Karlsruher Institut für Technologie
Institut für Theoretische Informatik
Prof. Dr. Bernhard Beckert
Jonas Schiffl

Formale Systeme, WS 2019/2020

Praxisaufgabe 2: Modellierung einer Blockchain in SMT-LIB

Abgabe der Lösungen bis zum 31.1.2020 über das ILIAS-Portal zur Vorlesung

Hinweis: Bitte beachten Sie, dass die Praxisaufgabe in Einzelarbeit und selbständig zu bearbeiten ist. Wir behalten uns vor, die selbständige Bearbeitung dadurch stichprobenartig zu prüfen, dass wir uns die eingereichte Lösung erklären lassen.
Für die vollständige Lösung dieser Praxisaufgabe erhalten Sie 10 Punkte. Die erzielten Übungspunkte werden im Verhältnis 1:10 als Bonuspunkte auf die bestandene Abschlussklausur angerechnet (max. ein Notenschritt Verbesserung).

1 Aufgabe

Eine Blockchain ist eine kryptographische Datenstruktur, die es ermöglicht, in einem dezentralen Rechnernetzwerk einen einheitlichen Zustand herzustellen und gleichzeitig Manipulationen der gespeicherten Historie zu verhindern.

In dieser Aufgabe geht es darum, eine informelle Beschreibung einer Blockchain als SMT-Problem zu formalisieren. Schreiben Sie Ihre Formalisierung im SMT-LIB-2-Format[1] auf, um sie mit einem SMT-Solver überprüfen zu können.

a) Erstellen Sie die SMT-Sorten `Account`, `Block` und `Transaction`. Nutzen Sie dazu das Kommando `declare-sort`. Überlegen Sie nun, welche Prädikate und Funktionen benötigt werden, um die informelle Beschreibung (1.1) zu formalisieren. Fügen Sie diese Prädikate und Funktionen mit den entsprechenden Signaturen mittels `declare-fun` Ihrer Modellierung hinzu.

b) Übersetzen Sie die informelle Beschreibung (1.1) in Axiome Ihrer Modellierung. Nutzen Sie dazu das Kommendo `assert`.

c) Zeigen Sie, dass Ihre Modellierung die Eigenschaften aus Abschnitt 1.2 erfüllt. Übersetzen Sie dazu die Beweisverpflichtungen nach SMT-LIB und zeigen Sie, dass deren Negation in Ihrer Modellierung unerfüllbar ist.

1.1 Informelle Beschreibung

- Es gibt einen initialen Block. Jeder Block hat einen Vorgänger. Der initiale Block ist der einzige, der sein eigener Vorgänger ist.
- Kein Block hat mehr als einen Nachfolger.
- Ein Konto hat einen Kontostand, der sich von Block zu Block verändern kann. Dieser Kontostand kann nicht negativ sein.
- Transaktionen haben einen Sender, einen Empfänger und einen Betrag. Sie gehören zu einem Block. Der zu überweisende Betrag ist positiv und nicht größer als der Kontostand des Senders.

[1] http://smtlib.cs.uiowa.edu/papers/smt-lib-reference-v2.6-r2017-07-18.pdf

- Wird in einer Transaktion in einem Block Geld von einem Konto an ein anderes überwiesen, dann werden die beiden Kontostände im nachfolgenden Block dementsprechend angepasst.
- Jedes Konto kommt pro Block nur einmal vor, d.h. es ist z.B. nicht möglich, dass in einem Block mehrere Transaktionen von einem Konto ausgehen oder dass ein Konto innerhalb eines Blocks gleichzeitig Geld empfängt und versendet.

1.2 Beweisverpflichtungen

BV1: Hat ein Konto `acc1` in einem Block `b1` den Kontostand 42 und überweist in diesem Block den Betrag 19 an ein anderes Konto, so hat `acc1` im nächsten Block den Kontostand 23.

BV2: Eine Transaktion erhält die Summe der Kontostände der beiden beteiligten Konten.

2 Abgabe

Erstellen Sie für die Abgabe drei Dateien im SMT-LIB-2-Format: `bc-model.smt2` für die Modellierung (Aufgabenteile *a)* und *b)*) sowie `bv1.smt2` und `bv2.smt2` jeweils für eine der beiden Beweisverpflichtungen (Aufgabenteil *c)*). Reichen Sie die drei Dateien bis zum Freitag, den 31.1.2019 23:59 Uhr im ILIAS-Portal unter "Praxisblatt 2" ein. Beachten Sie das untenstehende Beispiel und die folgenden Hinweise:

- Stellen Sie sicher, dass Ihre Abgabe syntaktisch korrekt ist. Überprüfen Sie dazu, ob sich die Dateien `bc-model.smt2` sowie die Konkatenationen `bc-model.smt2`:`bv1.smt2` und `bc-model.smt2`:`bv2.smt2` ohne Fehler von einem SMT-Solver (z.B. z3[2] oder CVC4[3]) aufrufen lassen.
- Plausibilitätsprüfung: Ihre Modellierung sollte erfüllbar sein! Überprüfen Sie dies, indem Sie am Ende der Datei `bc-model.smt2` das Kommando `(check-sat)` aufrufen.
- Zeigen Sie für die beiden Beweisverpflichtungen, dass die Negation der gewünschten Eigenschaften in Ihrem Modell unerfüllbar ist. Nutzen Sie dazu wiederum `(check-sat)`.

Beispiel zur Abgabe

- `bc-model.smt2`:

```
(declare-sort Thing 0)
(declare-fun five (Thing Thing) Int)
(assert (forall ((t1 Thing) (t2 Thing))
        (= (five t1 t2) 5)))
(check-sat)
```

- `bv1.smt2`:

```
(declare-const t1 Thing)
(declare-const t2 Thing)
(assert (= (five t1 t2) 6))
(check-sat)
```

Bash-Aufrufe für dieses Beispiel könnten folgendermaßen aussehen:

[2] https://github.com/Z3Prover/z3
[3] https://cvc4.github.io/

```
$ z3 model.smt2
sat
$ cat model.smt2 bv.smt2 | z3 -in
sat
unsat
```

Somit wäre gezeigt, dass die Modellierung an sich erfüllbar ist und dass die Funktion `five` auf zwei Instanzen der Sorte `Thing` nie 6 zurückgibt.