Karlsruher Institut für Technologie
Institut für Theoretische Informatik
Prof. Dr. B. Beckert
Dr. des. Vladimir Klebanov

Formale Systeme, WS 2009/2010

Praxisaufgabe 1: SAT-o-ban
Lösen eines Planungsproblems durch SAT-Solving

Abgabe am 3.1.2010.

1 Allgemeines

Aufgabe. Das Ziel dieser Praxisaufgabe ist, einen automatischen Spieler für das populäre Spiel Sokoban zu entwickeln. Dafür sollen Sie das durch das Spiel gegebene Planungsproblem in ein Erfüllbarkeitsproblem der Aussagenlogik übersetzen. Ein handelsüblicher SAT-Solver sucht die Lösung des Erfüllbarkeitsproblems, die dann in die Sprache des Spiels zurücktransformiert werden soll.

Sokoban. Sokoban ist ein einfaches und populäres Spiel. Das Spiel besteht aus Levels, die jeweils einen kleinen Lagerraum darstellen. Im Lagerraum stehen eine Reihe von Containern. Der von Ihnen gesteuerte Held des Spiels muß die Container auf die vorgegebenen Zielplätze schieben. Dabei kann er gleichzeitig nur einen Container schieben. Die Regeln sind nochmals vollständig unter `www.sokobano.de` aufgeführt.

Sokoban ist ein typisches Planungsproblem: Gegeben ist eine Weltformalisierung (die sog. Planungsdomäne), der initiale Zustand, sowie eine (partielle) Beschreibung des gewünschten Endzustandes. Der Planer muß eine Abfolge aus vordefinierten Aktionen finden, die die Welt aus dem Anfangszustand in einen die Anforderungen erfüllenden Endzustand überführt.

Sokoban ist NP-hart und PSPACE-vollständig.[1]

Präzisierung der Aufgabe. Schreiben Sie ein Java-Programm, das einen als Eingabe dienenden Sokobanlevel samt einer natürlichen Zahl N in eine aussagenlogische Formel übersetzt. Die Formel soll genau dann erfüllbar sein, wenn der gegebene Level eine Lösung aus N Aktionen (Spielerbewegungen) hat. In diesem Fall, sollte Ihr Programm die Levellösung aus der erfüllenden Belegung der Formel ablesen.

2 Technischer Rahmen

Das Rahmenwerk. Auf der Vorlesungswebseite finden Sie das Archiv `satobanSrc.zip` mit dem Rahmenwerk. Das Rahmenwerk basiert auf der freien Java-Implementierung MazezaM[2].

Sie brauchen, nur die Klasse `sokoban.Solver` zu verändern. Bitte ändern Sie die anderen Klassen nicht ohne Rücksprache. Übersetzen können Sie das Programm mit dem Build-Tool *ant*. Geben Sie dafür im Toplevel-Verzeichnis `ant jar` ein. Danach können Sie das Programm mit `java -jar satoban.jar` starten (bzw. mit einem Doppelklick auf die Jar-Datei, falls Ihr System das unterstützt).

[1] `http://web.cs.ualberta.ca/~joe/Preprints/Sokoban/`
[2] `http://webpages.dcu.ie/~tyrrelma/MazezaM/`

Datenfluß. Beim Starten liest SAT-o-ban die Levels aus der Datei `levels.txt` im aktuellen Verzeichnis. Für jeden gefundenen Level passiert jeweils folgendes:

- SAT-o-ban übersetzt den Level in eine aussagenlogische Formel (diesen Teil müssen Sie implementieren) und speichert diese in der Datei `in.ms`.
- SAT-o-ban ruft den SAT-Solver mit dieser Datei als Eingabe auf.
- Der SAT-Solver schreibt die Ausgabe in die Datei `out.ms`.
- SAT-o-ban liest und interpretiert die Ausgabe des SAT-Solvers, sowie gibt das interpretierte Ergebnis aus auf der Konsole.

Debug-Modus. Das Rahmenwerk hat einen Debug-Modus, der viele hilfreiche Zusatzinformationen ausgibt (u.a. eine lesbare Interpretation der SAT-Solver-Ausgabe). Im Debug-Modus kann außerdem eine ggf. gefundene Lösung durch wiederholtes Drücken der Taste `n` im aktuellen Spielfeld animiert werden.

Sie schalten den Debug-Modus ein und aus durch die boolesche Variable `DEBUG` in der Klasse `Solver`. Danach müssen Sie das *ganze* Programm neu übersetzen: Führen Sie unbedingt `ant clean jar` aus. Bitte schalten Sie den Debug-Modus vor der Abgabe aus.

Ein- und Ausgabeformat von SAT-o-ban. Die äußere Syntax der `levels.txt`-Datei ist selbsterklärend. Sie können sie anhand des mitgelieferten Beispiels nachvollziehen. Die innere Syntax eines Sokobanlevels ist unter `http://www.sokobano.de/wiki/index.php?title=Level_format` erklärt.

