Praktikum zu Einführung in den Compilerbau

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

TECHNISCHE UNIVERSITÄT DARMSTADT

Wintersemester 18/19
Praktikum 2

Abgabe bis Sonntag, 16.12.2018, 18:00 Uhr (MEZ)

Einleitung

Das Ziel dieses zweiten Praktikums ist die Entwicklung einer kontextuellen Analyse für einen MAVL-Compiler. Sie sollen dazu die Identifikationsphase und die Typprüfung gemäß der MAVL-Sprachspezifikation in einem kombinierten Visitor implementieren.

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.

In den folgenden Aufgaben werden Sie fehlende Methoden der Klassen `ContextualAnalysis` und `ConstantExpressionEvaluator` vervollständigen, die das Visitor-Interface der AST-Knoten-Klassen implementiert. Dabei nutzen Sie die Funktionalität der ebenfalls unvollständigen Klasse `IdentificationTable`. 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 bzw. AST-Aufbaus zur Verfügung. Verwenden Sie **nicht** den Parser, den Sie im Rahmen des ersten Praktikums 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 Ihre Gruppennummer, sowie die Namen und Matrikelnummern aller Gruppenmitglieder, 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 Ihres Compilers 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 D-AST in das Graphviz DOT-Format exportieren.

Abgabe

Nutzen Sie zur Vorbereitung des Abgabearchivs bitte unbedingt die in der Anleitung beschriebenen Kommandos. Beachten Sie, dass Ihre Abgabe **nur die Klassen `ContextualAnalysis`, `ConstantExpressionEvaluator` und `IdentificationTable`** 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.

Fehlermeldungen

Werfen Sie eine Instanz einer passenden Unterklasse von `CompilationError` im Package `mavlc.error_reporting`, um Fehler anzuzeigen, die Sie im Rahmen der Kontextanalyse ermittelt haben. Die Überprüfung der Art und des Inhalts dieser Fehlerobjekte ist Teil der Bewertung der nachfolgenden Aufgaben.

Aufgabe 2.1 Identifikationsphase *15 P*

Die Identifikationsphase, d.h. die Zuordnung von Verwendungen von Bezeichnern zu Deklarationen ist ein wichtiger Teil der kontextuellen Analyse. Bei dieser Zuordnung müssen die Geltungsbereiche beachtet werden, die in MAVL vor allem durch den Block-Befehl beeinflusst werden.

Machen Sie sich zuerst mit der vorgegebenen Klasse `Scope` vertraut. Anschließend ist es Ihre Aufgabe, die Klasse `IdentificationTable` zu vervollständigen, die Sie während der Kontextanalyse verwenden sollen. Vervollständigen Sie außerdem die Methode `visitCompoundStatement(...)` in der Klasse `ContextualAnalysis`.

Aufgabe 2.2 Variablenzuweisung *20 P*

Implementieren Sie die Prüfung der kontextuellen Einschränkungen für die Variablenzuweisung in folgenden Methoden in der Klasse `ContextualAnalysis`:

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

Aufgabe 2.3 Schleifen *15 P*

In dieser Aufgabe sollen Sie die kontextuelle Analyse für Schleifen implementieren. Vervollständigen Sie dazu die Methode `visitForEachLoop(...)` und `visitIteratorDeclaration(...)` in der Klasse `ContextualAnalysis`.

Aufgabe 2.4 Switch *10 P*

Nun ist es ihre Aufgabe, die Methoden `visitSwitchStatement(...)` und `visitSingleCase(...)` in der Klasse `ContextualAnalysis` zu implementieren, die die kontextuellen Einschränkungen für Switch-Case Statements überprüfen sollen.

Aufgabe 2.5 Ausdrücke *10 P*

In dieser Aufgabe sollen Sie die Typprüfung für ausgewählte Ausdrücke implementieren. Vervollständigen Sie dazu die Methoden `visitSelectExpression(...)` und `visitRecordInit(...)` in der Klasse `ContextualAnalysis`.

Aufgabe 2.6 Konstante Ausdrücke *10 P*

In dieser Aufgabe sollen Sie die Auswertung von (nicht-atomaren) konstanten Ausdrücken implementieren. Vervollständigen Sie dazu die Klasse `ConstantExpressionEvaluator`, indem Sie die fehlenden `visit...`-Methoden hinzufügen.