rightarrow Satz(x \frown y)).$$

Wir können die Grammatik also als ein Axiomensystem $\mathcal{G}$ auffassen, das aus einer Menge von Allaussagen und Einzelaussagen über Wortketten und Wörtern besteht. Eine Wortkette k_i ist dann ein grammatischer Satz, wenn die Aussage $Satz(k_i)$ aus dem Axiomensystem ableitbar ist, d.h. wenn gilt $\mathcal{G} \vdash Satz(k_i)$.

2. Prädikatenlogik erster Stufe: Syntax

Im folgenden soll zunächst die Syntax der Prädikatenlogik etwas genauer dargelegt werden. Die Syntax legt fest, welche Symbolfolgen aus einem Inventar von Grundelementen (dem ALPHABET) wohlgeformt sind und welche nicht.

Definition 2.1. *Alphabet*

Ein ALPHABET besteht aus folgenden Symbolklassen:

1. Individuenvariablen: $x, y, z, x_1 \ldots, y_1 \ldots, z_1 \ldots$
2. Individuenkonstanten: $a, b, c, \ldots, a_1 \ldots, b_1 \ldots, c_1 \ldots$
3. Funktionen: $f, g, h, f_1 \ldots, g_1 \ldots, h_1 \ldots$ mit verschiedener Stelligkeit > 0.
4. Prädikaten: $p, q, r, p_1 \ldots, q_1 \ldots, r_1 \ldots$ mit verschiedener Stelligkeit ≥ 0.
5. Negation: $\neg$
6. Konjunktion und Disjunktion: $\wedge, \vee$
7. Konditional und Bikonditional $\Rightarrow, \Leftrightarrow$
8. Allquantor: $\bigwedge$
9. Existenzquantor: $\bigvee$
10. Interpunktionszeichen: $(,), [,], \{,\}$ und ",".

Zur Bezeichnung von Variablen, Konstanten, Funktionen und Prädikaten werden hier die üblichen Konventionen angenommen. Zur Beschreibung eines bestimmten Gegenstandsbereiches kann es sinnvoll sein, enger am Gegenstand orientierte Repräsentationsformen zu wählen.

Die syntaktischen Bausteine von wohlgeformten Ausdrücken (FORMELN) werden TERME genannt.

Definition 2.2. *Term*

Ein TERM wird induktiv wie folgt definiert:

1. Jede Individuenvariable ist ein Term.
2. Jede Individuenkonstante ist ein Term.
3. Ist f eine n-stellige Funktion und sind $t_1, \ldots, t_n$ Terme, dann ist $f(t_1, \ldots, t_n)$ ein Term.
4. Nur so gebildete Zeichenketten sind Terme.

Beispiele:

x, y (Variable)
a_1, b_2, c_n (Konstante)
$f(a), g(x, f(a, y)), f(a, g(b, h(x, y)), f_1(z))$ (Funktionen)

Ist $f(t_1, \ldots, t_n)$ ein Funktionsterm, so heißen die Terme $t_1, \ldots, t_n$ die ARGUMENTE des Terms. Funktionen sind komplexe Repräsentationen von Individuen. Wofür eine Funktion steht, hängt von der Interpretation des Formalismus bezüglich eines bestimmten Gegenstandsbereiches ab. Beispielsweise ist die Verkettung eine Funktion, die Wortketten auf Wortketten abbildet.

Definition 2.3. *Primformel*

Ist p ein n-stelliges Prädikat und sind $t_1, \ldots, t_n$ Terme, dann ist $p(t_1, \ldots, t_n)$ eine PRIMFORMEL.

Beispiele:

p, q (0-stellige Prädikate)
$p_1(x), r(f(a))$ (1-stellige Prädikate)
$q_2(f(a,b), g(x)), s(x, f(y))$ (2-stellige Prädikate)

Ist $p(t_1, \ldots, t_n)$ eine Primformel, so heißen die Terme $t_1, \ldots, t_n$ die ARGUMENTE der Primformel.

Alle komplexen FORMELN sind aus Primformeln zusammengesetzt:

Definition 2.4. *Formel*

Eine (WOHLGEFORMTE) FORMEL kann induktiv wie folgt definiert werden:

1. Primformeln sind Formeln.
2. Ist F eine Formel so ist auch $(\neg F)$ eine Formel.
3. Sind F und G Formeln, so sind auch $(F \wedge G), (F \vee G), (F \Rightarrow G)$ und $(F \Leftrightarrow G)$ Formeln.
4. Ist F eine Formel und x eine Variable, dann sind $(\bigwedge xF)$ und $(\bigvee xF)$ Formeln.
   Für $(F \Rightarrow G)$ kann auch $(G \Leftarrow F)$ geschrieben werden.

Beispiele:

p, q (0-stellige Primformeln)
$p(x), q(y), r(a)$ (1-stellige Primformlen)
$(\neg p(x))$
$(p(a) \Rightarrow q(b)), (p(x) \Leftrightarrow q(x)), (p(a) \wedge q(b))$
$(\bigwedge x\ p(x)), (\bigvee y (p(y) \wedge q(y)))$

Die Bindungsbereiche der Quantoren ergeben sich aus der in der Definition aufgrund von (3.) verwendeten Klammerung.

Definition 2.5. *Skopus*

Der SKOPUS (Bindungsbereich) von $\bigwedge x$ (bzw. $\bigvee x$) in $\bigwedge x\ F$ (bzw. $\bigvee x\ F$) ist F.

Definition 2.6. *gebunden*

Eine Variable kommt in einer Formel GEBUNDEN vor, wenn sie unmittlbar nach einem Quantor steht (z.B. $\bigwedge x\ p$ oder $\bigvee x\ q$) oder wenn sie im Skopus eines Quantors mit der gleichen Variablen vorkommt (z.B. $\bigwedge x\ p(x,y)$ bzw. $\bigvee y\ (p(y) \wedge q(y))$).

Definition 2.7. *frei*

Eine Variable, die nicht gebunden ist ist frei.

Beispiele:

In der Formel $\bigvee x\ p(x,y) \wedge q(x)$ kommt x die ersten beiden Male gebunden vor, das dritte Mal frei, denn der Skopus von $\bigvee x$ is $p(x,y)$. In $\bigvee x\ (p(x,y) \wedge q(x))$ kommt x nur gebunden vor, weil der Skopus von $\bigvee x$ jetzt $p(x,y) \wedge q(x)$ ist.

Definition 2.8. *Geschlossene Formel*

Sind in einer Formel F alle Variablen durch Quantoren gebunden, gibt es also keine "freien" Variablen, dann wird F als GESCHLOSSENE FORMEL bezeichnet.

Es gelten folgende Bindungsregeln

- ▷ $\bigwedge$ und $\bigvee$ binden stärker als $\neg$
- ▷ $\neg$ bindet stärker als $\wedge$
- ▷ $\wedge$ bindet stärker als $\vee$

Definition 2.9. *Sprache erster Ordnung*

Eine Sprache erster Ordnung bei einem gegebenen Alphabet besteht aus der Menge der Formeln, die aus den Symbolen des Alphabets nach den Regeln der Syntax gebildet werden können.

Beispiele:

(2.1.) $(\bigwedge x(\bigvee y(p(x,y) \Rightarrow q(x))))$

(2.2.) $(\neg(\bigvee x(p(x,a) \wedge q(f(x)))))$

(2.3.) $(\bigwedge x(p(x,g(x)) \Leftarrow (q(x) \wedge (\neg r(x)))))$

sind Formeln. Unter Berücksichtigung der Bindungsregeln können diese wie folgt vereinfacht werden:

(2.4.) $\bigwedge x \bigvee y(p(x,y) \Rightarrow q(x))$

(2.5.) $\neg \bigvee x(p(x,a) \wedge q(f(x)))$

(2.6.) $\bigwedge x(p(x,g(x)) \Leftarrow q(x) \wedge \neg r(x))$

Zur Semantik der Prädikatenlogik sollen hier keine weitergehenden Ausführungen gemacht werden. Es sei lediglich vermerkt, daß die Junktoren $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$ in der Prädikatenlogik die gleiche Bedeutung haben, wie in der Aussagenlogik. Seien P und Q Formeln mit den Werten 1 oder 0, dann soll es eine Funktion $f : F \mapsto \{0,1\}$ geben mit folgenden Eigenschaften:

Regel 1: $f(P \wedge Q) = \begin{cases} 1, & \text{gdw } f(P) = 1 \text{ und } f(Q) = 1 \\ 0, & \text{sonst} \end{cases}$

Regel 2: $f(P \vee Q) = \begin{cases} 0, & \text{gdw } f(P) = 0 \text{ und } f(Q) = 0 \\ 1, & \text{sonst} \end{cases}$

Regel 3: $f(P \Rightarrow Q) = \begin{cases} 0, & \text{gdw } f(P) = 1 \text{ und } f(Q) = 0 \\ 1, & \text{sonst} \end{cases}$

Regel 4a: $f(\neg P) = 0$, gdw $f(P) = 1$

4b: $f(\neg P) = 1$, gdw $f(P) = 0$

3. PS-Grammatik als Axiomensystem im Rahmen der Prädikatenlogik

Man kann, wie oben erwähnt, eine PS-Grammatik als ein Axiomensystem auffassen, das aus einer Menge von Allaussagen und Einzelaussagen über Wortketten und Wörtern besteht. Die prädikatenlogische Interpretation der PS-Grammatik (1.1) sieht beispielsweise wie folgt aus:

(3.1.)