Eine Lösung ist eine Zeichenkette aus Zeichen `u`, `d`, `l`, `r`, die die Bewegungen des Spielers nach oben, unten, links und rechts kodieren. Die sonst übliche Großschreibung für Schiebebewegungen verwenden wir nicht. Demnach sieht die Lösung für den ersten Level aus der mitgelieferten Datei `levels.txt` so aus: `rddlruulduullddr`.

Abgabe der Lösungen. Die Abgabe der Lösung dieser Aufgabe erfolgt ebenfalls über die Webseite zur Vorlesung. Geben Sie dabei Ihre Quellcode-Datei `Solver.java` als ASCII-Text ab. Zusammen mit dem übrigen Rahmenwerk (das Sie nicht abgeben müssen) sollte sie ein Programm ergeben, das:

- die Levels aus der Datei `levels.txt` im aktuellen Verzeichnis einliest.
- genau folgendes auf der Standardausgabe ausgibt: Für jeden Eingabelevel eine Newline-terminierte Zeile: Die Zeile soll entweder die Lösung des Levels im oben-definierten Format beinhalten oder `UNSAT`, falls es keine Lösung der gegebenen Länge gibt.

Auch für Programme, die nicht völlig korrekt sind, werden Punkte anteilig vergeben.

3 Logische Umsetzung

Im folgenden geben wir einige Vorschläge zur logischen Umsetzung des Problems. Diese sind nicht bindend, können aber im gegebenen Rahmenwerk leichter umgesetzt werden.

Logisches Vokabular. Wir numerieren die Felder des Levels mit kartesischen Koordinaten. Das linke obere Feld hat die Koordinaten $(1, 1)$. Für einen gegebenen Level der Breite *width* und der Höhe *height*, nehmen wir an, daß für alle Variablenindizes x, y im folgenden gilt: $1 \leqslant x \leqslant \mathit{width}$ und $1 \leqslant y \leqslant \mathit{height}$.[3]

[3] Das Rahmenwerk ist so programmiert, daß die Variablen, deren Feldindizes außerhalb des Levels zeigen, durchaus benutzt werden können. Sie werden dabei immer als *false* interpretiert. Sie müssen sich also nicht extra um Randfälle kümmern.

Wir benutzen folgende aussagenlogische Variablen (Atome):

- $W_{x,y}$ ist wahr gdw. im Feld x, y eine Wand steht.
- $H^t_{x,y}$ ist wahr gdw. der Spielheld sich zum Zeitpunkt t im Feld x, y befindet.
- $C^t_{x,y}$ ist wahr gdw. ein Container sich zum Zeitpunkt t im Feld x, y befindet.

Das Zeitmodell. Die Zeitpunkte sind durch die Aktionen des Spielers markiert. Die initiale Konfiguration gilt als Zeitpunkt $t = 1$, der Zustand nach dem ersten Zug als Zeitpunkt $t = 2$, usw. Wie oben beschrieben, schaut SAT-o-ban nur N Aktionen voraus, der letzte betrachtete Zustand entspricht also $t = N + 1$. Die Zahl $N + 1$ nennen wir auch Leveltiefe (*depth*). Diese ist Teil der Eingabe und muß in der Leveldatei stehen. Für die mitgelieferten Levels ist jeweils die minimale Tiefe, die einer Lösung entspricht, bereits angegeben.[4]

Aktionen. In der Literatur wird manchmal empfohlen, zusätzlich Aktionen durch Variablen zu modellieren (eine für jede mögliche Aktion zu jedem Zeitpunkt). Dies können wir wegen der Einfachheit unserer Domäne weglassen. Gelten in einer erfüllenden Belegung $H^1_{2,2}$ und $H^2_{2,3}$ als wahr, so können wir (unter gewissen weiteren Annahmen) davon ausgehen, daß der Spielheld als erstes einen Schritt nach unten macht.

Erstellen der Formel mit dem Programm. Für jede der o.g. Variablen definiert das Rahmenwerk bereits eine entsprechende Methode. Negieren der Variablen erreichen Sie durch das bloße Voranstellen eines Minuszeichens. Außerdem definiert ist die Methode `clause(...)`, die eine CNF-Klausel in die Eingabedatei des SAT-Solvers schreibt. Z.B. bedeutet der Aufruf

```
clause(-H(x,y,t),-H(x,y,t+1));
```

daß die Klausel $\{\neg H^t_{x,y}, \neg H^{t+1}_{x,y}\}$ Teil der SAT-Solver-Eingabe wird. Diese entspricht der Teilformel

$$H^t_{x,y} \rightarrow \neg H^{t+1}_{x,y} \quad .$$

Sollte diese Formel nun erfüllbar sein, darf der Spieler nicht im Feld (x, y) zum Zeitpunkt t stehenbleiben.

