Die Implementation kontextfreier Grammatiken in PROLOG

Hans-Ulrich Zimmermann*
Dorneystr. 50A
58454 Witten

15. Juni 2001

Inhaltsverzeichnis

1.1 Einleitung

Neben den endlichen Automaten stellen Grammatiken einen wichtigen Zugang zu den formalen Sprachen dar. Die folgenden Anregungen wenden sich in erster Linie an den Praktiker, der das Thema zum Beispiel innerhalb eines “Wissensbasierten Ansatzes” [MSW99] in seinem Unterricht behandeln will und der Grundkenntnisse in PROLOG besitzt. Der Schwerpunkt liegt auf der ausführlichen Darstellung der Implementationsdetails. Methodisch-didaktische Fragestellungen zur Unterrichtsdurchführung werden nur am Rande gestreift. Leser, die (noch) keine Kenntnisse in PROLOG besitzen, können immerhin die Kürze und die Lesbarkeit der PROLOG-“Programme” erkennen und mit Problemlösungen in den ihnen vertrauten Sprachmitteln vergleichen.
An Hand dreier Beispiele werden zunächst die Grundideen der Implementation kontextfreier Grammatiken schrittweise entwickelt. Dabei wird die wenig bekannte DCG-Notation (definit clause grammar) benutzt. Die Beispiele lassen sich mühelos auf alle kontextfreien Grammatiken übertragen. Zum Schluss wird gezeigt, wie man in Kombination mit der “üblichen” PROLOG-Prädikatschreibweise den Zahlenwert syntaktisch korrekter Zeichenketten berechnet.

*☎02302-48695 zimmermann@h-u-zimmermann.de

Zum Einsatz kam das SWI-PROLOG
http://www.swi.psy.uva.nl/projects/SWI-Prolog/, welches kostenlos (GNU General Public License (GPL)) in WINDOWS und LINUX-Umgebungen eingesetzt werden kann. Es bietet eine Obermenge zum ISO - Standard und deckt alle Belange des Unterrichts in der SEK II ab.
Auch bei diesem Thema stellt ein PROLOG-System eine ideale Lernumgebung dar, um mit einem schülerzentrierten, experimentell-forschenden Unterrichtsstil die grundlegenden Fragestellungen zu erarbeiten.

1.2 Bezeichnungsweisen

In Anlehnung an [Sch95] und [Weg99] werden im Weiteren die folgenden Bezeichnungen verwendet:

Definition: Eine *Grammatik G* ist ein 4-Tupel $G = (V,\ \Sigma,\ P,\ S)$, das folgende Bedingungen erfüllt:

- V ist eine endliche Menge, die Menge der *Variablen* oder *nicht terminalen Symbole.*
- Σ ist eine endliche Menge, das *Terminalalphabet.* Die Elemente von Σ heißen auch *Terminalsymbole.* a^n bezeichnet eine Folge von n Terminalsymbolen $a \in \Sigma$. Die Mengen V und Σ dürfen keine gemeinsamen Elemente besitzen: $V \cap \Sigma = \emptyset$.
- P ist die endliche Menge der *Grammatikregeln* oder *Produktionen.*
- $S \in V$ heißt *Startvariable.*
- Wir schreiben $w \rightarrow z$, wenn sich z in einem Schritt durch Anwendung einer Ableitungsregel aus w ableiten lässt.
  Wir schreiben $w \rightarrow y \mid z$, wenn sich y oder z in einem Schritt durch Anwendung einer Ableitungsregel aus w ableiten lassen.
- Die von G definierte (erzeugte) *Sprache* (Language) bezeichnen wir mit $L(G)$.