R1: $\bigwedge x \bigwedge y(NP(x) \wedge VP(y) \Rightarrow Satz(x \frown y))$
R2: $\bigwedge x \bigwedge y(Det(x) \wedge N(y) \Rightarrow NP(x \frown y))$
R3: $\bigwedge x(Name(x) \Rightarrow NP(x))$
R4: $\bigwedge x \bigwedge y(Vt(x) \wedge NP(y) \Rightarrow VP(x \frown y))$
R5: $\bigwedge x(Vi(x) \Rightarrow VP(x))$
Lexikon:
$Det(the)$
$N(boy)$
$N(girl)$
$N(ball)$
$Name(John)$
$Name(Mary)$
$Vt(loves)$
$Vt(kicked)$
$Vi(jumped)$
$Vi(laughed)$

Nehmen wir beispielsweise an, es solle gezeigt werden, daß der Ausdruck *the girl laughed* ein grammatischer Satz ist. Dazu müssen wir mit den Regeln und dem Lexikon der Grammatik als Prämissen beweisen, daß die Aussage $Satz(the \frown girl \frown laughed)$ aus der Grammatik $\mathcal{G}$ ableitbar ist, daß also gilt: $\mathcal{G} \vdash Satz(the \frown girl \frown laughed)$. Die formale Logik stellt dafür Mittel zur Verfügung, auf die hier nicht weiter eingegangen werden kann. Zum Verständnis des nachstehenden Beweises genügen folgende Hinweise:

Konjunktion — Sind P und Q Axiome, dann kann die Konjunktion $P \wedge Q$ zur Axiomenmenge hinzugefügt werden.

Allbeseitigung — Da eine allquantifizierte Aussage für alle Individuen eines Individuenbereiches gelten soll, muß sie auch für ein einzelnes Individuum gelten. Ist $\bigwedge x\ p(x)$ ein Axiom, dann kann die Aussage $p(a)$ zur Axiomenmenge hinzugefügt werden, wenn a zum Individuenbereich von x gehört.

Modus Ponens — *Modus Ponens* ist eines der bekanntesten Schlußschemata. Es hat die folgende Form:

$p \Rightarrow q$
p
———
$\therefore\ q$

Ein gültiges Schlußschema geht bei Ersetzung der Aussagenvariablen in einen gültigen Schluß über. Setzen wir beispielsweise p = *Hansi kann fliegen* und q = *Hansi ist ein Vogel*, dann erhalten wir:

Wenn Hansi fliegen kann, ist er ein Vogel
Hansi kann fliegen
———————————————
$\therefore$ Hansi ist ein Vogel

Der Beweis, daß die Wortkette *the girl laughed* ein Satz ist, daß also gilt $Satz(the \frown girl \frown laughed)$, sieht folgendermaßen aus:

(1)	$Det(the)$	Lexikon
(2)	$N(girl)$	Lexikon
(3)	$Det(the) \wedge N(girl)$	(1),(2) Konjunktion
(4)	$Det(the) \wedge N(girl) \Rightarrow NP(the \frown girl)$	R2, Allbeseitigung
(5)	$NP(the \frown girl)$	(3),(4) Modus Ponens
(6)	$Vi(laughed)$	Lexikon
(7)	$Vi(laughed) \Rightarrow VP(laughed)$	R5, Allbeseitigung
(8)	$VP(laughed)$	(6),(7) Modus Ponens
(9)	$NP(the \frown girl) \wedge VP(laughed)$	(5),(8) Konjunktion
(10)	$NP(the \frown girl) \wedge VP(laughed) \Rightarrow Satz(the \frown girl \frown laughed)$	R1
(11)	$Satz(the \frown girl \frown laughed)$	(9),(10) Modus Ponens

4. Das Resolutionsprinzip

Da die Regeln einer PS-Grammatik und damit die äquivalenten prädikatenlogischen Formeln eine bestimmte syntaktische Form aufweisen, stellt sich die Frage, ob es für die Beweisführung in einer Logikgrammatik besonders effektive und effiziente Verfahren gibt.

In der Forschung zum automatischen Beweisen hat in den letzten Jahrzehnten das Schlußschema der RESOLUTION in Verbindung mit der KONJUNKTIVEN NORMALFORM große Bedeutung erlangt. Dieses sog. RESOLUTIONSPRINZIP wurde in den sechziger Jahren von J.A. Robinson (s. ROBINSON 1965) begründet.

Dieses Prinzip beruht, wie der Name schon sagt, auf dem Schlußschema der Resolution:

(4.1.) $p \vee q$
$\neg p \vee r$
———
$\therefore\ q \vee r$

Man bezeichnet die Konklusion beim Resolutionsschema als RESOLVENTE.
Ein möglicher Beweis für die Gültigkeit dieses Schemas sieht folgendermaßen aus:

(1) $p \vee q$	Prämisse
(2) $\neg p \vee r$	Prämisse
(3) $q \vee p$	(1), Kommutativität
(4) $\neg q \Rightarrow p$	(3), Konditional
(5) $p \Rightarrow r$	(2), Konditional
(6) $\neg q \Rightarrow r$	(4), (5), Kettenregel
(7) $q \vee r$	(6), Konditional

Der Schluß ist jedoch auch intuitiv einsichtig: Ist p wahr, dann muß $\neg p$ falsch sein. Wenn nach Voraussetzung aber $\neg p \vee r$ wahr sein soll, und $\neg p$ aber falsch ist, dann muß r wahr sein. Ist andererseits aber p falsch, dann muß, wenn $p \vee q$ wahr sein soll, q wahr sein. Nimmt man beide Möglichkeiten zusammen, so ergibt sich $q \vee r$.

Man kann das Schlußschema des *Modus Tollendo Ponens* als Sonderfall der Resolution auffassen:

(4.2.) $p \vee q$
$\neg p$
———
$\therefore\ q$

Mit anderen Worten, die Resolution von $p \vee q$ und $\neg p$ hat als Resolvente q

(1) $p \vee q$	Prämisse
(2) $\neg p$	Prämisse
(3) $\neg p \vee F$	Addition
(4) $q \vee F$	Resolution
(5) q	Identität

Ein weiterer Sonderfall ist die Resolution von p und $\neg p$, die zum Widerspruch führt. Man könnte wie folgt argumentieren:

(1) p	Prämisse
(2) $\neg p$	Prämisse
(3) $p \vee F$	Addition
(4) $\neg p \vee F$	Addition
(5) $F \vee F$	(3),(4) Resolution
(6) F	Idempotenz

Man bezeichnet die Resolvente von p und $\neg p$ mit $\square$.

Das Schlußschema der Resolution verbindet sich häufig mit dem Verfahren des indirekten Beweises (*Reductio ad absurdum*). Bei einem indirekten Beweis wird die Negation der zu beweisenden Konklusion als vorläufige Prämisse hinzugenommen, und zu zeigen versucht, daß die so entstandene Aussagenmenge zu einem Widerspruch führt. Wenn die einzelnen Ableitungsschritte zulässig sind, dann kommt man zu einem Widerspruch dann, wenn eine oder mehrere Prämissen falsch sind. Da alle ursprünglichen Prämissen als wahr vorausgesetzt werden, muß die als Prämisse hinzugenommene Negation der zu beweisenden Konklusion falsch sein. Somit ist die zu beweisende Konklusion wahr. Betrachten wir dazu folgendes Beispiel:

(4.3.) $p \vee q$
$q \Rightarrow r$
$\neg r$
———
$\therefore\ p$

Bei einem indirekten Beweis geht man folgendermaßen vor:

(1)	$p \vee q$	Prämisse
(2)	$q \Rightarrow r$	Prämisse
(3)	$\neg r$	Prämisse
	(4) $\neg p$	Indirekter Beweis
	(5) q	(1),(4) Modus Tollendo Ponens
	(6) r	(2),(5) Modus Ponens
	(7) $r \wedge \neg r$	(3),(6) Konjunktion
(8)	p	

Zu beweisen ist p. In Zeile (4) wird die Negation $\neg p$ vorläufig zu den Prämissen hinzugenommen. Dies führt in Zeile (7) zu dem Widerspruch $r \wedge \neg r$. $\neg p$ muß daher falsch sein. Somit ist p wahr.

4.1 Hornklauseln und Logikprogramme

Voraussetzung für die fruchtbare Anwendung des Resolutionsschemas ist, daß alle Prämissen Klauselform haben, d.h. Disjunktionen von Literalen sind. Dazu müssen Formeln gegebenenfalls in die konjunktive Normalform übersetzt werden. Zum Verständnis werden im folgenden weitere Begriffe definiert:

Definition 4.1. *Literal*

Ein *Literal* ist eine Primformel oder die Negation einer Primformel.

Beispiele für Literale sind $p(x),\ q(f(a,b)),\ \neg v(x,y)$.

Definition 4.2. *Klausel*

Eine *Klausel* ist eine Formel der Form $\bigwedge x_1 \ldots \bigwedge x_s(L_1 \vee \ldots \vee L_m)$ wobei jedes L_i ein Literal ist und $x_1, \ldots, x_s$ die einzigen Variablen sind, die in $L_1 \vee \ldots \vee L_m$ vorkommen.

