Compilerkonstruktion

Wintersemester 2015/16

Prof. Dr. R. Parchmann

02. Februar 2016

Generatoren für die Maschinencode-Erzeugung

Ziel: Die automatische Generierung eines Maschinencode-Erzeugers aus einer Beschreibung der Zielmaschine.
Dazu existieren mehrere wichtige Ansätze:

- über Baum-Grammatiken (allgemeiner Ansatz von Cattell)
- über SDTS und LR-Grammatiken für sequentiell dargestellte Syntaxbäume (Ansatz von Graham und Glanville)
- über Baum-Matching in Verbindung mit dynamischer Programmierung (Ansatz von Aho, Ganapathi, Tjiang)

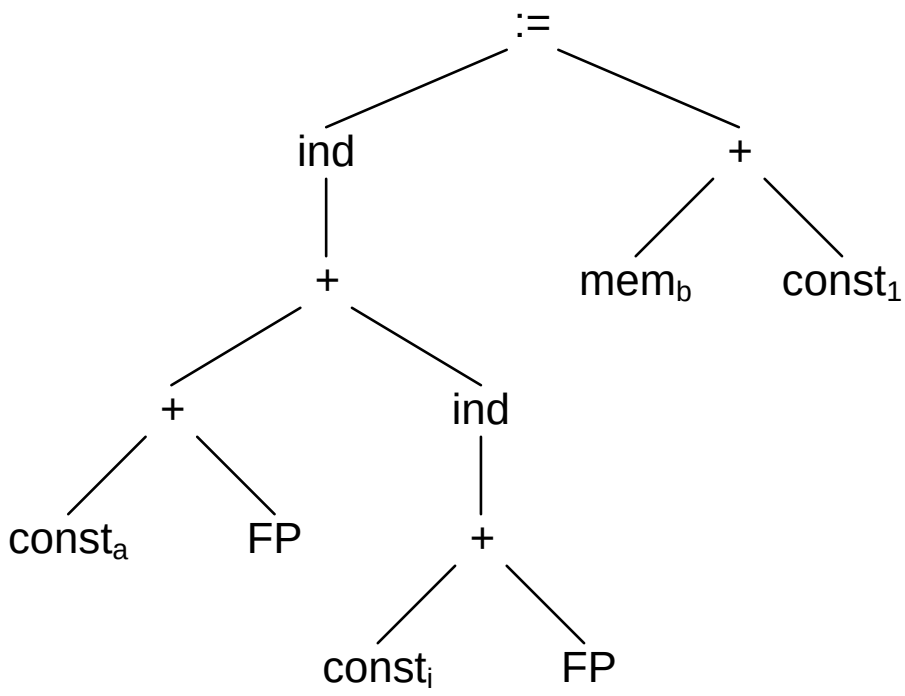
Bei allen Ansätzen müssen die Syntaxbäume relativ weit „aufgefächert“ sein, denn in allen Fällen wird die Arbeitsweise einzelner Maschinenbefehle durch Bäume bzw. Zeichenketten dargestellt.

Beispiel (Syntaxbaum für `a[i] := b + 1`)

Annahmen:

- `b` ist eine globale Variable
- `a` und `i` sind lokal auf dem Laufzeit-Stack gespeichert.
- Das Feld `a` hat als unteren Index 0.
- Alle Variablen sind vom Typ `character` bzw. `integer`.
- $\texttt{const}_a$ und $\texttt{const}_i$ sind die Abstände der Speicherorte von `a` und `i` relativ zum Framepointer `FP`,
- $\texttt{mem}_b$ ist die Speicheradresse von `b`.

Beispiel (Syntaxbaum für `a[i] := b + 1`)



Ansatz von Cattell - Verwendung von Baum-Grammatiken

Die prinzipielle Vorgehensweise ist wie bei üblichen kontextfreien Grammatiken, nur werden statt Zeichenketten Bäume erzeugt. Für die Übersetzung benötigt man Produktionen der Form:

$$
\begin{aligned}
A &\longrightarrow T\{\text{Regel}\} \quad \text{oder} \\
A &\longrightarrow T\{\text{Kosten}\}\{\text{Regel}\}.
\end{aligned}
$$

Dabei ist A ein nichtterminales Symbol, T ist ein Baum, dessen Blätter mit terminalen oder nichtterminalen Symbolen markiert sind, und alle internen Knoten von T sind mit terminalen Symbolen markiert. Die „Regel“ gibt die semantische Aktion an.

Häufig sind derartige Baumgrammatiken, die die Wirkungsweise der Maschinenbefehle eines Rechners beschreiben, stark mehrdeutig.
Um die verschiedenen Möglichkeiten bewerten zu können, wird den Produktionen auch noch eine Kostenfunktion zugeordnet, die dann zur Auswahl der „richtigen“ Produktionen benutzt wird.