1.3 Beispiel 1

Gemeinhin steht bei der Einführung des Themas eine Grammatik, die sich an eine natürliche Sprache anlehnt und an der die grundlegenden Begriffe und Notationen erarbeitet werden (vergl. [Bau93] [FB96] [Sch95]). Als Beispiel dient das folgende Beschreibungsmuster (nach [Sch95]):

<Satz> → <Subjekt> <Prädikat> <Objekt>
<Subjekt> → <Artikel> <Attribut> <Substantiv>
<Artikel> → ε | der | die | das
<Attribut> → ε | <Adjektiv> | <Adjektiv> <Attribut>
<Adjektiv> → kleine | bissige | grosse
<Substantiv> → Hund | Katze
<Prädikat> → jagt

<Objekt> → <Artikel> <Attribut> <Substantiv>

Die Implementation erfolgt nahezu eins zu eins. Dazu werden mit Hilfe eines Editors die folgendenden Regeln in eine ASCII-Datei (Wissensbasis) eingegeben.

```
/***  Implementation Beispiel 1   *******
 ***  Datei:   beisp1.pl          *******/

'Satz'       --> 'Subjekt', 'Praedikat', 'Objekt'.
'Subjekt'    --> 'Artikel', 'Attribut', 'Substantiv'.
'Artikel'    --> [] | [der] | [die] | [das].
'Attribut'   --> [] | 'Adjektiv' | 'Adjektiv', 'Attribut'.
'Adjektiv'   --> [kleine] | [bissige] | [grosse].
'Substantiv' --> [hund] | [katze].
'Praedikat'  --> [jagt].
'Objekt'     --> 'Artikel', 'Attribut', 'Substantiv'.
```

Man erkennt folgende Grundsätze für die Implementation:

- Terminalsymbole werden als einelementige Prolog-Listen in eckigen Klammern dargestellt. Das leere Wort ε entspricht der leeren Liste `[]`.
- Die nicht terminalen Symbole werden in Hochkommata eingekleidet.
  Anmerkung: Will man auf die Hochkommata verzichten, müssen die nicht terminalen Symbole unbedingt klein geschrieben werden. Das führt dann aber zu einem Bruch mit der in der Literatur üblichen Darstellung für Grammatikvariablen.
- Treten auf der rechten Seite einer Regel mehrere Variable auf, werden sie durch Komma abgetrennt.
- Der senkrechte Strich hat die gleiche Bedeutung wie in der gewohnten Grammatikschreibweise.
- Der Ableitungspfeil `-->` steht als vordefinierter Operator standardmäßig zur Verfügung.

Nach dem Laden der Wissensbasis in das Prologsystem benutzt man das zweistellige Standardprädikat `phrase/2` zum Test auf gültige Ableitungen. Die Zahl 2 hinter dem Schrägstrich steht für die Anzahl der Argumente. Das Prädikat `phrase(S, T)` ist erfüllbar, wenn sich mit Hilfe der implementierten Grammatik der Ausdruck `T` aus dem Symbol `S` ableiten lässt. In den folgenden Zeilen, die mit den Zeichen `?-` anfangen, werden die Anfragen an das Prologsystem gestellt. Anfragen werden jeweils durch einen Punkt und durch Betätigung der Eingabetaste abgeschlossen. Danach antwortet das System mit `Yes` oder `No` im Sinne einer Beweismaschine:

```
?- phrase('Satz',[hund, jagt, katze]).

Yes
?- phrase('Satz',[die, hund, jagt, das, bissige, katze]).
```

```
Yes
?- phrase('Satz',[der, hund, jagt]).

No
```

Ohne Ablenkung durch programmiertechnische Feinheiten können die Schüler bereits an dieser Stelle experimentierend-forschend die Grammatik "verfeinern" und die Auswirkungen ihrer Überlegungen systematisch überprüfen. Eine selbständig zu bearbeitende Aufgabenstellung besteht zum Beispiel darin, die Grammatik dahingehend abzuändern, dass auch Sätze wie `der hund jagt` ableitbar sind oder fragwürdige Sätze wie `das katze jagt hund` eben nicht mehr. Zur Ausgabe der Ableitungsbäume kann der Fachlehrer ein fertiges Modul liefern oder die notwendigen Prädikate erarbeiten lassen. Weisweber [Wei97] liefert hierzu wertvolle Hinweise und adaptierbare Beispiele.