Beispiele: Die folgenden Formeln sind Klauseln

(4.4.) $\bigwedge x \bigwedge y \bigwedge z(p(x,z) \vee \neg q(x,y) \vee \neg r(y,z))$

(4.5.) $\bigwedge x \bigwedge y(\neg p(x,y) \vee r(f(x,y),a))$

Für Klauseln gibt es eine besondere Schreibweise, die besonders in der Logikprogrammierung verwendet wird. Man erhält sie, indem man zunächst die Literale nach positiven und negativen Literalen sortiert, z.B.

(4.6.) $\bigwedge x_1 \ldots \bigwedge x_s(A_1 \vee \ldots \vee A_k \vee \neg B_1 \vee \ldots \vee \neg B_n)$

Sei (4.6.) eine Klausel mit den Primformeln
$A_1, \ldots, A_k, B_1, \ldots, B_n$ und $x_1, \ldots, x_s$ als den einzigen darin vorkommenden Variablen, so kann dafür

(4.7.) $A_1, \ldots, A_k \Leftarrow B_1, \ldots, B_n$

geschrieben werden. Dies ergibt sich aus der Äquivalenz von

(4.8.) $\bigwedge x_1 \ldots \bigwedge x_s(A_1 \vee \ldots \vee A_k \vee \neg B_1 \vee \ldots \vee \neg B_n)$

und

(4.9.) $\bigwedge x_1 \ldots \bigwedge x_s(A_1 \vee \ldots \vee A_k \Leftarrow B_1 \wedge \ldots \wedge B_n)$

und der Tatsache, daß alle vorkommenden Variablen allquantifiziert sind, so daß die Quantoren weggelassen werden können.

Definition 4.3. *Programmklausel*

Eine *Programmklausel* ist eine Klausel der Form $A \Leftarrow B_1, \ldots, B_n$ und enthält genau ein positives Literal (A). A ist der *Kopf* und $B_1, \ldots, B_n$ der *Rumpf* der Programmklausel.

Definition 4.4. *Einheitsklausel*

Eine *Einheitsklausel* ist eine Klausel der Form $A \Leftarrow$ d.h. eine Programmklausel mit leerem Rumpf.

Definition 4.5. *Logikprogramm*

Ein *Logikprogramm* ist eine endliche Menge von Programmklauseln.

Definition 4.6. *Definition*

In einem Logikprogramm ist die Menge aller Programmklauseln mit dem gleichen Prädikat p im Kopf die *Definition* von p.

Definition 4.7. *Zielklausel*

Eine *Zielklausel* ist eine Klausel der Form $\Leftarrow B_1, \ldots, B_n$ d.h. eine Klausel ohne Kopf. Jedes $B_i (i = 1, \ldots, n)$ ist ein *Teilziel* der Zielklausel.

Seien $y_1, \ldots, y_r$ die Variablen der Zielklausel $\Leftarrow B_1, \ldots, B_n$, dann ist dies eine Abkürzung für $\bigwedge y_1 \ldots \bigwedge y_r(\neg B_1 \vee \ldots \vee \neg B_n)$ oder, äquivalent dazu, $\neg \bigvee y_1 \ldots \bigvee y_r(B_1 \wedge \ldots \wedge B_n)$

Definition 4.8. *Leere Klausel*

Die *leere Klausel* $\square$, ist die Klausel ohne Kopf und Rumpf. Sie ist als Kontradiktion zu verstehen.

Definition 4.9. *Horn Klausel*

Eine *Horn Klausel* ist eine Klausel, die entweder eine Programmklausel oder eine Zielklausel ist.

4.2 Umwandlung von Formeln in Klauselform

Für die Umwandlung von Formeln in Klauselform gelten die folgenden Regeln:

1. Beseitigung des Bikonditionals (Bikond):
   $p \Leftrightarrow q \equiv (p \Rightarrow q) \wedge (q \Rightarrow p)$
2. Beseitigung des Konditionals (Kond):
   $p \Rightarrow q \equiv \neg p \vee q$
3. Skopus der Negation verringern (De Morgan):
   $\neg(p \wedge q) \equiv \neg p \vee \neg q$
   $\neg(p \vee q) \equiv \neg p \wedge \neg q$
   $\neg \bigwedge xP(x) \equiv \bigvee x \neg P(x)$
   $\neg \bigvee xP(x) \equiv \bigwedge x \neg P(x)$
4. Doppelte Negation beseitigen (Neg):
   $\neg\neg p \equiv p$
5. Variablen umbenennen, so daß jeder Quantor eindeutig eine Variable bindet. Da Variablen nur Platzhalter sind, wird dadurch der Wahrheitswert der Formel nicht beeinflußt. Zum Beispiel, die Formel
   $\bigwedge xP(x) \vee \bigwedge xQ(x)$
   würde dadurch zu
   $\bigwedge xP(x) \vee \bigwedge yQ(y)$
   umgewandelt werden. Dies dient der Vorbereitung für den nächsten Schritt:
6. Bringe die Formel in die Pränexe Normalform, indem alle Quantoren an den Anfang der Formel gestellt werden, ohne jedoch ihre relative Reihenfolge zu verändern. Dies ist möglich, weil es durch den vorherigen Schritt keine Namenskonflikte geben kann. Eine Pränexe Normalform besteht aus einem Präfix aus Quantoren gefolgt von einer quantorenfreien Matrix. Beispiel:
   $$\underbrace{\bigwedge x \bigwedge y}_{\text{Präfix}} \underbrace{P(x) \vee Q(y)}_{\text{Matrix}}$$
7. Eliminiere die Existenzquantoren. Dies ist ein etwas schwierig nachzuvollziehender Schritt. Eine Formel, die eine existenzquantifizierte Variable enthält, z.B. $\bigvee x\ Bundeskanzler(x)$ behauptet, daß es irgendein Indivduum gibt, das für x eingesetzt eine wahre Aussage ergibt. Wir wissen nicht, wer dieses Individuum ist, sondern nur, daß es existieren muß. Wir können diesem Individuum einen vorläufigen Namen geben — sagen wir s_1 — und nehmen an, daß es für x eingesetzt die Existenzaussage wahr macht. Die Formel $\bigvee x\ Bundeskanzler(x)$ kann damit in $Bundeskanzler(s_1)$ transformiert werden. Man bezeichnet einen solchen "vorläufigen" Namen als Skolemkonstante.

   Betrachten wir nun die Aussage "Jeder hat einen Vater": $\bigwedge x \bigvee y\ Vater(y, x)$. In diesem Falle steht der Existenzquantor im Skopus eines Allquantors. Es

kann daher sein, daß der Wert von y, der das Prädikat erfüllen würde, in irgendeiner Weise vom Wert von x abhängig ist. In diesem Fall geht man davon aus, daß es eine FUNKTION gibt, die in Abhängigkeit von x den Wert von y bestimmt: $y = s_2(x)$. Man nennt eine solche Funktion eine SKOLEM FUNKTION[4] Unsere Beispielformel kann damit in $\bigwedge x\ Vater(s_2(x), x)$ transformiert werden.

8. Da jetzt alle noch verbleibenden Variablen allquantifiziert sind, kann das Präfix per Konvention weggelassen werden.
9. Konvertiere die verbleibende Matrix in eine Konjunktion von Klauseln
   1. Konjunktion nach außen ziehen (Distributivgesetz der Disjunktion: -Distrib):
      $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$
   2. Gegebenenfalls Anwendung von Kommutativ- und Assoziativgesetzen (Komm, Assoz):
   3. Gegebenenfalls Vereinfachungen (Vereinf):
      a. $p \wedge p \equiv p$
      b. $p \vee p \equiv p$
      c. $p \Rightarrow p \equiv p$
10. Eine Konjunktion von Klauseln wird zu einer Klauselmenge zusammengefaßt. Falls mehrere Formeln zur Beschreibung des gleichen Sachverhalts dienen, werden sie nach der Konversion zu einer Klauselmenge zusammengefaßt.
11. Gegebenenfalls Umbenennung der Variablen in der Klauselmenge nach dem vorigen Schritt, so daß keine zwei Klauseln die gleichen Variablen enthalten.

Beispiel:

Schritt	**Formel**	**Kommentar**
0	$\bigwedge x[\bigwedge yP(x,y) \Rightarrow \neg \bigwedge y(Q(x,y) \Rightarrow R(x,y))]$	Ausgangsformel
1	$\bigwedge x[\neg \bigwedge yP(x,y) \vee \neg \bigwedge y(\neg Q(x,y) \vee R(x,y))]$	Konditional
2	$\bigwedge x[\bigvee y\neg P(x,y) \vee \bigvee y(Q(x,y) \wedge \neg R(x,y))]$	Skopus der Negation
3	$\bigwedge x[\bigvee y\neg P(x,y) \vee \bigvee z(Q(x,z) \wedge \neg R(x,z))]$	Variablen umbenennen
4	$\bigwedge x[\neg P(x, s_1(x)) \vee (Q(x, s_2(x)) \wedge \neg R(x, s_2(x)))]$	Skolemisierung
5	$\neg P(x, s_1(x)) \vee (Q(x, s_2(x)) \wedge \neg R(x, s_2(x)))$	Präfix weglassen
6	$(\neg P(x, s_1(x)) \vee Q(x, s_2(x)) \wedge$ $(\neg P(x, s_1(x)) \vee \neg R(x, s_2(x)))$	Distribution
7	$\{\neg P(x, s_1(x)) \vee Q(x, s_2(x)),$ $\neg P(x, s_1(x)) \vee \neg R(x, s_2(x))\}$	Klauselmenge
8	$\{\neg P(x_1, s_1(x_1)) \vee Q(x_1, s_2(x_1)),$ $\neg P(x_2, s_1(x_2)) \vee \neg R(x_2, s_2(x_2))\}$	Variable umbenennen

[4] Benannt nach einem norwegischen Mathematiker namens Skolem. Skolemkonstante sind Skolemfunktionen mit Null Argumenten.

