Lehrstuhl für Software and Computational Systems
Prof. Dr. Gidon Ernst, Daniel Baier, Marian Lingsch

Projekt zur Vorlesung
Formale Spezifikation und Verifikation

Wintersemester 2023

Projekt 02

Bekanntgabe am 7.12.2023, Abgabe am 22.12.2023

1 Informationen und Installation

In diesem Projekt experimentieren Sie mit der Formulierung von logischen Aussagen um dann Lösungen bzw. Unerfüllbarkeit mit Hilfen von SAT/SMT Solvern zu bestimmen.

Hintergrundinformationen finden Sie u.a. auf `http://smtlib.cs.uiowa.edu`, und im SMT-LIB Standard `http://smtlib.cs.uiowa.edu/papers/smt-lib-reference-v2.6-r2017-07-18.pdf`.

Das Projekt setzt eine Einarbeitung in die von SMT-Solvern genutzte Syntax voraus. Ein Tutorial finden Sie z.B. hier: `https://smtlib.github.io/jSMTLIB/SMTLIBTutorial.pdf`.

Bitte verwenden Sie eine halbwegs aktuelle Version von Z3 (z.B. $\geq$4.8.7), andere Solver können evtl die letzten beiden Aufgaben nicht lösen. Der Solver läuft auf allen Plattformen, die meisten Linux Distributionen bieten ein Paket an. Sie können das Programm aber auch direkt herunterladen `https://github.com/Z3Prover/z3/releases`.

Viele Editoren bieten Syntax-Highlighting an. Wir empfehlen Visual Studio Code, das Sie bereits aus dem Dafny Projekt kennen.

- Download: `https://code.visualstudio.com/Download`, alternativ als Paket in vielen Linux Distsributionen
- Das Plugin für schöne Syntax wird evtl automatisch angeboten, falls Sie eine Datei mit der Endung `.smt2` öffnen. Sie können es aber auch manuell herunterladen und installieren: `https://marketplace.visualstudio.com/items?itemName=martinring.smt-lib-syntax`

Hinweise:

- Lösen Sie jede Teilaufgabe in einer eigenen Datei. Jede Datei sollte genau **ein** `(check-sat)` Kommando am Schluss enthalten, möglicherweise gefolgt von `(get-model)`.
- Die Aufgaben erfordern Hintergrundwissen aus unterschiedlichen Teilen der Vorlesung und Transfer dieser Ideen auf konkrete Problemstellungen.

Alternativ können Sie das Projekt auch mit Hilfe der Z3-API in Python lösen, welches in diesem Paket verfürbar ist `https://pypi.org/project/z3-solver/`. *Bitte geben Sie dann trotzdem SMT-LIB Dateien ab!* Sie können sich mit `print(x.sexpr())` in Python alle Z3 Objekte, also auch ein Solver Objekt, im SMT-LIB format ausgeben lassen.

2 Beispiele

Erstellen Sie eine Datei `prop.smt2` mit folgendem Inhalt:

```
(set-logic ALL)
(declare-const A Bool)
(declare-const B Bool)
(assert  (not (= (and (not A) (not B))
                 (not (or A B)))))
(check-sat)
```

Auf der Kommandozeile bzw im Terminal von Visual Studio Code können Sie jetzt prüfen, ob eine Belegung für `A` und `B` existiert, welche die Formel wahr macht: `z3 prop.smt2`

In jedem Fall sollte hierbei `unsat` herauskommen—überlegen Sie sich, warum das der Fall ist. Sie können auch versuchsweise das äußere `(not ...)` entfernen, dann kommt `sat` heraus.

Betrachten Sie das unten gezeige Dafny Programm `abs.dfy`. Wir möchten herausfinden, ob die `assert` Anweisung fehlschlagen kann. Dazu führen wir von Hand die symbolische Suche durch und bestimmen die Formel φ, die den Zustand in der letzten Zeile genau beschreibt, und wir codieren die Bedingung der Zusicherung ebenfalls als eine Formel ψ, also:

$\varphi \equiv (x < 0 \wedge y = -x) \vee (x \geq 0 \wedge y = x)$ sowie $\psi \equiv y \geq 0$

Das Programm hat genau dann einen Fehler, wenn es einen Zustand s gibt, sodass $s \models \varphi$ aber nicht $s \models \psi$. Wir können diese Frage als ein Erfüllbarkeitsproblem ausdrücken:

Gibt es s sodass $s \models \varphi \wedge \neg\psi$? Genau diese Frage beantwortet ein SMT-Solver.

Programm `abs.dfy`

```
method abs()
{
  var x: int;
  var y: int;
  if x < 0 { y := -x; }
  else     { y :=  x;  }
  assert y >= 0;
}
```

Datei `abs.smt2`

```
(set-logic ALL)
(set-option :produce-models true)
(declare-const x Int)
(declare-const y Int)
(assert (or (and (<  x 0) (= y (- x)))
            (and (>= x 0) (= y    x))))
(assert (not (>= y 0)))
(check-sat)
(get-model)
```

Ein Aufruf von `z3 abs.smt2` ergibt, dass es keinen solchen Fehlerzustand gibt, durch die Ausgabe `unsat`, und der Meldung dass es keine erfüllende Belegung s verfügbar ist (no "model").

