8.5 Bison, ein LALR(1)-Parsergenerator

Bison ist ein LALR(1)-Parsergenerator, der ein Teil des GNU-Systems der Open Software Foundation ist. Er ist aufwärts kompatibel mit dem LALR(1)-Parsergenerator Yacc, einem Unix-Dienstprogramm. Wie Yacc erlaubt auch Bison die Eingabe mehrdeutiger Grammatiken und die Beseitung der daraus resultierenden Parser-Konflikte, insbesondere bei Operatoren, durch die Angabe von Präzedenzen und Assoziativitäten. Außerdem bietet Bison die Erzeugung eines rudimentären Mechanismus zur Behandlung von Syntaxfehlern an.

Bison-erzeugte Parser haben eine Schnittstelle zu C. Mit der Reduktion einer Produktion kann der Aufruf einer benutzerdefinierten C-Funktion verbunden werden. Es gibt eine vordefinierte s-Attributierung; zu jedem Grammatiksymbol gibt es genau ein abgeleitetes Attribut. Sein Vorkommen auf einer linken Produktionenseite wird mit '$$', sein Vorkommen beim i-ten Symbol der rechten Seite mit '$i' bezeichnet. Standardtyp dieses Attributs bei allen Symbolen ist *int*, jedoch sind abweichende Typvereinbarungen, auch verschiedene für verschiedene Symbole möglich.

Von Bison erzeugte Parser heißen `yyparse`. Sie erwarten die Existenz eines Scanners namens `yylex`. Dieser kann von z.B. Flex erzeugt oder auch handgeschrieben sein.

8.5.1 Bison-Eingabe

Die Eingabedatei für Bison hat das folgende Format:

%{
C Deklarationen
%}
Bison Deklarationen
%%
Grammatik-Regeln
%%
Benutzerdefinierter C-Code

Der *C-Deklarationsteil* führt Typen und Variablen ein, die in den semantischen Regeln benutzt werden. Außerdem stehen hier Macrodefinitionen und die notwendigen `includes`.

Der *Bison-Deklarationsteil* listet die Terminal- und Nichtterminalalphabete auf und ordnet dabei eventuell den Attributen der jeweiligen Grammatiksymbole Typen zu. Weiterhin können hier die Präzedenzen und Assoziativitäten von Operatoren angegeben werden.
Beispiel:

```
%token NUM      /* Symbolklasse ganze Zahlen */
%left '-' '+' /* linksassoziative Operatoren */
%left '*' '/'
```



```
%left NEG     /* unaeres Minus */
%right '^'    /* rechtsassoziativer Potenz-Operator */
```

Die Auflistung der Operatoren bestimmt die Präzedenzen; Operatoren in der gleichen Zeile haben die gleiche Präzedenz, später eingeführte höhere als früher eingeführte. NEG wird als Präzedenzstufe eingeführt, um das unäre Minus einzuordnen. Dieses kann erst aufgrund der syntaktischen Position vom binären Minus unterschieden werden.

Es folgt ein Beispiel für die Verwendung verschiedener Attributtypen. Nehmen wir an, daß ein abstrakter Syntaxbaum gesteuert durch die Syntaxanalyse aufgebaut werden soll. Dann würde der Knoten zu einer Anweisung Verweise auf seine Komponentenbäume haben, der Knoten zu einem Bezeichner einen Verweis in die Symboltabelle und der Knoten zu einer Konstanten einen Verweis auf eine Darstellung der Konstanten. Die Attribute zu den entsprechenden Nichtterminalen wären dann Zeiger auf `structs` mit den erforderlichen Komponenten. Durch eine *union*-Deklaration im Bison-Deklarationsteil wird die Menge der möglichen Attributtypen eingeführt.

```
%union{
   symrec *tptr;
   snode  *nptr;
   value  *vptr;
}
```

Den Attributen der Nichtterminale wird anschließend jeweils einer der so eingeführten Attributtypen zugeordnet.

```
%type <tptr> identifier
%type <nptr> statement, ifstat, whilestat
%type <vptr> const
```

