Karlsruher Institut für Technologie
Institut für Theoretische Informatik
Prof. Dr. Bernhard Beckert
Alexander Weigl, Mihai Herda

Formale Systeme, WS 2018/2019

Praxisaufgabe 2: Planungsproblem lösen mit SMT-Solver

Abgabe der Lösungen bis zum 10.02.2019 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 Verhaältnis 1:10 als Bonuspunkte auf die bestandene Abschlussklausur angerechnet (max. ein Notenschritt Verbesserung).

Aufgabenstellung

Gegeben sei das unten definierte Problem zur Planung von Aufträgen. Jeder Auftrag besitzt eine Durchführungsdauer, einen Fälligkeitstermin und eine zugeteilte Maschine. Ein Auftrag kann von anderen Aufträgen abhängen, sodass der abhängige Auftrag erst gestartet werden kann, nachdem alle Vorgänger abgeschlossen wurden. Eine parallele Abarbeitung der Aufträge ist möglich, solange niemals zwei Aufträge auf der gleichen Maschine laufen. Eine Lösung für das Problem ist eine Zuweisung von Startzeitpunkt zu jedem Auftrag, sodass es keine Arbeitspausen existieren und die obigen Einschränkungen eingehalten werden. Aufträge können nicht unterbrochen werden. Aufträge werden auf ganze Tage verteilt (und die Zeit somit als natürliche Zahl modelliert). Wenn ein Auftrag am $i$ten Tag endet, können Nachfolger am $i + 1$ten Tag starten.

Ihre Aufgabe ist es:

1. das Problems in SMTlib zu formalisieren, d.h. definieren Sie eine Signatur und formulieren Sie die Ausgabeeinschränkungen in sortierte PK1; beides im SMT-LIB. Reichen Sie Ihre Lösung als SMTlib-Datei ein, verwenden Sie bitte die Vorlage `jobscheduling.smt2`.

2. ein Programm zu schreiben, das Instanzen des Plannungsproblem einliest und in SMTlib übersetzt. Die Ausgabe wird anschließend vom SMT-Solver `z3` gelöst.

   > **Hinweis:** Starten Sie mit Ihrer Problemformalisierung. Diese müssen Sie nun um Eingaben aus der gegebenen Probleminstanz von Ihrem Programm ergänzt werden.

   Optimieren Sie! Die reine Problemformalisierung skaliert nicht in Anzahl der Aufträge: Experimentieren Sie mit den Funktionalitäten und SMT-Solver[1] Verwenden Sie zur Implementierung entweder Python (>=3.5) oder Java 11. Die technische Spezifikation und das Rahmenwerk folgen weiter unten.

Definition 1 (Auftragsplanung)**.**

Eingabe *Eine Menge* $J = \{j_1, ..., j_n\}$ *von Aufträgen wobei*

- $t_i \in \mathbb{N}_+$ *die Dauer,*
- $m_i \in \mathbb{N}$ *die zugewiesene Maschine,*
- $d_i \in \mathbb{N}$ *der Fälligkeitstermin (deadline)*

[1] Ideen: Quantoren ausrollen, Bitvektor-Arithmetik, Aufzählungsdatentyp

des i-ten Auftrags ist.

Zudem sei eine Halbordnung $\prec: J \times J$ gegeben, sodass $i \prec j$ gdw. Auftrag i vor den Auftrag j fertig gestellt sein muss.

Ausgabe *Startdatum b_i für jeden Auftrag, sodass*

- *das Fälligkeitsdatum jedes Auftrages $b_i + t_i \leq d_i$ eingehalten wird,*
- *zu jedem Zeitpunkt maximal nur ein Auftrag auf jeder Maschine läuft.*
- *kein Auftrag gestartet wird, bevor alle seine Vorgänger fertig gestellt wurden, und*
- *es keine Arbeitspausen gibt, d.h. zu jedem Zeitpunkt existiert ein Auftrag, der abgearbeitet wird (oder alle Aufträge sind erfüllt).*

Technische Spezifikation und Rahmenwerk

Im Rahmenwerk, verfügbar auf der Verlesungswebseite, finden Sie für die erlaubten Programmiersprachen bereits vordefinierte Templates, die das Eingabeformat einlesen und als Datenstruktur bereitstellen. Ihr Programm soll eine Instanz des Problems als Text von der Standardeingabe einlesen und auf der Standardausgabe eine Ausgabe in SMT-LIB ausgeben. Testen Sie Ihre Ausgabe gegen den SMT-Solver z3[2]. Ihr Programm wird wie folgt aufgerufen:

```
$ cat examples/aufgabenblatt.csv \
    | python3 jobscheduling.py
    | tee latest.smt2 \
    | z3 -in -smt2
```

Das Rahmenwerk bietet `run.sh`, welches die verwendete Programmiersprache detektiert, auf Bedarf kompiliert und Ihr Programm sowie z3 aufruft. Als ersten Parameter erwartet `run.sh` den Pfad zur CSV-Datei.

```
$ ./run.sh examples/aufgabenblatt.csv
```

Löschen Sie die Vorlage der Programmiersprachen, welches Sie nicht verwenden!

Hinweis zur Abgabe: Ihre Abgabe besteht aus dem Quelltext Ihrer Implementierung. Externe Bibliotheken sind untersagt. Verwendung der Standardbibliotheken ist erlaubt[3].

Eingabeformat Die Eingabe erfolgt in einem CSV-basierten Format. Jede Zeile ist ein Auftrag mit komma-separierten Werten für die Dauer, das Fälligkeitdatum, die zugewiesene Maschine. Alle weiteren Werte referenzieren die Aufträge (Zeilennummer), die vor dem aktuellen Auftrag ausgeführt werden müssen. Hier ein Beispiel mit fünf Aufträgen:

```
1,3,1
1,3,1
1,3,1,2
4,6,2,1
5,13,3,2,3
```

Der fünfte Auftrag (`5,13,3,2,3`) dauert 5 Schritte, muss spätestens am 13. Schritt fertigstellt werden, läuft auf Maschine 3 und hängt vom dem zweiten und dritten Auftrag ab. Eine gültige Lösung wäre:

$b_1 = 0$ $b_2 = 1$ $b_3 = 2$ $b_4 = 1$ $b_5 = 3$

[2]Eine online Version ist unter https://rise4fun.com/z3/ verfügbar.

[3]Hinweis: JavaFX ist in Java 11 nicht mehr enthalten, die häufig verwendete Klasse `javafx.util.Pair` steht nicht zur Verfügung.

Ausgabe

Die Ausgabe Ihres erfolgt in SMTlib. Ein interaktives Tutorial finden Sie auf den Seiten zu z3: https://rise4fun.com/z3/tutorial. Die vollständige Referenz gibt es hier: http://smtlib.cs.uiowa.edu/papers/smt-lib-reference-v2.6-r2017-07-18.pdf.

Hinweise zum Vorgehen: Fangen Sie mit der Formalisierung der Problembeschreibung an. Danach sollten Sie einzelne kleine Instanzen von Hand in SMT-LIB schreiben bevor Sie mit der vollständigen Implementierung anfangen.