An dieser Stelle darf noch ein Beispiel für die Mächtigkeit von PROLOG geliefert werden. Sucht man etwa alle Sätze, die aus genau vier Worten bestehen, so ist der zu untersuchenden Ausdruck `T` einfach durch die vierelementige Liste `[A, B, C, D]` zu ersetzen:

```
?- phrase('Satz',[A,B,C,D]).

A = hund
B = jagt
C = kleine
D = hund ;

A = hund
B = jagt
C = kleine
D = katze ;

A = hund
B = jagt
C = bissige
D = hund            usw. usw.
```

Bei fortgeschrittenen PROLOG-Kenntnissen bietet es sich an, mit nur einer Variablen X zu arbeiten, und diese zunächst mit einer vierelementigen Liste zu "matchen". Im Audruck `X = [_, _, _, _]` steht der Unterstrich jeweils für die "anonyme Variable":

```
?- X=[_, _, _, _], phrase('Satz', X).

X = [hund, jagt, kleine, hund] ;

X = [hund, jagt, kleine, katze] ;

X = [hund, jagt, bissige, hund] ;

X = [hund, jagt, bissige, katze] ;
```

```
X = [hund, jagt, grosse, hund] ;

X = [hund, jagt, grosse, katze]       usw. usw.
```

1.4 Beispiel 2

Die vertiefenden und ergänzenden Beispiele führen auch in den Schulbüchern (vergl. [Bau93]) schnell auf formale Sprachen, bei denen die Terminalsymbole in der Regel aus einzelnen Zeichen bestehen.

Gegeben ist $G_2 = (V,\ \Sigma,\ P,\ S)$ mit $V = \{S,\ A\}$, $\Sigma = \{0, 1\}$ und den Produktionen $P = \{S \to 0S \mid 1A, \quad A \to \varepsilon \mid 1S \mid 0A\}$ (Quelle: [Bau93]).
Diese Grammatik erzeugt alle Worte, in denen die Anzahl der Einsen ungerade ist, beispielsweise $1010111 \in L(G_2)$. Eine Implementation analog zu den Grundsätzen aus dem Beispiel 1 führt sofort zu :

```
/***  Beispiel 2_1           ***
 ***  Datei: beispiel2_1.pl  ***/

'S' --> [0],'S' | [1],'A'.
'A' --> [] | [1],'S' | [0],'A'.
```

Zur Beantwortung der Frage "Gilt $1^5\ 0^5\ 1^6 \in L(G)$?", ist nun allerdings die Anfrage

```
?- phrase('S',[1,1,1,1,1,0,0,0,0,0,1,1,1,1,1,1]).
```

zu stellen. Der Tippaufwand wird ziemlich quälend. Damit einhergehend wächst die Fehlerhäufigkeit. Hilfe bietet hier die Verwendung von sogenannten ASCII-Listen. Eine ASCII-Liste in PROLOG ist eine in "Gänsefüßchen" eingekleidete Zeichenkette, die intern als Liste äquivalenter ASCII-Dezimalwerte verwaltet wird. Als Beispiel wird die Variable X mit den sieben (!) Zeichen "abc ABC" instanziert:

```
?- X = "abc ABC".

X = [97, 98, 99, 32, 65, 66, 67] ;

No
```

Nach der Einführung der ASCII-Listen bleibt die Lesbarkeit der Wissensbasis zwar nicht ganz so gut erhalten:

```
/***  Beispiel 2_2            ***
 ***  Datei: beispiel2_2.pl   ***/

'S' --> "0", 'S' | "1", 'A'.
'A' --> [] | "1", 'S' | "0", 'A'.
```

aber die Anfragen sind erheblich einfacher einzutippen:

```
?- phrase('S',"1111100000111111").

Yes
```

Will man sich, wie im Beispiel 1, nun alle Worte der Länge vier ausgeben lassen, so erhält man natürlich zunächst die interne Darstellung der Ergebnislisten:

```
?- X=[_,_,_,_],phrase('S', X).

X = [48, 48, 48, 49] ;

X = [48, 48, 49, 48] ;

X = [48, 49, 49, 49] ;

X = [48, 49, 48, 48] usw. usw...
```

Dabei sind 48 und 49 die dezimalen ASCII-Werte für die Null und die Eins.