Die *Grammatikregeln* geben die Produktionen und die mit ihnen assoziierten semantischen Aktionen an.
Beispiel: Eine Grammatik für variablenfreie, arithmetische Ausdrücke mit semantischen Regeln zu ihrer Auswertung.

```
expr      NUM                { $$ = $1;      }
        | exp '+' exp        { $$ = $1 + $3; }
        | exp '-' exp        { $$ = $1 - $3; }
        | exp '*" exp        { $$ = $1 * $3; }
        | exp '/' exp        { $$ = $1 / $3; }
        | '-' exp %prec NEG  { $$ = - $2;    }
        | exp '^' exp        { $$ = pow ( $1, $3 );}
        | '(' exp ')'        { $$ = $2;      }
;
```

Die Mehrdeutigkeiten dieser Grammatik werden, wie oben gesagt, durch die Assoziativitätsregeln und die Präzedenzen aufgelöst.

Der *benutzerdefinierte C-Code* enthält die Definitionen der Aktionen, die in Grammatikregeln auftreten, eventuell die Definition eines Scanners `yylex` und von weiteren benötigten Funktionen.

8.5.2 Fehlerbehandlung

Findet ein Bison-erzeugter Parser einen syntaktischen Fehler, so ruft er eine Routine `yyerror` auf, die eine Fehlermeldung ausgibt und normalerweise die Syntaxanalyse beendet.

Bison unterstützt jedoch wie Yacc die folgende einfache Art der Behandlung von Syntaxfehlern. Es gibt ein vordefiniertes Symbol `error`. Dieses Symbol kann an beliebiger Stelle auf rechten Seiten von Produktionen stehen. Der Bison-erzeugte Parser meldet ein Vorkommen dieses Symbols, wenn er keinen legalen Übergang hat. Enthält der aktuelle Zustand ein Item, in dem `error` auf den Punkt folgt, so wird ein Übergang unter `error` ausgeführt. Folgt eine Reduktion mithilfe dieser Produktion, so kann mit der zugehörigen Aktion eine Fehlerbehandlung vorgenommen werden. Die vordefinierte Routine `yyerrok` veranlaßt den Parser zur Fortsetzung der Analyse.

8.6 Übungen

2.1 Geben Sie eine Definition der Zukunft einer Folge von Items, $fut(\gamma)$, so daß Sie die folgende Invariante (I') beweisen können:
(I') Für alle Sätze $uv \in L(G)$ gibt es ein $\gamma \in It_G^*$ mit der Eigenschaft:
Aus $(q_0, uv) \vdash_{\kappa_G}^* (\gamma, v)$ folgt $fut(\gamma) \stackrel{*}{\Longrightarrow} v$.

2.2 Definieren Sie $LAST_k$ induktiv

2.3 Definieren Sie $PRECEDE_k$ induktiv

2.4 Ein Kellerautomat mit Ausgabe (Definition 8.2.13) gibt die Nummern der angewendeten Produktionen aus. Aus dieser Ausgabe läßt sich der Syntaxbaum zu einem analysierten Wort erstellen.
In vielen Anwendungen soll der Syntaxbaum gleichzeitig mit der Analyse erstellt werden.
Sei folgendes eine Signatur für Syntaxbäume zu der kontextfreien Grammatik

$$G = (\{N_1, \ldots, N_k\}, \{t_1, \ldots, t_l\}, P, N_1).$$

$$\begin{array}{lll} \underline{\textbf{syntaxtree}}(G) = & & \\ \quad \textbf{sorts:} & V_N & \\ & V_T & \\ & \mathit{syntree}(G) & \\ \quad \textbf{opns:} & N_1, \ldots, N_k : & \to V_N \\ & t_1, \ldots, t_l : & \to V_T \end{array}$$

$$\begin{array}{lll} \mathit{LEAF}: & V_T & \to \mathit{syntree} \\ \forall p \in P, p \equiv N_i \to b_1 \ldots b_n, b \in V_T \cup V_N : & & \\ \mathit{NODE}_p : V_N\ \mathit{syntree}^n & \to & \mathit{syntree} \end{array}$$

Sei X eine abzählbar unendliche Variablenmenge zu $\mathit{syntaxtree}(G)$.
$T_{\text{syntaxtree}(G)}$ ist die Menge aller Syntaxbäume (Grundterme),
$T_{\text{syntaxtree}(G)}(X)$ die Menge aller Syntaxbäume mit Variablen.
Geben Sie die Definition, die Übergangsrelation aus Beispiel 8.2.9 und Konfigurationenfolge nach Tabelle 8.2 des Kellerautomaten mit Baumausgabe an für
(a) Linksparser,
(b) Rechtsparser.

