WS 2018/2019 08.11.2018

Praktische Übungen zur Vorlesung
Programmanalyse
Blatt 3

Prof. Dr. Roland Meyer,
M. Sc. Sebastian Wolff,
M. Sc. Peter Chini Abgabe bis 05.12.2018 um 12 Uhr

Aufgabe 3.1 (Vorbereitungen)
Um die Programmieraufgaben auf diesem Übungsblatt bearbeiten zu können, müssen Sie ihr System entsprechend vorbereiten. Wir empfehlen ein Unix System.

1. Installieren Sie den SMT-Solver *Microsoft Z3* in der Version 4.8.1.[1]
2. Installieren Sie den Parser-Generator *AntLR* in der Version 4.1.7.[2]
3. Laden Sie sich unser Framework herunter.[3]

Hinweis: Das Zip-Archiv unseres Frameworks enthält genauere Informationen zur Konfiguration Ihres Systems. Alternativ stellen wir ein Virtual-Machine-Image bereit, das Sie zum Lösen der Aufgaben verwenden können.[4]

Aufgabe 3.2 (Abstract Syntax Tree, Visitors)
Erstellen Sie einen Z3 `Context` für ein gegebenes Programm in Form eines AST. Gehen Sie dazu wie folgt vor.

1. Machen Sie sich mit dem Abstract Syntax Tree (AST) im Paket `paparser.ast` vertraut, welcher While-Programme repräsentiert.
2. Machen Sie sich mit dem Visitor Pattern vertraut.[5] Dieses Entwurfsmuster ist (sowohl für dieses Übungsblatt als auch im Allgemeinen) hilfreich, um Problem mit statischem/dynamischem Binden zu lösen.
3. Implementieren Sie einen Visitor, der die vorhandenen Variablen in einem Programm ausfindig macht. Vervollständigen Sie dazu die Datei `ContextMaker.java`.
4. Erstellen Sie ein Beispielprogramm und testen Sie ihre Implementierung. Die Klasse `paparser.Parser` kann While-Programme aus Dateien einlesen und liefert den entsprechenden AST.

[1] `https://github.com/Z3Prover/z3`
[2] `https://www.antlr.org`
[3] `https://www.tcs.cs.tu-bs.de/documents/ProgAnalyse_WS_1819/Programmanalyse.zip`
[4] `ttps://www.tcs.cs.tu-bs.de/documents/ProgrammanalyseVM1819.ova`
[5] `https://en.wikipedia.org/wiki/Visitor_pattern`

Bemerkung: Das Sammeln der Programmvariablen ist von Nöten, da Z3, bevor es SAT/SMT Anfragen lösen kann, mit den (potentiell) vorhanden Variablen initialisiert werden muss. Die Klasse `paparser.ContextHelper` hilf Ihnen dabei.

Aufgabe 3.3 (Invarianten)
Prüfen Sie, ob ein gegebenes Programm Invarianten enthält, welche Tautologien oder unerfüllbar sind. Dazu:

1. Machen Sie sich mit Z3[6] und der entsprechenden Java API[7] vertraut.
2. Implementieren Sie einen geeigneten Visitor, der alle Invarianten prüft.
3. Erstellen Sie ein Beispielprogramm und testen Sie ihre Implementierung.

Bemerkung: Aus technischen Gründen sind Invarianten im AST im STMlib Format gespeichert.[8] Die Klasse `paparser.ContextHelper` kann diese in Z3 Repräsentation umformen.

Hinweis: Mit Hilfe der von Z3 zur Verfügung gestellten Klasse `Solver` können Sie Formeln auf (Un-)Erfüllbarkeit prüfen. Wenn Sie allerdings eine solche Anfrage gestellt haben, müssen Sie den Solver *zurücksetzten*. Machen Sie sich dazu mit den Methoden `push` und `pop` der Klasse `Solver` vertraut.

Aufgabe 3.4 (Verification Conditions)
Generieren und prüfen Sie die Verification Conditions eines gegeben Programms. Gehen Sie wie folgt vor.

1. Implementieren Sie weakest preconditions für Z3 Formeln, d.h. implementieren Sie die Funktion *pred* aus der Vorlesung.

   Hinweis: Formeln in Z3 werden als Objekte der Klasse `Expr` gespeichert. Diese bietet die Methode `substitute` an, welche Teilausdrücke ersetzen kann.
2. Implementieren Sie einen geeigneten Visitor, welcher die Funktion *vc* aus der Vorlesung berechnet.
3. Prüfen Sie die Validität der zuvor berechneten Verification Conditions.
4. Erstellen Sie Beispielprogramme und testen Sie ihre Implementierung.

Bei Fragen zu oder Problem mit den Aufgaben, wenden Sie sich bitte an uns!

Abgabe bis 05.12.2018 um 12 Uhr als `zip` Archiv per EMail an Peter Chini.

[6] `https://github.com/Z3Prover/z3`
[7] `https://z3prover.github.io/api/html/namespacecom_1_1microsoft_1_1z3.html`
[8] `http://smtlib.cs.uiowa.edu`