1.5 Beispiel 3

Gegeben ist $G_3 = (V,\ \Sigma,\ P,\ S)$ mit $V = \{S\},\ \Sigma = \{\ a\ \}$ und den Produktionen $P = \{S \rightarrow\ a \mid SS\}$.
Diese Grammatik erzeugt beliebig lange, nicht leere Worte aus dem Zeichen a
Diesmal funktioniert eine allzu sorglose Übertragung der bisherigen Verfahren nur auf den ersten Blick:

```
/* fehlerhafte Implementation des Beispiels 3  */

'S' --> "a" |'S','S'.
```

Die fehlerhafte Implementation liefert das gewünschte Ergebnis nur dann, falls das untersuchte Wort w ein Element von $L(G_3)$ ist:

```
?- phrase('S',"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa").

Yes
```

Im anderen Fall erhält man einen Speicherüberlauf:

```
?- phrase('S',"aaaabaaaaaaaaaaaaaaaaaaaaaaaaa").
ERROR: Out of local stack
```

Die Ursache liegt in der Produktion `'S' --> 'S','S'`. Sie liefert bei der Untersuchung des Zeichens b keinen Rekursionsabbruch und verursacht damit den Speicherüberlauf. Wir verbessern dieses grundlegende Beispiel wie folgt:

```
/* fehlerlose Implementation des Beispiels 3  */

'S' --> "a" | "a",'S'.
```

Damit liefert die Implementation die gewünschten Ergebnisse:

```
?- phrase('S',"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa").

Yes
?- phrase('S',"aaaabaaaaaaaaaaaaaaaaaaaaaaaaa").

No
```

1.6 Beispiel 4

Zum Schluss wird an einem elementaren Beispiel gezeigt, wie man die DCG-Notation mit der herkömmlichen Prädikatschreibweise kombinieren kann. Dazu werden Zeichenketten überprüft, ob sie Hexadezimalzahlen darstellen. Falls ja, wird zugleich ihr Dezimalwert berechnet und ausgegeben. Wir beschränken uns hier auf Hexadezimalzahlen mit maximal drei Stellen. Die Grammatik selbst ist problemlos:

$G_4 = (V,\ \Sigma,\ P,\ Hex\_zahl)$ mit $V = \{Ziffer, Hex\_zahl\}$,
$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$, und den Produktionen
$Hex\_zahl \rightarrow Ziffer \mid Ziffer,\ Ziffer \mid Ziffer,\ Ziffer,\ Ziffer,$
$Ziffer \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \mid A \mid B \mid C \mid D \mid E \mid F.$

Auf der Implementation (Beispiel 4.1), die nichts Neues liefert,

```
/***      Beispiel 4.1              ***
 ***      Datei beisp4_1.pl         ***/

'Hex_zahl' --> 'Ziffer' | 'Ziffer', 'Ziffer'
                        | 'Ziffer', 'Ziffer', 'Ziffer'.

'Ziffer' --> "0" | "1" | "2" | "3" | "4" | "5" | "6" |
             "7" | "8" | "9" | "A" | "B" | "C" | "D" |
             "E" | "F" .
```

baut die nun folgende numerische Berechnung (Beispiel 4.2) auf. Arithmetische Wertzuweisungen erfolgen in Prolog durch das zweistellige Standardprädikat `is/2`, welches sowohl in der Präfix- als auch vorzugsweise in der Infixschreibweise wie folgt benutzt wird:

```
?- is(X, 1234 + 234).

X = 1468 ;

No
?- X is 1234 + 234.

X = 1468 ;

No
```