2.5
(a) Konstruieren Sie den Item-Kellerautomaten zu

$$G = (\{S\},\ \{\textbf{if}\ ,\ \textbf{then}\ ,\ \textbf{else}\ ,\ a,\ b\}, \left\{ \begin{array}{lll} S & \to & a \\ S & \to & \textbf{if}\ b\ \textbf{then}\ S \\ S & \to & \textbf{if}\ b\ \textbf{then}\ S\ \textbf{else}\ S \end{array} \right\}, S)$$

(b) Geben Sie eine akzeptierende Konfigurationenfolge für

$$\textbf{if}\ b\ \textbf{then}\ \textbf{if}\ b\ \textbf{then}\ a\ \textbf{else}\ a$$

an.
(c) Zeigen Sie, daß G mehrdeutig ist.
(d) Geben Sie eine eindeutige Grammatik G' an mit $L(G') = L(G)$.

2.6
(a) Konstruieren Sie den Item-Kellerautomaten zu

$$G = (\{S, A, B, C\},\ \{a, b\}, \left\{ \begin{array}{lll} S & \to & AB \mid BC \\ A & \to & BA \mid a \\ B & \to & CC \mid b \\ C & \to & AB \mid a \end{array} \right\}, S)$$

(b) Wieviele akzeptierende Konfigurationsfolgen gibt es für *babaab*?

2.7
(a) Geben Sie einen effizienten Algorithmus zur Bestimmung der erreichbaren Nichtterminale einer Grammatik an.
Hinweis: Man durchlaufe die Grammatik geeignet.
(b) Überprüfen Sie mit Hilfe dieses Verfahrens
(α) die Produktivität und
(β) die Erreichbarkeit

der Nichtterminale der Grammatik

$$G = (\{S, A, B, C, D, E\},\ \{a, b, c\}, \left\{ \begin{array}{lcl} S & \to & aAa \mid bS \\ A & \to & BB \mid C \\ B & \to & bC \\ C & \to & B \mid c \\ D & \to & aAE \\ E & \to & Db \end{array} \right\}, S)$$

2.8 Gegeben sei die folgende Grammatik:

$$G = (\{S', S, B, E, J, L\}, \{;, :=, (,), ,, \}, \left\{ \begin{array}{lcl} S' & \to & S \\ S & \to & LB \\ B & \to & ;S;L| := L \\ E & \to & a|L \\ J & \to & ,EJ|) \\ L & \to & (EJ \end{array} \right\}, S')$$

(a) Berechnen Sie $FIRST_1$ und $FOLLOW_1$ mit den angegebenen Gesamtschrittverfahren.
(b) Berechnen Sie $FIRST_1$ und $FOLLOW_1$ mit Hilfe von geeigneten Einzelschrittverfahren.
(c) Geben Sie für die Grammatik G die Relationen R_{Fi} und R_{Fo} und die Funktionen g_{Fi} und g_{Fo} an, die bei der Berechnung von $FIRST_1$ und $FOLLOW_1$ als reine Vereinigungsprobleme gebraucht werden.

2.9 Am Beispiel 8.2.17 erkennt man, daß man nach dem Entfernen der nicht erreichbaren und dann dem Entfernen der unproduktiven Nichtterminale i.a. keine reduzierte Grammatik erhält.

Zeigen Sie, daß man durch die Umkehrung der Entfernungsreihenfolge immer eine reduzierte Grammatik erhält.

2.10 Beweisen Sie die Assoziativität von $\oplus_k$.

2.11 Berechnen Sie zu

$$G = (\{S, A, B\},\ \{a, b\}, \left\{ \begin{array}{lcl} S & \to & aAaB \mid bAbB \\ A & \to & a \mid ab \\ B & \to & aB \mid a \end{array} \right\}, S)$$

(a) R_{Fi} und den von R_{Fi} induzierten Graphen.
(b) R_{Fo} und den von R_{Fo} induzierten Graphen. Wenden Sie Algorithmus SZK auf die Graphen an, um
(i) $FIRST_1$
(ii) $FOLLOW_1$
zu den Nichtterminalen von G zu erhalten.

