Prof. Dr. M. Jäger
FB MNI

Compilerbau - Praktikumsaufgabe 2 „SPL Parser“

Schritt 1 (Informieren)

Lesen Sie im Praktikumsskript das Kapitel zur Syntaxanalyse.

Beachten Sie, dass der Parsergenerator aus seiner Eingabedatei nicht nur den Parser erzeugt, sondern auch Schnittstellendefinitionen für den Scanner!

Schritt 2 (Scanner anpassen)

Der SPL-Scanner soll mit dem SPL-Parser zusammenarbeiten. Nehmen Sie deshalb die folgenden Änderungen an Ihrem SPL-Scanner vor:

Java:

a) Löschen Sie die Dateien ’sym.java’ und ’Symbol.java’. Die darin stehenden Informationen werden nun vom Parsergenerator Cup erzeugt bzw. bereitgestellt.

b) Ergänzen Sie ’scanner.jflex’ um die Import-Anweisung `import java_cup.runtime.*;` Damit steht die Cup-Laufzeitunterstützung zur Verfügung.

c) Ersetzen Sie in ’scanner.jflex’ die Zeilen

```
%type Symbol
%function next_token
%eofval{
  return symbol(sym.EOF);
%eofval}
%eofclose
```

durch die Zeile

```
%cup
```

Damit wird der Scannergenerator angewiesen, das von Cup verlangte Interface zu implementieren. (Das ist im Wesentlichen dasselbe, was vorher dort in einzelnen Direktiven stand).

C:

a) Löschen Sie die Datei ’preliminary.h’; die darin stehende Information wird nun vom Parsergenerator erzeugt und in die Datei ’parser.tab.h’ geschrieben.

b) Dementsprechend ersetzen Sie in ’scanner.l’ die Zeile

```
#include "preliminary.h"
```

durch die Zeile

```
#include "parser.tab.h"
```

c) Löschen Sie in ’scanner.l’ die Definition der Variablen ’yylval’; auch diese wird vom Parsergenerator definiert.

Schritt 3 (Grundlegende Bedienung des Parsergenerators)

Lesen Sie in den Unterlagen zum Parsergenerator *cup* (C: *bison*) die Einführung und die Grundlagen der Benutzung. Sie sollten wissen, wie man den Generator aufruft und was die in den vorgegebenen Skeletten für die Parserspezifikation 'parser.cup' (C: 'parser.y') vorhandenen Direktiven bedeuten.

Schritt 4 (Parserspezifikation)

Schreiben Sie die Spezifikation eines Parsers für SPL. Das ist zunächst nichts anderes als eine kontextfreie Grammatik für die Sprache. Orientieren Sie sich dabei an der Ihnen vorliegenden Sprachbeschreibung. Was dort informal beschrieben ist, müssen Sie jetzt in Ableitungsregeln umsetzen.

Ergänzen Sie dazu die Datei 'parser.cup' (C: 'parser.y'). Eliminieren Sie ALLE reduce/reduce-Konflikte, falls Sie welche bekommen. Es darf maximal EIN shift/reduce-Konflikt übrig bleiben. Welcher? Wie wird der aufgelöst? Warum ist das richtig?

Hinweis: Halten Sie sich in der Reihenfolge an die Sprachbeschreibung. Verwenden Sie aussagekräftige und eindeutige Nonterminalsymbole. Beispielsweise ist ein Nonterminal „Prozedur“ ungeschickt, weil der Leser nicht weiß, ob es sich um einen Prozeduraufruf oder eine Prozedurdefinition oder etwas anderes handelt.

Schritt 5 (SPL Testprogramme)

Testen Sie Ihren Parser sowohl mit den im vorigen Aufgabenblatt geschriebenen 5 Testprogrammen als auch mit einer Reihe von ganz kleinen Tests, die jeweils einen bestimmten Aspekt der Grammatik testen, den aber möglichst vollständig. Führen Sie diesen Schritt mit der gebotenen Sorgfalt aus.

Schritt 6 (Testausgabe des generierten Compilers aktivieren)

Machen Sie sich damit vertraut, wie man die Testausgabe des von 'cup' (C: 'bison') erzeugten Parsers aktiviert. Verfolgen Sie an den Tests, wie die syntaktische Analyse vonstatten geht!

Hinweis: Der generierte Parser ist ein „Shift-Reduce“-Parser, der nach dem LALR(1)-Verfahren arbeitet. Damit ist jeder Berechnungsschritt vom nächsten Token und vom aktuellen Zustand des DEA abhängig, der zur Steuerung der Berechnung dient. Zum Verfolgen einer Berechnung ist daher die Beschreibung des generierten DEA nötig. Sie müssen beim Aufruf des Generators explizit eine DEA-Beschreibung anfordern: *cup* mit `-dump` aufrufen, *bison* mit `-dtv` aufrufen.

Ein Protokoll einer Parserberechnung enthält für jeden Schritt die Information über den Parserstack und den aktuellen Zustand des DEA, das vom Scanner gelieferte Token und die vom Zustand und Token abhängige Parseraktion: SHIFT, REDUCE, ACCEPT oder ERROR.

Protokoll: *cup* Parser-Methode: `debug_parse` in `main.java` aufrufen (statt `parse`). Für *bison* notwendig: Präprozessorvariable `YYDEBUG` und Parservariable `yydebug` benutzen.

Ausblick

In der Praktikumsaufgabe 3 wird die Parserspezifikation erweitert. In die Ableitungsregeln der Grammatik werden dann noch semantische Aktionen eingefügt, die zum Aufbau des Abstrakten Syntaxbaums dienen.

Damit ist das Frontend des SPL-Compilers dann komplett. Dieses Frontend wird als erste Hausübung abgegeben.