Fügen Sie Beispielhaft einen Fehler in das Programm ein, passen Sie die `.smt2` Datei an und prüfen Sie erneut. Beispielsweise können Sie `(- x)` durch `(+ x)` in `abs.smt2` ersetzen, dann erhalten Sie ein konkretes Gegenbeispiel, Z3 Version 4.8.7 wählt beispielsweise $x = -1$, und damit ebenfalls $y = -1$. Zum Nachdenken: ist dies das einzig mögliche Modell?

```
sat
(model
  (define-fun y () Int  (- 1))
  (define-fun x () Int  (- 1)))
```

3 Einfache Erfüllbarkeit (1 Punkt)

Überprüfen Sie welche der folgenden Formeln **erfüllbar** sind welche **allgemeingültig** sind: Codieren Sie dazu für jede Teilaufgabe und beide dieser Fragen **eine eigenene SMT-LIB Eingabedatei** mit einem finalen `(check-sat)` Befehl. Lassen Sie sich für *erfüllbare* Formeln ein Modell anzeigen mit `(get-model)`. Den selben Befehl sollten Sie verwenden, um für *nicht allgemeingültige* Formeln ein Gegenbeispiel anzuzeigen.

- `3a-satisfiable.smt2` und `3a-tautology.smt2`

$$A \wedge (B \vee C) \iff (A \wedge B) \vee (A \wedge C)$$

- `3b-satisfiable.smt2` und `3b-tautology.smt2`

$$(A \wedge B) \Longrightarrow \Big((\neg (C \Longrightarrow A)) \vee (B \Longrightarrow A) \Big)$$

Hinweise:

- Lösen Sie jede Teilaufgabe in einer separaten Datei mit dem angegeben Namen.
- Denken Sie daran, alle verwendeten atomaren Propositionen zu deklarieren.
- Implikation schreibt sich als `=>`, Äquivalentz $\iff$ wird, ebenfalls wie Gleichheit $=$, in SMT-LIB beide mit dem Symbol `=` codiert.

4 Logikrätsel (4 Punkte)

In der Vorlesung haben Sie bereits die Insel der Ritter und Schurken kennen gelernt. Auf dieser Insel sprechen Ritter immer die Wahrheit, während Schurken immer lügen. Jeder/e ist entweder Ritter oder Schurke, aber man sieht es ihnen nicht an.

Lösen Sie die folgenden Rätsel mit Hilfe von Z3, indem Sie das Problem in der angegebenen Datei codieren, gefolgt von `(check-sat)` und `(get-model)`. Kommentieren Sie die Antworten in den Abgabedateien! Zeilenkommentare starten mit Semikolon: `; comment`

- `4a.smt2` Während eines Besuches auf der Insel treffen Sie auf Alex und Chris.

  Alex sagt: *Wir sind beide Ritter.*

  Chris sagt: *Wenn ich Ritter bin dann spreche ich die Wahrheit.*

  Was sind die beiden? Ist die Lösung eindeutig?

- `4b.smt2` Später treffen Sie auf Evelyn und erhalten zusätzlich diese Information:

  Evelyn sagt: *Wenn ich Ritter bin, dann ist Alex Schurke.*

  Welche Möglichkeiten gibt es jetzt?

- `4c.smt2` Evelyn überlegt kurz und sagt dann: *Ich bin kein Ritter.*

  Welche Möglichkeiten gibt es jetzt? Interpretieren Sie das Ergebnis im Hinblick auf die Beschreibung der Insel.

Hinweise:

- Bei der ersten Teilaufgabe können Sie sich am Vorgehen in der Vorlesung orientieren.
- Wenn Sie eine Lösung wie z.B. $A \mapsto$ `true` und $C \mapsto$ `false` gefunden haben, dann können Sie diese z.B. $\neg(A \wedge \neg C)$ ausschließen um eine weitere Lösung zu finden.
- Überlegen Sie, ob Ihre Lösung intuitiv Sinn macht, falls nicht, prüfen Sie die Formalisierung.

5 Spezifikation von Eigenschaften (4 Punkte)

In dieser Aufgabe geht es um Arrays und Quantoren, sowie der Definition von Funktionen.

- `5.smt2` **Definieren Sie eine Funktion `isPalindrome`** die überprüft, ob ein Array mit ganzzahligen Werten in einem bestimmten Bereich ein Palindrom ist.

Hinweise:

- Sie haben bereits eine Datei als Vorlage erhalten. Ersetzen Sie die Zeile `true ; TODO` durch Ihre Definition.
- Beachten Sie dabei die Namen der Parameter: `a` ist das Array und `l` und `r` sind die Grenzen, wobei z.B. `l` *inklusive* und `r` *exklusive* ist.
- Quantoren $\forall$ und $\exists$ können Sie mit `forall` und `exists` spezifizieren, z.B. `(exists ((i Int) (j Int)) (<= i j))` (diese Formel ist nicht Teil der Lösung).
- Nachschlagen in einem Array $a[i]$ schreiben Sie als `(select a i)`.
- Achten Sie sehr genau auf eine korrekte Klammerung! Ein Editor mit Hervorhebung von Klammern hilft sehr (z.B. VS Code).

Der Rest der Datei enthält Tests, die prüfen, ob Ihre Lösung korrekt ist. Wenn Ihre Lösung diese Tests erfüllt, dann sehen Sie ein paar `sat` und `unsat` Ausgaben. Bei einer falschen Lösung kommt zusätzlich sowas wie:

Z3: `(error “line X column Y: check annotation that says unsat”)`

Sie können dann z.B. ein `(get-model)` Kommando anfügen, um sich das Gegenbeispiel anzeigen zu lassen, hierbei ist `((as const (Array Int Int)) x)` ein Array, dass an allen Indizes die Zahl `x` enthält.