Wir ergänzen nun alle nicht terminalen Symbole im Beispiel 4.1 durch genau ein Argument in runden Klammern, welches den Dezimalwert des Symbols angibt. Der Dezimalwert der Hexadezimalzahl errechnet sich aus den Werten, die die einzelnen Ziffern repräsentieren. Diese Berechnungen sind, als "normale" Prolog-Anfragen in geschweifte Klammern zu setzen und stehen auf den rechten Seiten der Grammatikregeln:

```
/***      Beispiel 4.2              ***
 ***      Datei beisp4_2.pl         ***/

'Hex_zahl'(Wert) --> 'Ziffer'(Wert)

                   | 'Ziffer'(Wert2), 'Ziffer'(Wert1),
                     {Wert is Wert2 * 16 + Wert1}

                   | 'Ziffer'(Wert3), 'Ziffer'(Wert2), 'Ziffer'(Wert1),
                     {Wert is Wert3 * 16**2 + Wert2 * 16 +  Wert1 }.

'Ziffer'(0) -->  "0".
'Ziffer'(1) -->  "1".
'Ziffer'(2) -->  "2".
'Ziffer'(3) -->  "3".
'Ziffer'(4) -->  "4".
'Ziffer'(5) -->  "5".
'Ziffer'(6) -->  "6".
'Ziffer'(7) -->  "7".
'Ziffer'(8) -->  "8".
'Ziffer'(9) -->  "9".
'Ziffer'(10) --> "A".
'Ziffer'(11) --> "B".
'Ziffer'(12) --> "C".
'Ziffer'(13) --> "D".
'Ziffer'(14) --> "E".
```

```
'Ziffer'(15) --> "F".
```

Eine Anfrage liefert jetzt, neben der syntaktischen Analyse, gleichzeitig den Dezimalwert der untersuchten Zeichenkette:

```
?- phrase('Hex_zahl'(Dezimalwert),"B6A").

Dezimalwert = 2922 ;

No
?- phrase('Hex_zahl'(Dezimalwert),"B6G").

No
```

1.7 Zusammenfassung

Die DCG-Notation wurde bisher in der fachdidaktischen Diskussion kaum berücksichtigt. Die Einfachheit der Implementation ermöglicht eine schülerzentrierte Unterrichtsform mit ausgeprägten forschend-entdeckenden Phasen. Letzteres gilt grundsätzlich beim Einsatz eines wissensbasierten Systems im Unterricht. Die eigenen unterrichtlichen Umsetzungen in einem Grundkurs zum "wissensbasierten Ansatz" sind ermutigend und wurden auf dem Bildungsserver des Landes Nordrhein-Westfalen
http://www.learn-line.nrw.de/angebote/werkstattzbw/prolog/docs/start.htm
dokumentiert. Dort finden sich auch Übungs- und Klausuraufgaben. Entwurf, Analyse und Implementation von Grammatiken und die Einordnung der Sprachen in das Ordnungsschema nach Chomsky, lieferten Aufgabenstellungen für mündliche und schriftliche Abituraufgaben.

Literatur

[Bau93] BAUMANN, RÜDEGER: *Informatik für die Sekundarstufe II, Band 2*. Klett, Stuttgart, Düsseldorf, Berlin, Leipzig, 1993. 2, 5

[FB96] FLOYD, ROBERT W. und RICHARD BEIGEL: *Die Sprache der Maschinen.* International Thomson Publishing, 1996. 2

[MSW99] MSWWF: *Richtlinien und Lehrpläne für die Sekundarstufe II Gymnasium/Gesamtschule in Nordrhein-Westfalen - Informatik.* Ritterbach Verlag GmbH, 1999. (Ministerium für Schule und Weiterbildung, Wissenschaft und Forschung des Landes Nordrhein - Westfalen). 1

[Sch95] SCHÖNING, UWE: *Theoretische Informatik - kurzgefaßt.* Spectrum, Akad. Verl., Heidelberg; Berlin; Oxford, 1995. 2

[Weg99] WEGENER, INGO: *Theoretische Informatik - eine algorithmische Einführung.* B.G.Teubner, Struttgart, Leipzig, 1999. 2

[Wei97] WEISWEBER, WILHELM: *Prolog - Logische Programmierung in der Praxis.* Internat. Thomson Publ., Bonn, Albany ,Attenkirchen, 1997. 4