Als weiterer Schritt wäre noch die Umformung zur Klauselnotation möglich gewesen, z.B. $\{Q(x_1, s_2(x_1)) \Leftarrow P(x_1, s_1(x_1)), \Leftarrow P(x_2, s_1(x_2)), R(x_2, s_2(x_2))\}$.

4.3 Anwendung des Resolutionsprinzips

Die einzelnen Klauseln einer Konjunktion werden zu selbständigen Prämissen. Das Beweisverfahren ist das des indirekten Beweises, d.h. die Negation der zu beweisenden Aussage wird zu den Prämissen hinzugenommen. Es wird dann das Resolutionsschema auf Paare von Aussagen angewandt und die Resolvente zur Aussagenmenge hinzugefügt, solange bis ein Widerspruch entsteht oder die Resolution nicht mehr angewandt werden kann.

Beispiel:

Prämissen	**Übersetzt in Klauselform**		
$p \vee q$	(1)	$p \vee q$	
$q \Rightarrow r$	(2)	$\neg q \vee r$	
$\neg r$	(3)	$\neg r$	
	(4)	$\neg p$	Indirekter Beweis
	(5)	$\neg q$	(2),(3) Resolution
	(6)	p	(1),(5) Resolution
	(7)	$\square$	(4),(6) Resolution

Das Verfahren zum Beweis einer Aussage p nach dem Resolutionsprinzip auf der Grundlage einer Aussagenmenge $\mathcal{P}$ kann wie folgt beschrieben werden:

1. Übersetze alle Aussagen aus $\mathcal{P}$ in Klauselform. Die so erhaltene Klauselmenge sei $\mathcal{K}$.
2. Negiere p, übersetze das Resultat in Klauselform und füge das Ergebnis zur Klauselmenge $\mathcal{K}$ hinzu.
3. Wiederhole die folgenden Schritte solange, bis eine Kontradiktion gefunden ist, oder erkennbar ist, daß kein Ergebnis erzielbar ist.
    1. Wähle zwei Klauseln K_1 und K_2 aus der jeweils gültigen Klauselmenge $\mathcal{K}_n$ derart, daß es ein Literal L gibt, das in einer der beiden Kauseln positiv, in der anderen negativ vorkommt.
    2. Bilde die Resolvente R aus K_1 und K_2.
    3. Wenn $R = \square$, dann ist ein Widerspruch gefunden worden. Andernfalls füge R zur Klauselmenge $\mathcal{K}_n$ hinzu.

4.4 Die PS-Grammatik als Logikprogramm

Die Umformung der PS-Grammatik (3.1) in die Klauselform ist relativ einfach. Nehmen wir als Beispiel (R1):

(4.10.)

Schritt	**Formel**	**Kommentar**
0	$\bigwedge x \bigwedge y(NP(x) \wedge VP(y) \Rightarrow Satz(x \frown y))$	Ausgangsformel
1	$\bigwedge x \bigwedge y(\neg(NP(x) \wedge VP(y)) \vee Satz(x \frown y))$	Konditional
2	$\bigwedge x \bigwedge y(\neg NP(x) \vee \neg VP(y) \vee Satz(x \frown y))$	Skopus der Negation
3	$\neg NP(x) \vee \neg VP(y) \vee Satz(x \frown y)$	Präfix weglassen
4	$Satz(x \frown y) \vee \neg NP(x) \vee \neg VP(y)$	Ordnung der Literale
5	$Satz(x \frown y) \Leftarrow NP(x), VP(y)$	Klauselnotation

Die Gesamtgrammatik in konjunktiver Normalform lautet wie folgt, wobei die Variablen für spätere Referenzzwecke durch Indizes umbenannt werden:

(4.11.) Grammatik in konjunktiver Normalform
R1: $\neg NP(x_1) \vee \neg VP(y_1) \vee Satz(x_1 \frown y_1)$
R2: $\neg Det(x_2) \vee \neg N(y_2) \vee NP(x_2 \frown y_2)$
R3: $\neg Name(x_3) \vee NP(x_3)$
R4: $\neg Vt(x_4) \vee \neg NP(y_4) \vee VP(x_4 \frown y_4)$
R5: $\neg Vi(x_5) \vee VP(x_5)$
Lexikon:
$Det(the)$
$N(boy)$
$N(girl)$
$N(ball)$
$Name(John)$
$Name(Mary)$
$Vt(loves)$
$Vt(kicked)$
$Vi(jumped)$
$Vi(laughed)$

Bei der Umwandlung in Klauselnotation ist nur zu beachten, daß die Lexikoneinträge positive Literale sind und daher zu Einheitsklauseln werden:

(4.12.) PS-Grammatik in Klauselnotation
R1: $Satz(x_1 \frown y_1) \Leftarrow NP(x_1), VP(y_1)$
R2: $NP(x_2 \frown y_2) \Leftarrow Det(x_2), N(y_2)$
R3: $NP(x_3) \Leftarrow Name(x_3)$
R4: $VP(x_4 \frown y_4) \Leftarrow Vt(x_4), NP(y_4)$
R5: $VP(x_5) \Leftarrow Vi(x_5)$

Lexikon:
$Det(the) \Leftarrow$
$N(boy) \Leftarrow$
$N(girl) \Leftarrow$
$N(ball) \Leftarrow$
$Name(John) \Leftarrow$
$Name(Mary) \Leftarrow$
$Vt(loves) \Leftarrow$
$Vt(kicked) \Leftarrow$
$Vi(jumped) \Leftarrow$
$Vi(laughed) \Leftarrow$

An dieser Form der Grammatik ist zweierlei zu erkennen:

1. Alle Klauseln sind Programmklauseln oder Einheitsklauseln, d.h. die Grammatik ist ein Logikprogramm im definierten Sinne.
2. PS-Regeln im üblichen Format haben eigentlich im Kern bereits die Form von Programmklauseln. In einer PS-Regel wie $A \rightarrow B$ entspricht A einem positiven Literal als Kopf einer Programmklausel und B einer Folge von negativen Literalen als Rumpf der Klausel.

4.5 Substitution und Unifikation

Für die Anwendung des Resolutionsschemas auf zwei Klauseln ist Voraussetzung, daß ein Literal in einer Klausel positiv, in der anderen negativ vorkommt. In Rahmen der Prädikatenlogik entsteht ein Problem dadurch, daß Formeln erst durch die Substitution von Variablen verlgeichbar werden. Nehmen wir beispielsweise das folgende Paar

(4.13.) $\neg Vi(x_5) \vee VP(x_5)$
$Vi(laughed)$

Das Resolutionsschema kann hier erst angewandt werden, wenn man die Variable x_5 durch *laughed* substituiert.

(4.14.) $\neg Vi(laughed) \vee VP(laughed)$
$Vi(laughed)$
$VP(laughed)$ Resolvente

Das Verfahren, durch das festgestellt wird, ob zwei Ausdrücke durch geeignete Substitutionen für ihre Variablen gleich gemacht werden können, nennt man Unifikation. Die Möglichkeit der Unifikation ist eine Grundvoraussetzung für die Anwendung des Resolutionsprinzips in der Prädikatenlogik.

Definition 4.10. *Substitution*

Eine Substitution θ ist eine endliche Menge der Form $\{v_1/t_1, \ldots, v_n/t_n\}$, wobei jedes v_i eine Variable und jedes t_i ein von v_i verschiedener Term

ist und die Variablen $v_1, \ldots, v_n$ verschieden sind. Man sagt die v_i werden durch die t_i substituiert bzw. an sie gebunden. Jedes Element v_i/t_i ist eine BINDUNG für v_i.

Definition 4.11. *Ausdruck*

Ein AUSDRUCK ist entweder ein Term oder eine Formel. Ein EINFACHER AUSDRUCK ist entweder ein Term oder eine Primformel.

Definition 4.12. *Substitutionsinstanz*

Sei ϕ ein Ausdruck und $\theta = \{v_1/t_1, \ldots, v_n/t_n\}$ eine Substitution. Die Anwendung von θ auf ϕ, geschrieben $\phi\theta$, ist der Ausdruck, der entsteht, wenn in ϕ die v_i gleichzeitig durch die t_i ersetzt werden (SUBSTITUTIONSINSTANZ). Man sagt auch, die Variablen v_i in ϕ werden zu den Werten t_i *instantiiert.*

Beispiel: $\theta = \{x_5/laughed\}, \phi = \neg Vi(x_5) \vee VP(x_5)$,
und $\phi\theta = \neg Vi(laughed) \vee VP(laughed)$

Definition 4.13. *Komposition*

Seien $\theta = \{u_1/s_1, \ldots, u_m/s_m\}$ und $\sigma = \{v_1/t_1, \ldots, v_n/t_n\}$ Substitutionen. Die KOMPOSITION (Hintereinanderausführung) $\theta\sigma$ von θ und σ ist die Substitution, die man aus der Menge

$$\{u_1/s_1\sigma, \ldots, u_m/s_m\sigma, v_1/t_1, \ldots, v_n/t_n\}$$

erhält, indem man alle Bindungen $u_i/s_i\sigma$ entfernt, für die $u_i = s_i\sigma$ und alle Bindungen v_j/t_j, für die $v_j \in \{u_1, \ldots, u_m\}$.