Beispiel

Bemerkung

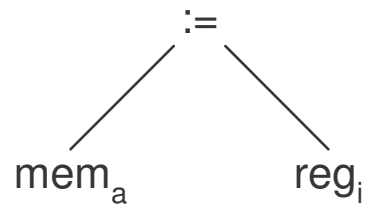
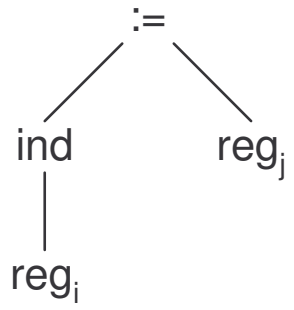
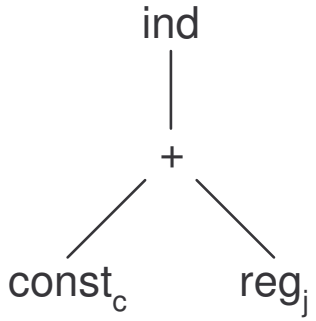
Die tiefgestellten Buchstaben haben die Bedeutung von Zusatzbedingungen bzw. semantischen Prädikaten (Attributen).

Beispiel (Baumgrammatik)

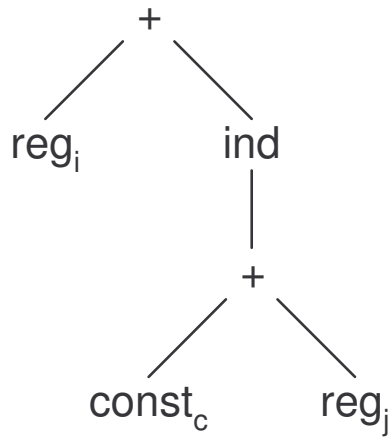
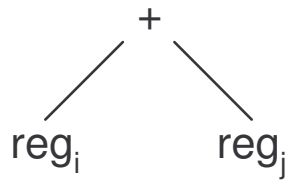
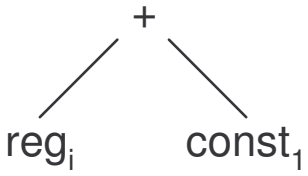
In den Regeln wird hier zur besseren Lesbarkeit statt einer ausführbaren Anweisung der erzeugte Assemblercode angegeben.

Nr.	**Produktion**	**Regel**, (erzeugter Befehl)
1	$\texttt{reg}_i \rightarrow \texttt{const}_c$	{ `LD Ri,#c` }
2	$\texttt{reg}_i \rightarrow \texttt{mem}_a$	{ `LD Ri,a` }
3	$\texttt{reg}_{FP} \rightarrow \texttt{FP}$	

Beispiel (Fortsetzung)

Nr.	**Produktion**	**Regel**, (erzeugter Befehl)
4	`mem`$\rightarrow$ 	{ `ST a,Ri` }
5	`mem`$\rightarrow$ 	{ `ST *Ri,Rj` }
6	`reg`$_i$$\rightarrow$ 	{ `LD Ri,c(Rj)` }

Beispiel (Fortsetzung)

Nr.	**Produktion**	**Regel**, (erzeugter Befehl)
7	$\texttt{reg}_k \rightarrow$ 	{ `ADD Rk,Ri,c(Rj)` }
8	$\texttt{reg}_k \rightarrow$ 	{ `ADD Rk,Ri,Rj` }
9	$\texttt{reg}_i \rightarrow$ 	{ `INC Ri` }

Übersetzungen mit Baumgrammatiken

Prinzip: Parsen des gegebenen Syntax-Baums und bei Erkennen einer Produktion Ausführen der entsprechenden semantischen Aktionen.
Ansatz: Bottom-Up Verfahren – der vorgelegte Syntaxbaum wird von unten nach oben mit Hilfe der gegebenen Regeln reduziert.
Dabei stellen sich folgende Fragen:

1. In welcher Reihenfolge sollen die Reduktionen durchgeführt werden?
2. Was ist zu tun, wenn mehrere Regeln anwendbar sind?
3. Was ist zu tun, wenn keine Regel anwendbar ist?
4. Wie verhütet man unendliche Schleifen in der Reduktionsphase?

Der Ansatz von Graham und Glanville

Ein anderer Ansatz transformiert den vorliegenden Eingabebaum in eine Zeichenkette und verwendet die üblichen Parsermethoden, z.B. den Parsergenerator `yacc`. Bei den Baum-Produktionen müssen die rechten Seiten natürlich ebenfalls transformiert werden, so dass „übliche“ kontextfreie Produktionen entstehen.