2.12 Definieren sie *ε-ffi* und $FOLLOW_1$ für die rechtsregulären kontextfreien Grammatiken als reines Vereinigungsproblem.

3.1 Testen Sie die LL(1)-Eigenschaft von
(a) G aus Aufgabe 2.5.
(b) G aus Aufgabe 2.6.
(c) G aus Aufgabe 2.8.
(d)

$$G = (\{E, E', D, D', F\},\ \{a, (,), +, *\}, \left\{ \begin{array}{lcl} E & \to & DE' \\ E' & \to & +DE' \mid \varepsilon \\ D & \to & FD' \\ D' & \to & *FD' \mid \varepsilon \\ F & \to & (E) \mid a \end{array} \right\}, E)$$

3.2
(a) Stellen Sie die LL(1)-Parsertabelle für G aus Aufgabe 3.1. (d) auf.
(b) Geben Sie einen Lauf des zugehörigen Parsers für die Eingabe $(a + a) * a + a$ an.

3.3 Geben Sie die LL(1)-Tabelle für die folgende Grammatik an:

$$\begin{array}{lcl} E & \to & -E|(E)|VE' \\ E' & \to & -E|\varepsilon \\ V & \to & \mathbf{id}\ V' \\ V' & \to & (E)|\varepsilon \end{array}$$

Skizzieren Sie einen Lauf des Parsers für die Eingabe $-$ **id** $(-$ **id**$) -$ **id**.

3.4 Formulieren Sie
(a) $FIRST_1$
(b) $FOLLOW_1$
für rechtsreguläre kontextfreie Grammatiken als reines Vereinigungsproblem.

3.5 Berechnen Sie die ε-Produktivität, die ε-freie first-Funktion und $FOLLOW_1$ für folgende rechtsregulären kontextfreie Grammatiken
(a) die rechtsreguläre kontextfreie Grammatik für arithmetische Ausdrücke (Beispiel 8.3.10)
(b) die Grammatik

$$G = (\{S, A, B\}, \{c, d, e\}, \left\{ \begin{array}{lcl} S & \to & c\{A \mid \varepsilon\} \\ A & \to & \{B\}^*Sd \\ B & \to & S \mid Ae \end{array} \right\}, S)$$

Sind die beiden Grammatiken RLL(1)?

3.6
(a) Berechnen Sie ε-Produktivität, die ε-freie first-Funktion und $FOLLOW_1$ für die folgende rechtsreguläre kontextfreie Grammatik:

$$\begin{array}{lcl} S & \to & E \\ E & \to & T(+T)^* \\ T & \to & F(*F)^* \\ F & \to & -E| \ \mathbf{if}\ E\ \mathbf{then}\ E\ \mathbf{else}\ E|CC^* \\ C & \to & \mathbf{id}\ |'('E')'|[(E(','E)^*)|\varepsilon] \end{array}$$

(b) Überprüfen Sie, ob es sich um eine RLL(1)-Grammatik handelt.
(c) Geben Sie den RLL(1)-Parser in der Tabellenform an.
(d) Geben Sie den RLL(1)-Parser als Programm an.

3.7 Wir definieren die $LAST_1$-Menge eines Nichtterminals X durch

$$LAST_1(X) = \{w : 1 | X \stackrel{*}{\Longrightarrow} w\}$$

Dabei sei $w : 1$ der 1-Suffix des Wortes w, der analog zum 1-Präfix definiert ist.
(a) Geben Sie das GFA-Problem für $LAST_1$ für eine kontextfreie Grammatik G an.
(b) Geben Sie $LAST_1$ in Form eines reinen Vereinigungsproblems an.

3.8 Geben Sie ein modifiziertes Schema zur Generierung von recursive descent RLL(1)-Parsern an, welches direkt aufeinanderfolgende Abfragen auf das gleiche Symbol vermeidet.