Beispiel: Sei $\theta = \{x/f(y), y/z\}$ und $\sigma = \{x/a, y/b, z/y\}$.
Dann ist $\theta\sigma = \{x/f(b), z/y\}$:

$$\{\underbrace{x/f(y)\{x/a, y/b, z/y\}}_{x/f(b)}\ \underbrace{y/z\{x/a, y/b, z/y\}}_{\underline{y/y}},\ \underline{x/a}, \underline{y/b}, z/y\}$$

Die unterstrichenen Terme fallen weg: $\{x/f(b), z/y\}$

Definition 4.14. *leere Substitution*

Die LEERE SUBSTITUTION ist durch die leere Menge { } gegeben und wird mit ϵ bezeichnet. Ist ϕ ein Ausdruck, so gilt $\phi\epsilon = \phi$.

Definition 4.15. *Varianten*

Seien ϕ und ψ Ausdrücke. ϕ und ψ sind VARIANTEN, wenn es zwei Substitution θ und σ gibt derart, daß $\phi = \psi\theta$ und $\psi = \phi\sigma$. Man sagt ϕ sei eine Variante von ψ und umgekehrt.

Beispiel: $p(f(x, y), g(z), a)$ ist eine Variante von $p(f(y, x), g(u), a)$. Dabei ist $\theta = \{y/x, x/y, u/z\}$ und $\sigma = \{x/y, y/x, z/u\}$. Varianten lassen sich durch Umbenennung der Variablen ineinander überführen.

Von besonderem Interesse im Diskussionszusammenhang sind Substitutionen, die eine Menge von Ausdrücken gleich machen (UNIFIZIEREN).

Definition 4.16. *unifizierbar*

Eine Menge von Ausdrücken $\{\phi_1, \ldots, \phi_n\}$ heißt UNIFIZIERBAR, wenn es eine Substitution θ gibt, welche alle Ausdrücke gleich macht, d.h. $\phi_1\theta = \ldots = \phi_n\theta$.

Beispiel: die Substitution $\{x/A, y/B, z/C\}$ unifiziert die Ausdrücke $P(A, y, z)$ und $P(x, B, z)$ mit dem Ergebnis $P(A, B, C)$

$$P(A, y, z)\{x/A, y/B, z/C\} = P(A, B, C) = P(x, B, z)\{x/A, y/B, z/C\}$$

Definition 4.17. *Unifikator*

Eine Substitution θ heißt UNIFIKATOR für eine Menge von Ausdrücken $\{\phi_1, \ldots, \phi_n\}$ falls diese dadurch unifiziert werden, d.h. wenn gilt $\{\phi_1, \ldots, \phi_n\}\theta = \{\psi\}$ wobei $\psi = \phi_1\theta = \ldots = \phi_n\theta$.

Obwohl die Substitution $\{x/A, y/B, z/C\}$ die beiden Ausdrücke $P(A, y, z)$ und $P(x, B, z)$ unifiziert, ist sie nicht der einzige Unifikator. Da die Variable z in beiden Ausdrücken an der gleichen Argumentstelle vorkommt, muß sie nicht substituiert werden, um die Gleichheit zu erreichen. Mit der Substitution $\{x/A, y/B\}$ wären die Ausdrücke ebenfalls unifizierbar gewesen.

$$P(A, y, z)\{x/A, y/B\} = P(A, B, z) = P(x, B, z)\{x/A, y/B\}$$

Der Unifikator $\{x/A, y/B\}$ ist *allgemeiner* als der Unifikator $\{x/A, y/B, z/C\}$. Intuitiv ist ein Unifikator umso allgemeiner, je weniger Variable bei der Unifikation substituiert werden. Die Substitution $\{z/F(w)\}$ ist allgemeiner als $\{z/F(C)\}$. Eine Substitution θ ist allgemeiner als eine Substitution σ gdw. es eine andere Substitution δ gibt derart, daß $\theta\delta = \sigma$. Im Beispiel: $\{z/F(w)\}\{w/C\} = \{z/F(C)\}$.

Definition 4.18. *allgemeinster Unifikator*

Sei S eine endliche Menge von einfachen Ausdrücken. Ein Unifikator θ für S heißt ALLGEMEINSTER UNIFIKATOR für S (engl. *most general unifier (mgu)*), wenn es für jeden weiteren Unifikator σ von S eine Substitution γ gibt derart, daß $\sigma = \theta\gamma$.

Beispiel: Die Ausdrucksmenge $\{p(f(x), z), p(y, a)\}$ ist unifizierbar beispielsweise durch den Unifikator $\sigma = \{y/f(a), x/a, z/a\}$. $p(f(x), z)\sigma = p(f(a), a) = p(y, a)\sigma$. Ein allgemeinster Unifikator ist $\theta = \{y/f(x), z/a\}$. Es gilt $\sigma = \theta\{x/a\}$.

4.6 UNIFIKATIONSALGORITHMUS

Ein Unifikationsalgorithmus soll zunächst in informeller Form gegeben werden. Gegeben seien zwei einfache Ausdrücke ϕ und ψ, gesucht wird der allgemeinste Unifikator von ϕ und ψ:

1. Sind ϕ und ψ gleiche Konstanten oder gleiche Variable, so sind sie mit der leeren Substitution { } unifizierbar. Sind sie verschiedene Konstante schlägt die Unifikation fehl.
2. Ist ϕ eine Variable, die nicht in ψ vorkommt, so sind ϕ und ψ mit der Substitution $\{\phi/\psi\}$ unifizierbar. Kommt ϕ in ψ vor, so schlägt die Unifikation fehl.
3. Sind ϕ und ψ Funktionen mit gleichem Funktor und gleicher Stelligkeit (Argumentzahl) ($f(s_1, \ldots, s_n)$ bzw. $f(t_1, \ldots, t_n)$) so sind sie unifizierbar, wenn jedes Argumentpaar $t_i, s_i, i = 1 \ldots n$ unifizierbar ist. Andernfalls schlägt die Unifikation fehl.

Für eine präzisere Formulierung muß zunächst ein weiterer Begriff definiert werden:

Definition 4.19. *Unterschiedsmenge*

Sei A eine endliche Menge von einfachen Ausdrücken. Die UNTERSCHIEDSMENGE (engl. *disagreement set*) von A ist wie folgt definiert. Ermittle die am weitesten links stehende Symbolposition an der nicht alle Ausdrücke in A das gleiche Symbol haben und extrahiere aus jedem Ausdruck in A den Teilausdruck, der an dieser Symbolposition beginnt. Die Menge all dieser Teilausdrücke ist die Unterschiedsmenge.

Beispiel: Sei $A = \{p(f(x), h(y), a), p(f(x), z, a), p(f(x), h(y), b)\}$. Dann ist die Unterschiedsmenge $\{h(y), z\}$.

Ein Unifikationsalgorithmus kann auf dieser Grundlage präziser und einfacher wie folgt angegeben werden (A sei eine endliche Menge von einfachen Ausdrücken):

UNIFIKATIONSALGORITHMUS

1. Setze $k = 0$ und $\theta_0 = \epsilon$
2. Wenn $A\theta_k$ (d.h. die Anwendung von θ_k auf A) eine Einermenge ist (nur aus einem Element besteht), dann halt; θ_k ist der allgemeinste Unifikator von A. Andernfalls ermittle die Unterschiedsmenge D_k von $A\theta_k$.
3. Gibt es ein v und ein t in D_k, so daß v eine Variable ist, die in t nicht vorkommt, dann setze $\theta_{k+1} = \theta_k\{v/t\}$, erhöhe k um 1 und gehe zu 2. Andernfalls halt; A ist nicht unifizierbar.

Beispiel: Sei $A = \{p(a, x, h(g(z))), p(z, h(y), h(y))\}$

(a) $\theta_0 = \epsilon$

(b) $D_0 = \{a, z\}$, $\theta_1 = \{z/a\}$, und $A\theta_1 = \{p(a, x, h(g(a))), p(a, h(y), h(y))\}$.

(c) $D_1 = \{x, h(y)\}$, $\theta_2 = \{z/a, x/h(y)\}$ und $A\theta_2 = \{p(a, h(y), h(g(a))), p(a, h(y), h(y))\}$.

(d) $D_2 = \{g(a), y\}$, $\theta_3 = \{z/a, x/h(g(a)), y/g(a)\}$ und $A\theta_3 = \{p(a, h(g(a)), h(g(a)))\}$.

Also ist A unifizierbar und θ_3 ist ein allgemeinster Unifikator.