Beispiel

Als Transformation verwenden wir die sequentielle Darstellung des Baumes in Präfix-Notation. Da der Verzweigunggrad der internen Knoten implizit gegeben ist, müssen keine weiteren Informationen abgespeichert werden. Die sequentielle Darstellung des Syntaxbaums wäre dann:

$$:= \texttt{ind} + + \texttt{const}_a \ \texttt{FP} \ \texttt{ind} + \texttt{const}_i \ \texttt{FP} + \texttt{mem}_b \ \texttt{const}_1$$

Die transformierte Baum-Grammatik wird als SDTS umgeschreiben:

Nr.	Produktion	Regel (erzeugter Befehl)
(1)	$\texttt{reg}_i \rightarrow \texttt{const}_c$	`LD Ri,#c`
(2)	$\texttt{reg}_i \rightarrow \texttt{mem}_a$	`LD Ri,a`
(3)	$\texttt{reg}_{FP} \rightarrow \texttt{FP}$	
(4)	$\texttt{mem} \rightarrow \texttt{:= mem}_a\ \texttt{reg}_i$	`ST a,Ri`
(5)	$\texttt{mem} \rightarrow \texttt{:= ind reg}_i\ \texttt{reg}_j$	`ST *Ri,Rj`
(6)	$\texttt{reg}_i \rightarrow \texttt{ind + const}_c\ \texttt{reg}_j$	`LD Ri,c(Rj)`
(7)	$\texttt{reg}_k \rightarrow \texttt{+ reg}_i\ \texttt{ind + const}_c\ \texttt{reg}_j$	`ADD Rk,Ri,c(Rj)`
(8)	$\texttt{reg}_k \rightarrow \texttt{+ reg}_i\ \texttt{reg}_j$	`ADD Rk,Ri,Rj`
(9)	$\texttt{reg}_i \rightarrow \texttt{+ reg}_i\ \texttt{const}_1$	`INC Ri`

Probleme:

1. Die aus den Maschinenbefehlen konstruierte Grammatik ist meist hochgradig mehrdeutig. Hier hilft eventuell die heuristische Regel, möglichst viel in einem Schritt zu reduzieren. Also wird bei shift-reduce-Konflikten auf shift entschieden und bei reduce-reduce-Konflikten die längere Produktion bevorzugt.
2. Die Reihenfolge der Auswertung von Teilbäumen ist durch das Parsingverfahren festgelegt (links $\rightarrow$ rechts).
3. Es kann immer noch passieren, daß der Parsingprozess blockiert oder in eine Schleife gerät.
4. Die Registerzuweisung ist offen und muss von außen gesteuert werden.

Der Ansatz von Aho, Ganapathi und Tjiang

Der Ansatz beruht wiederum auf Baumgrammatiken. Es wird ein Baum-Übersetzungs-Schema (tree-translation-scheme) definiert und gezeigt, wie man mit Hilfe von Pattern-Matching und dynamischer Programmierung ein solches System implementiert werden kann. Dieses System namens twig ist zur Generierung von Maschinencode-Erzeugern für mehrere Compiler benutzt worden.

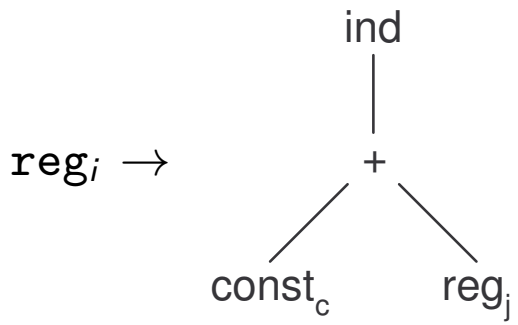
Eine allgemeine Produktion hat hat bei diesem Ansatz folgendes Aussehen:

$$\texttt{A} \rightarrow \texttt{T} \qquad \{\texttt{cost}\} = \{\texttt{action}\}$$

Dabei ist `A` ein nichtterminaler Knoten, `T` ein Baum, `cost` ist ein Code-Fragment, das zur Berechnung der Kosten dieses Musters aufgerufen wird. Fehlt dieses Feld, so werden Einheitskosten angenommen. `action` ist ein Code-Fragment, das bei Akzeptieren dieser Produktion ausgeführt wird und üblicherweise den Maschinencode generiert.

