Einführung in den Compilerbau

Prof. Dr.-Ing. Andreas Koch
David Volz, Tim Noack, Jonas Renk

TECHNISCHE UNIVERSITÄT DARMSTADT

Praktikum
Wintersemester 23/24
Übungsblatt 2

Abgabe bis Sonntag, 17.12.2023, 18:00 Uhr (MEZ)

Einleitung

Das Ziel dieses zweiten Praktikums ist die Entwicklung einer kontextuellen Analyse für den 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. Weitere Informationen zum Framework finden Sie in unseren Anleitungsvideos[1].

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.

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[2], erhalten Sie **mindestens 40** der erreichbaren 80 Punkte.

Im Rahmen der Bewertung durch Ihre Tutoren werden wir Ihren Compiler weiteren nicht-öffentlichen Tests unterziehen,

[1] https://www.youtube.com/playlist?list=PLz70m4N3m7xvSCMk_FhWOTl8EKklgLkrb
[2] Falls Sie die Testfälle in Ihrer Entwicklungsumgebung ausführen, stellen Sie bitte sicher, dass Ihre Abgabe auch mit `gradle test` funktioniert.

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 Wichtig!

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`** aus dem Package `mavlc.context_analysis` 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.

Bitte beachten Sie bei der Abgabe unbedingt Folgendes:

- **Benutzen Sie Gradle zur Abgabe**: Verwenden Sie bitte `./gradlew prepareSubmission`, um ein gültiges Abgabearchiv zu erzeugen. Manuell-erzeugte Abgaben mit einer anderen Ordnerstruktur werden wir nicht akzeptieren. Genauere Hinweise finden Sie unter `HOWTO.md` oder in unserem Tutorial.
- **Entfernen Sie unnötige Imports**: Wir werden Ihre Abgabe mit Java 11 testen. Wenn Ihre Abgabe unter Java 11 nicht kompiliert werden kann, wird sie mit 0 Punkten bewertet. Versehentlich hinzugefügte Imports sind hierbei ein häufiges Problem, bitte achten Sie darauf.

Andernfalls werden wir Ihre Abgabe mit **0 Punkten** bewerten.

Fehlermeldungen

Werfen Sie eine Instanz einer passenden Unterklasse von `CompilationError` im Package `mavlc.errors`, 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.

Zugehörige Vorlesungen und Folien

Um dieses Praktikum gut bearbeiten zu können, sollten Sie die Folien bis einschließlich **Kapitel 3, Folie 92** bearbeitet haben.

Aufgabe 2.1: Identifikationsphase (12 Punkte)

Die Identifikationsphase, bei der Bezeichner deklariert und später zu ihrer Deklaration zugeordnet werden, ist ein wichtiger Teil der kontextuellen Analyse. Bei dieser Zuordnung müssen die Geltungsbereiche beachtet werden, die in MAVL vor allem – aber nicht ausschließlich – durch Befehlsblöcke 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 (24 Punkte)

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

- `visitVariableAssignment(...)`
- `visitLeftHandIdentifier(...)`
- `visitVectorLhsIdentifier(...)`
- `visitMatrixLhsIdentifier(...)`
- `visitRecordLhsIdentifier(...)`

Hinweis: Der öffentliche `assign_2_ctx.mavl` Testfall erfordert, dass sie auch `visitRecordInit` aus Aufgabe 2.4 implementieren.

Aufgabe 2.3: Konstante Ausdrücke (8 Punkte)

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.

Hinweis: Das Verhalten für konstante Ausdrücke, die (beispielsweise durch Overflows) nicht zu einem `int` im Sinne der MAVL-Spezifikation auswerten, ist Ihnen überlassen. Wir werden diese Fälle nicht testen.

Aufgabe 2.4: Ausdrücke (12 Punkte)

Implementieren Sie die Typprüfung für ausgewählte Ausdrücke, in dem Sie die folgende Methoden in der Klasse `ContextualAnalysis` implementieren:

- `visitRecordInit(...)`
- `visitMatrixMultiplication(...)`
- `visitMatrixTranspose(...)`
- `visitCompare(...)`

Hinweis: Für die Analyse von `RecordInit` kann die Klasse `ModuleEnvironment` hilfreich sein, eine passende Instanz steht in der Visitorklasse durch das Feld `env` bereit.

Aufgabe 2.5: Schleifen (12 Punkte)

In dieser Aufgabe sollen Sie die kontextuelle Analyse für `for`-Schleifen implementieren. Füllen Sie dafür die Methode `visitForLoop(...)` in der Klasse `ContextualAnalysis`.

Aufgabe 2.6: Funktionsaufrufe (12 Punkte)

Erweitern Sie kontextuelle Analyse, sodass Funktionsaufrufe analysiert werden können. Erweitern Sie dafür die Methoden `visitCallExpression(...)` und `visitCallStatement(...)` in der Klasse `ContextualAnalysis`.