4.1 Untersuchen Sie, ob eine kfG mit den folgenden Produktionen eine LR(k)-Grammatik sein kann. Begründen sie Ihre Antwort.

object-declaration → identifier-list : subtype-indication
renaming-declaration → identifier : type-mark **renames** object-name
identifier-list → identifier | identifier-list , identifier
subtype-indication → type-mark

4.2 Welche der folgenden Grammatiken sind keine LR(0)-Grammatiken. Begründen Sie Ihre Antwort.

(a)
$S \rightarrow L$
$L \rightarrow L;A|A$
$A \rightarrow a$

(b)
$S \rightarrow L$
$L \rightarrow A;L|A$
$A \rightarrow a$

(c)
$S \rightarrow L$
$L \rightarrow L;L|A$
$A \rightarrow a$

(d)
$S \rightarrow L$
$L \rightarrow aT$
$T \rightarrow \varepsilon|;L$

4.3 Zeigen Sie, daß die folgende Grammatik SLR(1) ist, und geben Sie die Action-Tabelle an:

$$\begin{array}{lcl} S & \rightarrow & E \\ E & \rightarrow & T|E+T \\ T & \rightarrow & P|T*P \\ P & \rightarrow & F|F\uparrow P \\ F & \rightarrow & id\,|(E) \end{array}$$

4.4 Zeigen Sie, daß die folgende Grammatik LL(1), aber nicht SLR(1) ist:

$$\begin{array}{lcl} S & \rightarrow & AaAb|BbBa \\ A & \rightarrow & \varepsilon \\ B & \rightarrow & \varepsilon \end{array}$$

4.5 Zeigen Sie, daß die folgende Grammatik LALR(1), aber nicht SLR(1) ist:

$$\begin{array}{lcl} S & \rightarrow & Aa|bAc|dc|bda \\ A & \rightarrow & d \end{array}$$

4.6 Zeigen Sie, daß die folgende Grammatik LR(1), aber nicht LALR(1) ist:

$$\begin{array}{lcl} S & \rightarrow & Aa|bAc|Bc|bBa \\ A & \rightarrow & d \\ B & \rightarrow & d \end{array}$$

4.7 Gegeben sei die folgende Grammatik:

$$\begin{array}{lcl} S & \rightarrow & A \\ A & \rightarrow & bB \\ B & \rightarrow & cC \\ B & \rightarrow & cCe \\ C & \rightarrow & dA \\ A & \rightarrow & a \end{array}$$

(a) Berechnen Sie die Menge der LR(1)-Items.
(b) Ist die Grammatik SLR(1)?
(c) Ist die Grammatik LALR(1)?
(d) Ist die Grammatik LR(1)?

4.8 Gegeben sei die folgende Grammatik:

$$\begin{array}{lcl} S & \rightarrow & A \\ B & \rightarrow & \varepsilon \\ C & \rightarrow & \varepsilon \\ A & \rightarrow & BCA \\ A & \rightarrow & a \end{array}$$

(a) Berechnen Sie die Menge der LR(1)-Items. Ist die Grammatik LR(1)?
(b) Konstruieren Sie den LR_DEA für die obige Grammatik. Berechnen Sie für die ungeeigneten Zustände die LALR(1)-Vorausschaumengen mit Hilfe des in Abschnitt 8.4.5 angegebenen effizienten Verfahrens.

8.7 Literaturhinweise

Ausführliche Darstellungen der Theorie der formalen Sprachen und der Automaten finden sich in den Büchern von Hopcroft und Ullman [HU79] und Harrison [Har83]. Ganz dem Gebiet der Syntaxanalyse gewidmet sind die Bücher [May78], [SSS90a] und [SSS90b].

Die Grammatikflußanalyse wurde in [MW82] erstmals beschrieben und in [Mön85] und [MW91] weiter ausgearbeitet. D. E. Knuth stellte in [Knu77] einen verwandten Ansatz vor, allerdings auf total geordneten Mengen. Ein ähnlicher Ansatz wurde in mehreren Arbeiten von Courcelle, z.B. in [Cou86] verfolgt.

LL(k)-Grammatiken wurden von Lewis und Stearns eingeführt [IS66], [IS68]. Heckmann [Hec86] entwickelte einen effizienten RLL(1)-Parsergenerator, der die