4.7 Anwendungsbeispiel

Zur besseren Vergleichbarkeit soll das gleiche Beispiel wie in Abschnitt 3. verwendet werden: *the girl laughed*. Es soll also nach dem Resolutionsverfahren bewiesen werden, daß die Aussage $Satz(the \frown girl \frown laughed)$ wahr ist. Wir legen dafür die gleiche Grammatik zugrunde, allerdings in Klauselnotation. Der Bequemlichkeit halber soll sie hier noch einmal wiederholt werden:

(4.15.) PS-Grammatik in Klauselnotation
R1: $Satz(x_1 \frown y_1) \Leftarrow NP(x_1), VP(y_1)$
R2: $NP(x_2 \frown y_2) \Leftarrow Det(x_2), N(y_2)$
R3: $NP(x_3) \Leftarrow Name(x_3)$
R4: $VP(x_4 \frown y_4) \Leftarrow Vt(x_4), NP(y_4)$
R5: $VP(x_5) \Leftarrow Vi(x_5)$
Lexikon:
$Det(the) \Leftarrow$
$N(boy) \Leftarrow$
$N(girl) \Leftarrow$
$N(ball) \Leftarrow$
$Name(John) \Leftarrow$
$Name(Mary) \Leftarrow$
$Vt(loves) \Leftarrow$
$Vt(kicked) \Leftarrow$
$Vi(jumped) \Leftarrow$
$Vi(laughed) \Leftarrow$

Damit die Resolution durchgeführt werden kann, ist erforderlich, daß in zwei verschiedenen Klauseln ein Literal einmal positiv und einmal negativ vorkommt. Hier zeigt sich der syntaktische Vorteil von Programmklauseln, insofern nur der Kopf ein positives Literal sein kann, während der Rumpf nur aus negativen Literalen besteht. Zur Beseitigung eines Literals aus dem Rumpf einer Klausel müssen wir versuchen, dieses mit dem Kopf einer Programmklausel zu unifizieren.

Die Prämissen sind die Programmklauseln (einschließlich Einheitsklauseln) der Grammatik. Gemäß dem Verfahren des indirekten Beweises nehmen wir zunächst die Negation der zu beweisende Aussage zu den Prämissen hinzu.

(4.16.) $\neg Satz(the \frown girl \frown laughed)$

Es handelt sich um ein negatives Literal, so daß wir die Klauselnotation

(4.17.) $\Leftarrow Satz(the \frown girl \frown laughed)$,

d.h. eine Zielklausel, erhalten. Im folgenden Beweis steht Z für Zielklausel, P für Programmklausel, U für Unifikator und R für Resolvente. Die Resolvente ist jeweils die Zielklausel für den nächsten Schritt:

Z:		$\Leftarrow Satz(the \frown girl \frown laughed)$
P:	$Satz(x_1 \frown y_1)$	$\Leftarrow NP(x_1), VP(y_1)$
U:	$\{x_1/the \frown girl, y_1/laughed\}$	
R:		$\Leftarrow NP(the \frown girl), VP(laughed)$
Z = R:		
P:	$NP(x_4 \frown y_4)$	$\Leftarrow Det(x_4), N(y_4)$
U:	$\{x_4/the, y_4/girl\}$	
R:		$\Leftarrow Det(the), N(girl), VP(laughed)$
Z = R:		
P:	$Det(the)$	$\Leftarrow$
U:	ϵ	
R:		$\Leftarrow N(girl), VP(laughed)$
Z = R:		
P:	$N(girl)$	$\Leftarrow$
U:	ϵ	
R:		$\Leftarrow VP(laughed)$
Z = R:		
P:	$VP(x_5)$	$\Leftarrow Vi(x_5)$
U:	$\{x_5/laughed\}$	
R:		$\Leftarrow Vi(laughed)$
Z = R:		
P:	$Vi(laughed)$	$\Leftarrow$
U:	ϵ	
R:		$\square$

Im folgenden Beispiel geht es um die zunächst merkwürdig erscheinende Frage, ob es überhaupt Ketten gibt, die nach der Grammatik zugelassen sind. Wir gehen aus von der Behauptung $\bigvee x\ Satz(x)$. Die Negation dieser Behauptung, $\neg \bigvee x\ Satz(x)$, wird als zusätzliche Prämisse übernommen. Sie muß zunächst unter Verwendung der Äquivalenz $\neg \bigvee x\ P(x) \equiv \bigwedge x \neg P(x)$ in Klauselform übersetzt werden. Das Ergebnis, $\bigwedge x \neg Satz(x)$, hat die Klauselform $\Leftarrow Satz(x)$. Der Beweis sieht dann wie folgt aus:

Z:	$\Leftarrow Satz(x)$
P: $Satz(x_1 \frown y_1)$	$\Leftarrow NP(x_1), VP(y_1)$
U: $\{x/x_1 \frown y_1\}$	
R:	$\Leftarrow NP(x_1), VP(y_1)$
Z = R:	
P: $NP(x_3)$	$\Leftarrow Name(x_3)$
U: $\{x_1/x_3\}$	
R:	$\Leftarrow Name(x_3), VP(y_1)$
Z = R:	$\Leftarrow Name(x_3), VP(y_1)$
P: $Name(Mary)$	$\Leftarrow$
U: $\{x_3/Mary\}$	
R:	$\Leftarrow VP(y_1)$
Z = R:	
P: $VP(x_5)$	$\Leftarrow Vi(x_5)$
U: $\{y_1/x_5\}$	
R:	$\Leftarrow Vi(x_5)$
Z = R:	
P: $Vi(jumped)$	$\Leftarrow$
U: $\{x_5/jumped\}$	
R:	$\square$

Von Interesse ist hier primär nicht die Tatsache, daß die Behauptung $\bigvee x\ Satz(x)$ beweisbar ist, sondern daß auch eine Instanz für x geliefert wird, die man erhält, wenn man die Komposition der Unifikatoren bildet:

(4.18.) Komposition der Unifikatoren

$$\underbrace{\{x/x_1 \frown y_1\}\{x_1/x_3\}}\{x_3/Mary\}\{y_1/x_5\}\{x_5/jumped\}$$
$$\underbrace{\{x/x_3 \frown y_1\}\{x_3/Mary\}}\{y_1/x_5\}\{x_5/jumped\}$$
$$\underbrace{\{x/Mary \frown y_1\}\{y_1/x_5\}}\{x_5/jumped\}$$
$$\underbrace{\{x/Mary \frown x_5\}, x_5/jumped\}}$$
$$\{x/Mary \frown jumped\}$$

Mit anderen Worten, im Zuge des Beweises werden durch Unifikation Variablen gebunden und damit Sätze GENERIERT. Jeder alternative Beweisweg generiert einen anderen Satz.

5. Definite Clause Grammar (DCG)

5.1 Verkettung

Wir waren bisher davon ausgegangen, daß in einer Regel wie

(5.1.) $Satz(x \frown y) \Leftarrow NP(x), VP(y)$

eine Verkettungsoperation definiert ist, die Paare von Ketten auf Ketten abbildet ($\mathcal{K} \times \mathcal{K} \mapsto \mathcal{K}$, wenn $\mathcal{K}$ die Menge aller möglichen Ketten ist) und ein funktionales Argument wie $x \frown y$ entsprechend ausgewertet wird. Dies ist jedoch insbesondere in Prolog nicht gegeben.[5] Es gibt aber eine Möglichkeit, die Verkettung ausschließlich über die Unifikation von Termen zu beschreiben. Dazu benötigen wir zunächst eine präzise Definition des Begriffs Kette:

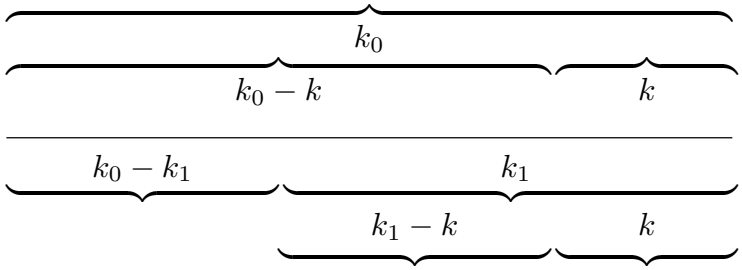
Definition 5.1. *Kette*

Eine Kette kann induktiv wie folgt definiert werden

1. Der Term *nil* ist eine Kette, die leere Kette.
2. Ist R eine Kette und K ein Term, dann ist $K.R$ eine Kette. K ist der Kopf und R der Rumpf der Kette.

Ein Ausdruck wie *the girl laughed* wird damit als *the.girl.laughed.nil* notiert.

Die Behandlung der Verkettung (bzw. umgekehrt der Zerlegung von Ketten in Teilketten) beruht auf folgender Überlegung: Ein Kette k_0 enthält am Anfang einen Satz mit einem Rest k (d.h. $Satz(k_0 - k)$), wenn es eine Anfangskette $k_0 - k_1$ gibt, für die gilt $NP(k_0 - k_1)$ und die Restkette k_1 eine Kette $k_1 - k$ mit $VP(k_1 - k)$ enthält. Die gesamte Kette k_0 ist ein Satz, wenn die Restkette k leer ist. Die leere Kette soll mit der Konstante *nil* bezeichnet werden.



