Praktikum zu Einführung in den Compilerbau

Prof. Dr.-Ing. Andreas Koch
Julian Oppermann, Lukas Sommer

TECHNISCHE UNIVERSITÄT DARMSTADT

Wintersemester 18/19
Praktikum 3

Abgabe bis Sonntag, 10.02.2019, 18:00 Uhr (MEZ)

Einleitung

Das Ziel dieses dritten Praktikums ist die Entwicklung einer Codegenerierung für einen MAVL-Compiler. Ihre Implementierung in einem kombinierten Visitor soll dabei Binärcode für eine modifizierte Version der Triangle Abstract Machine erzeugen. Eine Dokumentation der modifizierten TAM finden Sie im Moodle-Kurs.

Wir stellen Ihnen eine **erweiterte** Compilerumgebung zur Verfügung, die Sie als Archiv im Moodle-Kurs finden. Eine Anleitung, die Ihnen zeigt, wie Sie das Projekt auf Ihrem Rechner einrichten, bauen und ausführen können, ist im Archiv enthalten. Zusätzlich steht die Anleitung auch im Moodle-Kurs zur Ansicht bereit. Die erweiterte Compilerumgebung für dieses Praktikum enthält außerdem einen Interpreter, mit dem Sie Ihren erzeugten Binärcode ausführen können, sowie einen Disassembler, der den Binärcode in ein lesbares Textformat umwandelt. Die Verwendung von Interpreter und Disassembler ist ebenfalls in der erweiterten Anleitung dokumentiert.

In den folgenden Aufgaben werden Sie fehlende Methoden der Klasse `CodeGeneration` vervollständigen, die das Visitor-Interface der AST-Knoten-Klassen implementiert. Zum Erzeugen der TAM-Assembler-Befehle nutzen Sie dabei bitte stets die Funktionen der Klasse `TAMAssembler`. Die Javadoc-Dokumentation finden Sie wieder im Unterordner `doc` des von uns bereitgestellten Projekts sowie online unter `https://www.esa.informatik.tu-darmstadt.de/campus/mavl/`.

Wir stellen Ihnen unsere Implementierung des Parsers und der Kontextanalyse zur Verfügung. Verwenden Sie **nicht** Ihre Implementierungen, die Sie im Rahmen der ersten beiden Praktika entwickelt haben, um etwaige Folgefehler daraus zu vermeiden.

Achten Sie bei Ihrer Implementierung auch auf einen gut verständlichen und lesbaren Code-Stil und dokumentieren Sie Ihre Abgabe mit ausreichend Kommentaren! Es gelten weiterhin die Regeln zur Arbeit im Team vom ersten Aufgabenblatt. Bitte schreiben Sie Ihren Namen und Ihre Gruppennummer als Kommentar in *jede* abzugebende Datei.

Testen und Bewertung

Um Ihre Implementierung zu testen, stellen wir Ihnen eine Reihe von öffentlichen Testfällen bereit, die die Ausgabe der von Ihrem Compiler erzeugten Programme mit dem erwarteten Output vergleichen. Die erforderlichen Schritte zum Ausführen der Tests finden Sie in der Anleitung. Wenn Ihre Implementierung alle bereitgestellten öffentlichen Testfälle besteht, erhalten Sie **mindestens 40** der erreichbaren 80 Punkte.

Im Rahmen der Bewertung durch Ihre Tutoren werden wir Ihren Compiler weiteren nicht-öffentlichen Tests unterziehen, deren Ergebnis Ihre Punktzahl ergibt. **Daher ist es unabdingbar, dass Sie Ihre Implementierung mit eigenen MAVL-Beispielprogrammen gründlich testen!** Um die Ausgabe Ihres Compilers zu überprüfen, können Sie, wie in der Anleitung beschrieben, den erzeugten Binärcode mithilfe des Interpreters ausführen.

Abgabe

Nutzen Sie zur Vorbereitung des Abgabearchivs bitte unbedingt die in der Anleitung beschriebenen Kommandos. Beachten Sie, dass Ihre Abgabe **nur die Klasse `CodeGeneration`** umfasst. **Nehmen Sie keinesfalls Änderungen an anderen Klassen vor, da Sie damit riskieren, dass wir ihre Abgabe nicht ordnungsgemäß testen können. Darüber hinaus sollten Sie keinesfalls Funktionssignaturen ändern.** Sie können jedoch beliebig viele Hilfsmethoden definieren, sofern diese in den abzugebenden Klassen enthalten sind.

Aufgabe 3.1 Division *5 P*

Zu Beginn sollen Sie die Codegenerierung für die Division von 2 Werten (int oder float) in der Methode `visitDivision(...)` implementieren. Zum Erzeugen von TAM-Instruktionen sollen Sie die Funktionen der Klasse `TAMAssembler` nutzen!

Aufgabe 3.2 Variablenzuweisung und Adressermittlung *20 P*

Im Rahmen dieser Teilaufgabe sollen Sie Code für Variablenzuweisungen erzeugen. Dazu ist es notwendig, die Adresse der zugewiesenen Variablen zu ermitteln. Implementieren Sie dazu folgende Methoden:

- `visitLeftHandIdentifier(...)`
- `visitVectorLHSIdentifier(...)`
- `visitMatrixLHSIdentifier(...)`
- `visitRecordLHSIdentifier(...)`

Die Basisadresse relativ zur *Local Base* einer zuvor deklarierten Variablen lässt sich mithilfe der Methode `getLocalBaseOffset()` auf der zugehörigen `Declaration` feststellen. Implementieren Sie abschließend auch die Methode `visitVariableAssignment(...)`.

Aufgabe 3.3 Zugriff auf Record-Elemente *10 P*

Implementieren Sie in der Methode `visitRecordElementSelect(...)` die Codeerzeugung für den Zugriff auf Record-Elemente.

Aufgabe 3.4 Kontrollstrukturen

In den nächsten drei Aufgaben sollen Sie Code für MAVL-Kontrollkonstrukte erzeugen. Orientieren Sie sich dabei an den aus der Vorlesung bekannten Codeschablonen für bedingte Ausführung und Schleifen. Falls erforderlich stellt die Klasse `TAMAssembler` passende Methoden zum Erzeugen und Backpatching von Sprung-Instruktionen zur Verfügung.

Aufgabe 3.4.1 Switch *15 P*

Implementieren Sie die Codegenerierung für das Switch-Case-Konstrukt in MAVL, verteilt auf die Methoden

- `visitSwitchStatement(...)`,
- `visitCase(...)` und
- `visitDefault(...)`.

Aufgabe 3.4.2 Foreach-Schleife *20 P*

Erzeugen Sie passenden Code für die MAVL-Foreach-Schleife in der Methode `visitForEachLoop(...)`.
Wichtig: Rollen Sie die Schleife **nicht** während der Codeerzeugung ab. Abgaben, die den Schleifenkörper mehrfach und/oder keinen Rückwärtssprung enthalten, führen zu Punktabzug.

Aufgabe 3.4.3 Ternärer Operator *10 P*

In dieser Aufgabe sollen Sie die Codegenerierung für den ternären Operator (`?:`) in der Methode `visitSelectExpression(...)` implementieren.