Beispiel

Die Produktion (6) aus dem vorigen Beispiel würde hier wie folgt dargestellt:



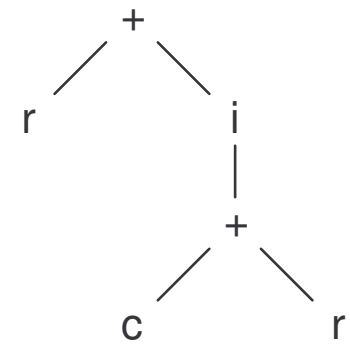
$\{\ \texttt{3+cost.reg}_j\ \} = \{\ \texttt{LD Ri,c(Rj)}\}$

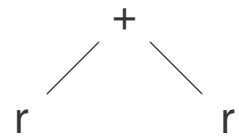
In die Berechnung der Kostenfunktion gehen in diesem Fall die notwendigen Kosten zur Berechnung des Wertes im Register `Rj` mit ein.

Idee: Die Bäume werden durch die Menge aller Wege von der Wurzel zum Blatt festgelegt, wobei die Verzweigungsnummern (die Nummer des Nachfolgers) auch mit notiert sind. Für die so erzeugten Pattern wird ein Aho-Corasick-Automat konstruiert, der eine parallele Suche nach all diesen Pattern im Baum erlaubt.

Beispiel

Man betrachte die folgenden drei Produktionen einer Baumgrammatik:

t_1 : r $\rightarrow$ 

t_2 : r $\rightarrow$ 

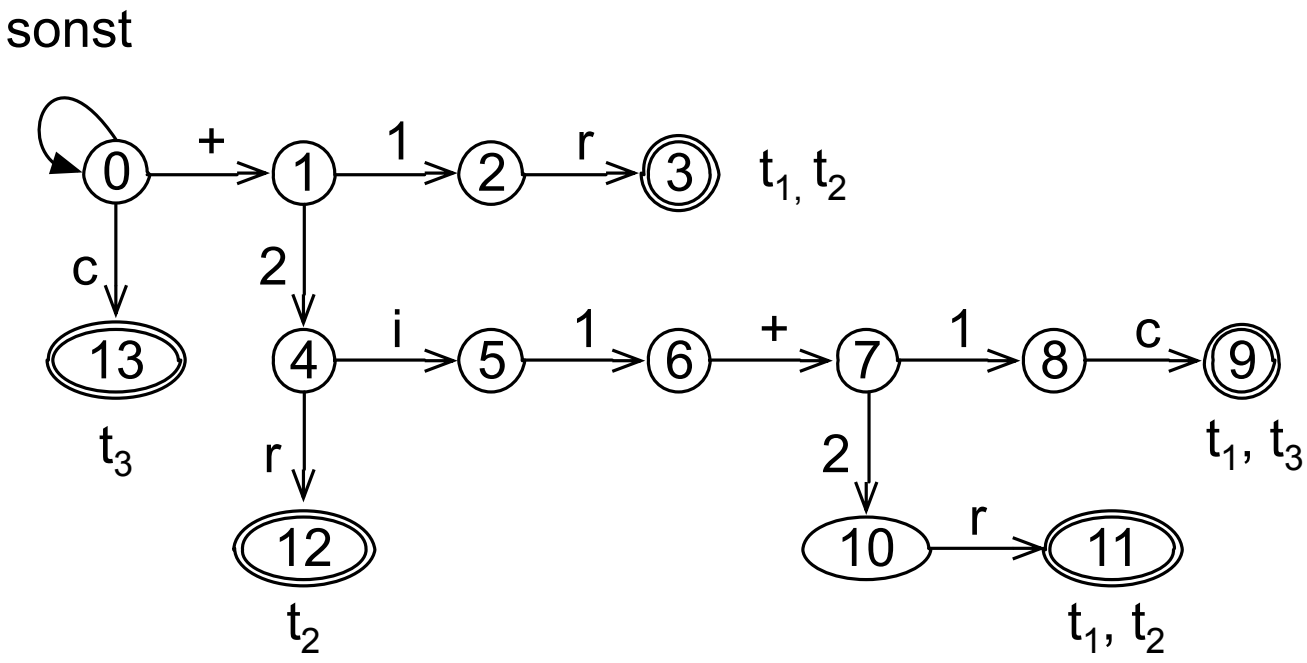
t_3 : r $\rightarrow$ c

Die Menge der Wege ist:

$$
\begin{array}{llllllll}
+ & 1 & \mathsf{r} & & & & & \\
+ & 2 & \mathsf{i} & 1 & + & 1 & \mathsf{c} & \\
+ & 2 & \mathsf{i} & 1 & + & 2 & \mathsf{r} & \\
+ & 2 & \mathsf{r} & & & & & \\
\mathsf{c} & & & & & & &
\end{array}
$$

wobei zu bemerken ist, dass der Weg `+ 1 r` sowohl zu t_1 als auch zu t_2 gehört.

Der zugehörige Aho-Corasick-Automat



Bemerkung

Die Übergänge der Fehlerfunktion sind weggelassen worden!