Der entscheidende Punkt ist dabei, daß im Falle eines terminalen Symbols t gilt: $k_0 = t.k$. Die Zerlegung einer Kette in Teilketten wird damit zurückgeführt auf die Zerlegung der Kette in das erste teminale Element und die Restkette. Die Differenzketten[6] wie $k_0 - k$ werden im folgenden durch zwei Argumente dargestellt, so daß die erste Syntaxregel wie folgt formuliert werden kann:

[5] In Prolog sind Funktionen im prädikatenlogischen Sinne bloße syntaktische Strukturen, die nicht ausgewertet werden.

[6] In Prolog werden Ketten als Listen dargestellt. Man spricht dort entsprechend von Differenzlisten.

(5.2.) $Satz(k_0, k) \Leftarrow NP(k_0, k_1), VP(k_1, k)$

Lexikonregeln hätten die Form $LK(k_0, k) \Leftarrow$, wobei jedoch gilt $k_0 = Wort.k$, z.B.:

(5.3.) $Det(the.k, k) \Leftarrow$

(5.4.) PS-Grammatik in Klauselnotation mit Differenzketten
R1: $Satz(x_1, z_1) \Leftarrow NP(x_1, y_1), VP(y_1, z_1)$
R2: $NP(x_2, z_2) \Leftarrow Det(x_2, y_2), N(y_2, z_2)$
R3: $NP(x_3, z_3) \Leftarrow Name(x_3, z_3)$
R4: $VP(x_4, z_4) \Leftarrow Vt(x_4, y_4), NP(y_4, z_4)$
R5: $VP(x_5, z_5) \Leftarrow Vi(x_5, z_5)$
Lexikon:
$Det(the.z_6, z_6) \Leftarrow$
$N(boy.z_7, z_7) \Leftarrow$
$N(girl.z_8, z_8) \Leftarrow$
$N(ball.z_9, z_9) \Leftarrow$
$Name(John.z_{10}, z_{10}) \Leftarrow$
$Name(Mary.z_{11}, z_{11}) \Leftarrow$
$Vt(loves.z_{12}, z_{12}) \Leftarrow$
$Vt(kicked.z_{13}, z_{13}) \Leftarrow$
$Vi(jumped.z_{14}, z_{14}) \Leftarrow$
$Vi(laughed.z_{15}, z_{15}) \Leftarrow$

Die der Behauptung "*John kicked the ball* ist ein Satz" entsprechende Zielklausel hat jetzt die Form

(5.5.) $\Leftarrow Satz(John.kicked.the.ball.nil, nil)$

Der Beweis erhält nunmehr die in (5.6.) gezeigte Form.

5.2 Der DCG-Formalismus

Nachdem nun der Beweismechanismus der hier diskutierten Form der Logikgrammatik (hoffentlich) deutlich geworden ist, können wir dazu übergehen, diesen Mechanismus vor dem Grammatiker zu "verstecken". Wir haben gesehen, daß eine PS-Regel wie $Satz \rightarrow NP\ VP$ in der Klauselnotation der Logikgrammatik als $Satz(k_0, k) \Leftarrow NP(k_0, k_1), VP(k_1, k)$ wiedergegeben wird. Lexikonregeln, die nichtterminale lexikalische Kategorien zu terminalen Symbolen expandieren, werden zu Einheitsklauseln, d.h. eine Regel wie $Vt \rightarrow kicked$ wird zu $Vt(kicked.k, k) \Leftarrow$. Es ist nunmehr möglich, Übersetzungsregeln zu formulieren, die PS-Regeln in Regeln der Logikgrammatik überführen: Ein Ausdruck wie $A \rightarrow B \quad C$ wird übersetzt in $A(k_0, k) \Leftarrow B(k_0, k_1), C(k_1, k)$

(5.6.)

Z:		$\Leftarrow Satz(John.kicked.the.ball.nil, nil)$
P:	$Satz(x_1, z_1)$	$\Leftarrow NP(x_1, y_1), VP(y_1, z_1)$
U:	$\{x_1/John.kicked.the.ball.nil, z_1/nil\}$	
R:		$\Leftarrow NP(John.kicked.the.ball.nil, y_1), VP(y_1, nil)$

Z = R:

P:	$NP(x_3, z_3)$	$\Leftarrow Name(x_3, z_3)$
U:	$\{x_3/John.kicked.the.ball.nil, z_3/y_1\}$	
R:		$\Leftarrow Name(John.kicked.the.ball.nil, y_1), VP(y_1, nil)$

Z = R:

P:	$Name(John.z_{10}, z_{10})$	$\Leftarrow$
U:	$\{z_{10}/kicked.the.ball.nil, y_1/kicked.the.ball.nil\}$	
R:		$\Leftarrow VP(kicked.the.ball.nil, nil)$

Z = R:

P:	$VP(x_4, z_4)$	$\Leftarrow Vt(x_4, y_4), NP(y_4, z_4)$
U:	$\{x_4/kicked.the.ball.nil, z_4/nil\}$	
R:		$\Leftarrow Vt(kicked.the.ball.nil, y_4), NP(y_4, nil)$

Z = R:

P:	$Vt(kicked.z_{13}, z_{13})$	$\Leftarrow$
U:	$\{z_{13}/the.ball.nil, y_4/the.ball.nil\}$	
R:		$\Leftarrow NP(the.ball.nil, nil)$

Z = R:

P:	$NP(x_2, z_2)$	$\Leftarrow Det(x_2, y_2), N(y_2, z2)$
U:	$\{x_2/the.ball.nil, z_2/nil\}$	
R:		$\Leftarrow Det(the.ball.nil, y_2), N(y_2, nil)$

Z = R:

P:	$Det(the.z_6, z_6)$	$\Leftarrow$
U:	$\{z_6/ball.nil, y_2/ball.nil\}$	
R:		$\Leftarrow N(ball.nil, nil)$

Z = R:

P:	$N(ball.z_9, z_9)$	$\Leftarrow$
U:	$\{z_9/nil\}$	
R:		$\Box$

Allgemein soll gelten:

1. Seien $A, B_1, \ldots, B_n$ nichtterminale Symbole und k_i Variable über Wortketten, so gilt
   $A \to B_1 \ldots B_n$ wird übersetzt in
   $A(k_0, k_n) \Leftarrow B_1(k_0, k_1), \ldots, B_i(k_{i-1}, k_i), \ldots, B_n(k_{n-1}, k_n)$,
   wobei für k_n einfach k geschrieben werden kann.
2. Sind A und B_i nichterminale Symbole und ist a ein Terminalsymbol so gilt:
   $A \to a$ wird übersetzt in $A(a.k, k) \Leftarrow$;
   $\ldots B_i a \ldots$ wird übersetzt in $\ldots, B_i(k_{i-1}, a.k_i), \ldots$

Unter Berücksichtigung dieser Übersetzungsregeln kann eine DCG im wesentlichen in der bekannten Form der PSG formuliert werden.[7]

Nun ist mehrfach gezeigt worden, daß kontextfreie Phrasenstrukturgrammatiken in dieser einfachen Form nicht deskriptiv adäquat sind. Der DCG-Formalismus enthält daher auch zwei wesentliche Erweiterungen, die seine Expressivität und Mächtigkeit erheblich vergrößern:

1. Nichtterminale Symbole sind nicht atomar, sondern können durch eine beliebige Zahl von Argumenten aus Termen beliebiger Komplexität erweitert werden, die der Unifikation unterliegen, z.B.
   $Satz \to NP(num, pers), VP(num, pers)$.
2. Es können beliebige Bedingungen (*constraints*) in Form von zusätzlichen Literalen (bzw. Klauseln) hinzugefügt werden. Diese bleiben bei der Übersetzung erhalten, und werden durch geschweifte Klammern gekennzeichnet, z.B.
   $Satz \to NP(typ_1), VP(typ_2), \{subtyp(typ_1, typ_2)\}$

Wir benötigen entsprechend zusätzliche Übersetzungsregeln:

1. $A(x_\iota, \ldots, x_\nu) \to \ldots$ wird übersetzt in $A(x_\iota, \ldots, x_\nu, k_0, k) \Leftarrow \ldots$;
   $B_i(x_\iota, \ldots, x_\nu)$ wird übersetzt in $B_i(x_\iota, \ldots, x_\nu, k_{i-1}, k_i)$
2. $\{B_1, \ldots, B_n\}$ wird übersetzt in $B_1, \ldots, B_n$

Zusätzliche Argumente können beispielsweise dazu verwendet werden, Abhängigkeiten der verschiedensten Art zu formulieren. Eine Regel wie

(5.7.) $Satz \to NP(num, pers), VP(num, pers)$
(in der Übersetzung: $Satz(k_0, k) \Leftarrow NP(num, pers, k_0, k_1), VP(num, pers, k_1, k)$)

drückt z.B. den Sachverhalt aus, daß Subjekt und Prädikat hinsichtlich der grammatischen Kategorien NUMERUS und PERSON übereinstimmen müssen.

Da die Argumente beliebig komplexe Terme sein können, ist es möglich, gleichzeitig mit der Konstruktion eines Beweises Strukturbeschreibungen durch Unifikation aufzubauen. Unsere Beispielgrammatik erhält dadurch folgende Form:

[7] In der üblichen Form der DCG werden terminale Symbole in eckige Klammern [ ] gesetzt, z.B. $N \to [boy]$. Diese Klammern werden in Prolog zur Kennzeichnung von Listen verwendet und terminale Symbole sind einelementige Listen.

