Praktikum 2 zu TI 12.06.2018 u. 19.06.2018

SoSe 18

Ziele:
In diesem Versuch sollen Sie aufbauend auf Versuch 1 einen vollständigen Compiler für „WHILE0- nach URM-Quellcode" entwickeln.
Dieser URM-Quellcode soll anschließend im *URM Simulator* (siehe Downloadbereich) korrekt ausführbar sein. Der *URM Simulator* geht dabei davon aus, dass der (Eingabe-)URM-Quellcode **keine** Zeilennummern enthält (diese werden implizit vom *URM Simulator* gesetzt) und dass nur eine endliche, nicht-leere Folge von Befehlen folgt (**Programmspeicher**).
Der hier zu entwickelnde Compiler für „WHILE0- nach URM-Quellcode" ist entsprechend dieser Kriterien zu implementieren.

Gehen Sie in diesem Versuch wie folgt vor:

Schritt 1)

- Erweitern Sie im ersten Schritt die EBNF-Regeln mit Java-Code. Hierfür können Sie den Java-Code direkt an den richtigen Stellen in die Grammatik-Datei schreiben. Vermeiden Sie viel Code innerhalb der Grammatik-Regeln, in dem Sie zusätzlich Hilfsklassen definieren und verwenden
- Sie müssen die Zuweisungen von Variablen zu den entsprechenden URM-Registern speichern können. Hier bietet sich zur Implementierung einer Symboltabelle eine Hash-Map an, die beim Lesen der Variablendeklarationen aufgebaut und immer als Parameter der rekursiven Methoden weitergereicht wird
- Sie können den Teil-Code jeweils als Rückgabewert der JavaCC-Methoden übergeben (der Code wird also rekursiv erzeugt, alternativ kann auch eine Hilfsklasse verwendet werden) Hier muss es möglich sein temporäre Sprungmarken einzufügen und später durch Zeilennummern zu ersetzen (siehe Schritt 2)
- Erzeugen Sie für jede WHILE0-Anweisung gemäß der Transformationen im Skript URM-Code. Verwenden Sie dabei zur Übersetzung der While-Schleife Ihre Lösung aus Praktikum 1, Aufgabe 2
- Überlegen Sie sich sinnvolle Exception-Klassen

Schritt 2)
Eine Schwierigkeit bei der Entwicklung dieses Compilers ist die Übersetzung der Sprungmarken. URM unterstützt keine Textsprungmarken, sondern nur *direkte Zeilennummern* als Sprungziele.
Ein möglicher Lösungsansatz ist der Folgende:

- Erzeugen Sie in der Ausgabe zunächst temporäre, eindeutige Textsprungmarken (Marker)
- Ersetzen Sie am Schluss der Transformation die Marker durch die richtigen Zeilennummern (überlegen Sie sich hierfür, wie Sie an die Zeilennummern kommen)

Bei der Umsetzung einer Schleife in URM-Code ist demnach folgendes Vorgehen sinnvoll:

- Existieren die verwendeten Identifier überhaupt?
- Ermitteln des zugewiesenen Registers für die Variablen im Schleifenkopf
- Temporäre Sprungmarken erzeugen
- URM-Code für die Schleifenstruktur erzeugen
- Nachträglich: die temporären Sprungmarken durch Zeilennummern ersetzen