Axiomatisierung der Lösung. Überlegen Sie sich, welche Klauseln eine Sokoban-Level-Lösung der Länge N axiomatisieren. Es lassen sich mindestens folgende Komponenten erkennen:

- Der initiale Zustand, also die Konfiguration des Spielfeldes zum Zeitpunkt $t = 1$. Diese ist durch die Leveleingabe gegeben.
- Der finale Zustand. Ihre Formel muß fordern, daß der Level im Zustand $t = N + 1$ tatsächlich gelöst ist.
- N Zustandsübergänge. In jedem davon muß Ihre Formel für jede mögliche Aktion kodieren, wann die Aktion ausgeführt werden kann (sog. Vorbedingung), und was ihr Effekt ist.
- Framing. Die Formel muß kodieren, daß Felder, die nicht direkt von einer Aktion betroffen sind, unverändert bleiben müssen (ansonsten erhalten Sie „magische“ Lösungen).

Ein Beispiel für ein (partielles) Framing-Axiom ist die Klausel

[4] Im allgemeinen ist die Länge der Lösung natürlich nicht im voraus bekannt. Falls jedoch eine Lösung existiert, kann sie durch wiederholte Lösungsversuche mit steigender Tiefe gefunden werden.

```
clause(-H(x,y,t+1),H(x+1,y,t), H(x,y+1,t), H(x-1,y,t), H(x,y-1,t));
```

Sie entspricht der Formel

$$H^{t+1}_{x,y} \rightarrow H^t_{x+1,y} \vee H^t_{x,y+1} \vee H^t_{x-1,y} \vee H^t_{x,y-1}$$

und besagt, daß das Erscheinen des Helds im Feld (x, y) zum Zeitpunkt $t+1$ nur dann möglich ist, wenn der Held zum Zeitpunkt t auf einem der Nachbarfelder stand.[5] Zusammen mit anderen geeigneten Axiomen kann diese Klausel „Beamen“, Duplizieren des Helds, und andere unerwünschte Phänomene verhindern.

4 Hintergrund: Die SAT-Solver MiniSat und SAT4J

Für diese Aufgabe haben Sie die Wahl zwischen den SAT-Solvern MiniSat[6] und SAT4J. MiniSat ist wegen seiner Leistung preisgekrönt, muß aber von Ihnen heruntergeladen und übersetzt werden (was wir empfehlen). SAT4J ist (zumindest in der Standardkonfiguration) etwas langsamer, wird aber direkt als Teil des Rahmenwerks mitgeliefert. Die Auswahl des verwendeten Solvers erfolgt über die boolesche Variable `SAT4JAVA` in der Klasse `sokoban.Solver`.

Eingabeformat. Beide Solver benutzen für ihre Ein- und Ausgabe das sogenannte DIMACS-Format, das auf sehr simple Weise die Formulierung (großer) aussagenlogischer Probleme in Klauselform erlaubt.

Dieses Format besteht aus einer Kopfzeile und mehreren Klauselzeilen. Die Kopfzeile enthält neben Schlüsselwörtern (“`p cnf`” für “problem in conjunctive normal form”) die Anzahl der verwendeten AL-Variablen und die Zahl der Klauseln. Es folgen die Klauselzeilen, von denen jede aus einer leerzeichenseparierten Liste von Literalen (von 0 verschiedene ganze Zahlen) gefolgt von einer abschließenden 0 besteht. Negative Zahlen stehen dabei für die negierten Variablen. Beispielsweise entspricht die Eingabe

```
p cnf 3 2
-1 2 0
1 -3 -2 0
```

der aussagenlogischen Formel $(\neg P_1 \vee P_2) \wedge (P_1 \vee \neg P_3 \vee \neg P_2)$.

Ausgabe von MiniSat. MiniSat liefert das Ergebnis in einer Datei zurück. Ist die Klauselmenge unerfüllbar, so lautet die erste und einzige Zeile `UNSAT`. Für den Fall der Erfüllbarkeit der Klauselmenge lautet diese Zeile `SAT` und die zweite Zeile enthält eine Beschreibung der erfüllenden Interpretation. Dabei werden diejenigen AL-Variablen, die als wahr interpretiert werden, durch eine positive ganze Zahl und diejenigen, die zu falsch ausgewertet werden, durch eine negative dargestellt. Das Ausgabeformat von SAT4J ist sehr ähnlich.

Aufrufen von MiniSat. Falls Sie mal MiniSat von Hand starten wollen, erfolgt das durch den Aufruf `minisat`, gefolgt von zwei Kommandozeilenargumenten: `minisat infile out`. Das erste ist die Datei mit der DIMACS-Eingabe und das zweite ist der Dateinamen, unter dem MiniSat das Ergebnis speichern soll. MiniSat gibt dann auf die Standardausgabe noch eine Menge Statusinformationen über die Anzahl der verwendeten Klauseln, benötigter Zeit, Speicher usw. aus.

5 Hinweise

Versuchen Sie zunächst ein einfacheres Problem zu lösen: den Spieler in einem Lager ohne Container automatisch zu einer gegebenen Stelle zu navigieren.

[5] Das nennt man auch „explanatory framing“.

[6] `http://minisat.se/`