(5.8.) DC-Grammatik mit zusätzlichem Argument zur Strukturbeschreibung
R1: $Satz(S(np_0, vp)) \rightarrow NP(np_0), VP(vp)$
R2: $NP(NP(d, n)) \rightarrow Det(d), N(n)$
R3: $NP(NP(name)) \rightarrow Name(name)$
R4: $VP(VP(vt, np_1) \rightarrow Vt(vt), NP(np_1)$
R5: $VP(VP(vi)) \rightarrow Vi(vi)$
Lexikon:
$Det(Definite) \rightarrow the$
$N(N(boy)) \rightarrow boy$
$N(N(girl)) \rightarrow girl$
$N(N(ball)) \rightarrow ball$
$Name(N(John)) \rightarrow John$
$Name(N(Mary)) \rightarrow Mary$
$Vt(Vt(love)) \rightarrow loves$
$Vt(Vt(kick)) \rightarrow kicked$
$Vi(Vi(jump)) \rightarrow jumped$
$Vi(Vi(laugh)) \rightarrow laughed$

Nach unseren Übersetzungsregeln erhalten wir daraus das folgende Logikprogramm:

(5.9.) DCG in Klauselnotation mit zusätzlichem Argument zur Strukturbeschreibung
R1: $Satz(S(np_1, vp), x_1, z_1) \Leftarrow NP(np_1, x_1, y_1), VP(vp, y_1, z_1)$
R2: $NP(NP(d, n), x_2, z_2) \Leftarrow Det(d, x_2, y_2), N(n, y_2, z_2)$
R3: $NP(NP(name), x_3, z_3) \Leftarrow Name(name, x_3, z_3)$
R4: $VP(VP(vt, np_2), x_4, z_4) \Leftarrow Vt(vt, x_4, y_4), NP(np_2, y_4, z_4)$
R5: $VP(VP(vi), x_5, z_5) \Leftarrow Vi(vi, x_5, z_5)$
Lexikon:
$Det(Det(Definite), the.z_6, z_6) \Leftarrow$
$N(N(boy), boy.z_7, z_7) \Leftarrow$
$N(N(girl), girl.z_8, z_8) \Leftarrow$
$N(N(ball), ball.z_9, z.9) \Leftarrow$
$Name(N(John), John.z_{10}, z_{10}) \Leftarrow$
$Name(N(Mary), Mary.z_{11}, z_{11}) \Leftarrow$
$Vt(Vt(love), loves.z_{12}, z_{12}) \Leftarrow$
$Vt(Vt(kick), kicked.z_{13}, z_{13}) \Leftarrow$
$Vi(Vi(jump), jumped.z_{14}, z_{14}) \Leftarrow$
$Vi(Vi(laugh), laughed.z_{15}, z_{15}) \Leftarrow$

Die folgende Seite zeigt die Ableitung des Satzes *John kicked the ball* mit gleichzeitiger Konstruktion einer Strukturbeschreibung. Die mit S: bezeichnete Zeile zeigt nach jedem Resolutionsschritt die Strukturbeschreibung nach Anwendung des Unifikators auf den vorherigen Output. Zu beweisen ist

die Aussage: $\bigvee sb\ Satz(sb, John.kicked.the.ball.nil, nil)$, d.h. die Zielklausel $\Leftarrow Satz(sb, John.kicked.the.ball.nil, nil)$.

Durch Regeln wie

(5.10.) $S \rightarrow NP(typ_0), VP(typ_1), \{Subtyp(typ_0, typ_1)\}$
$\vdots$
$VP(typ_1) \rightarrow Vt(typ_1, typ_2), NP(typ_3), \{Subtyp(typ_3, typ_2)\}$
$Vt(Person, Universal) \rightarrow admired$
$\vdots$
$N(Person) \rightarrow John$
$N(Idea) \rightarrow sincerity$
$\vdots$

werden Kontextabhängigkeiten zwischen dem Verb und seinen Aktanten ausgedrückt, die z.B. *John admired sincerity* zulassen, **Sincerity admired John* aber ausschließen. Das Prädikat *Subtyp* kann dabei z.B. auf eine Typhierarchie Bezug nehmen.
In ähnlicher Weise ließe sich eine Semantik im Stile von Montague in die DCG integrieren.

5.3 Zum theoretischen Status der DCG

Die DCG hat keine spezifischen sprachtheoretischen Grundlagen. Die Expressivität ist dazu viel zu groß. Es handelt sich vielmehr um einen Formalismus, der es erlaubt, verschiedene Grammatiken mit je unterschiedlichen sprachtheoretischen Grundlagen auszudrücken. Es macht keinen Sinn, beispielsweise die DCG direkt mit der lexikalisch-funktionalen Grammatik (LFG) zu vergleichen. Es ist aber möglich, eine LFG als DCG auszudrücken.

Z: $\Leftarrow Satz(sb, John.kicked.the.ball.nil, nil)$
P: $Satz(S(np_1, vp), x_1, z_1) \Leftarrow NP(np_1, x_1, y_1), VP(vp, y_1, z_1)$
U: $\{sb/S(np_1, vp), x_1/John.kicked.the.ball.nil, z_1/nil\}$
S: $S(np_1, vp)$

R: $\Leftarrow NP(np_1, John.kicked.the.ball.nil, y_1), VP(vp, y_1, nil)$

Z = R:
P: $NP(NP(name), x_3, z_3) \Leftarrow Name(name, x_3, z_3)$
U: $\{np_1/NP(name), x_3/John.kicked.the.ball.nil, z_3/y_1\}$
S: $S(NP(name), vp)$

R: $\Leftarrow Name(name, John.kicked.the.ball.nil, y_1), VP(vp, y_1, nil)$

Z = R:
P: $Name(N(John), John.z_{10}, z_{10}) \Leftarrow$
U: $\{name/N(John), z_{10}/kicked.the.ball.nil, y_1/kicked.the.ball.nil\}$
S: $S(NP(N(John)), vp)$

R: $\Leftarrow VP(vp, kicked.the.ball, nil)$

Z = R:
P: $VP(VP(vt, np_2), x_4, z_4) \Leftarrow Vt(vt, x_4, y_4), NP(np_2, y_4, z_4)$
U: $\{vp/VP(vt, np_2), x_4/kicked.the.ball.nil, z_4/nil\}$
S: $S(NP(N(John)), VP(vt, np_2))$

R: $\Leftarrow Vt(vt, kicked.the.ball.nil, y_4), NP(np_2, y_4, nil)$

Z = R:
P: $Vt(Vt(kick), kicked.z_{13}, z_{13}) \Leftarrow$
U: $\{vt/Vt(kick), z_13/the.ball.nil, y_4/the.ball.nil\}$
S: $S(NP(N(John)), VP(Vt(kick), np_2))$

R: $\Leftarrow NP(np_2, the.ball.nil, nil)$

Z = R:
P: $NP(NP(d, n), x_2, z_2) \Leftarrow Det(d, x_2, y_2), N(n, y_2, z_2)$
U: $\{np_2/NP(d, n), x_2/the.ball.nil, z_2/nil\}$
S: $S(NP(John), VP(Vt(kick), NP(d, n)))$

R: $\Leftarrow Det(d, the.ball.nil, y_2), N(n, y_2, nil)$

Z = R:
P: $Det(Det(Definite), the.z_6, z_6) \Leftarrow$
U: $\{d/Det(Definite), z_6/ball, nil, y_2/ball.nil\}$
S: $S(NP(John), VP(Vt(kick), NP(Det(Definite), n)))$

R: $\Leftarrow N(n, ball.nil, nil)$

Z = R:
P: $N(N(ball), ball.z_9, z_9) \Leftarrow$
U: $\{n/N(ball), z_9/nil\}$
S: $S(NP(John), VP(Vt(kick), NP(Det(Definite), N(ball))))$

R: $\square$

Bibliographie

Abramson, Harvey & Dahl, Vernonica

1989 *Logic Grammars.* Springer-Verlag: New York etc.

Bates. M.

1978 The Theory and Practice of Augmented Transition Network Grammars. In: Bolc (1978), 191–259.

Bolc, Leonard (ed.)

1978 *Natural Language Communication with Computers.* Springer Verlag: Berlin.

Colmerauer, Alain

1978 Metamorphosis Grammars. In Bolc(1978), 1933–89.

Grosz, Barbara J., Karen Sparck Johne, & Bonnie Lynn Webber, eds.

1986 *Readings in Natural Language Processing.* Morgan Kaufmann: Los Altos, California.

Lloyd, J.W.

1984 *Foundations of Logic Programming.* Springer-Verlag: Berlin etc.

Pereira, Fernando C.N. & Stuart M. Shieber

1987 *Prolog and Natural-Language Analysis.* Center for the Study of Language and Information: Stanford.

Pereira, Fernando C. N. & David H. D. Warren

1980 Definite clause grammars for language analysis — a survey of the formalism and a comparison with augmented transition networks. In: *Artificial Intelligence* **13**:231–278. (Reprint in Grosz et. al. 1986).

Robinson, J.A.

1965 A machine-oriented logic based on the resolution principle. In: *Journal of the ACM* **12**:23–44.

Woods, W. A.

1970 Transition Network Grammars for Natural Language Analysis. In: *Communications of the ACM* **3**: 591–606. (Reprint in Grosz et al. 1986: 71–87.)