Karlsruher Institut für Technologie
Institut für Theoretische Informatik
Prof. Dr. B. Beckert
Thorsten Bormer

Formale Systeme, WS 2010/2011

Praxisaufgabe 1: SATromino
Lösen eines Packungsproblems durch SAT-Solving

Abgabe am 02.01.2011.

Hinweis: Bitte beachten Sie, dass die Praxisaufgabe in Einzelarbeit zu bearbeiten ist.

1 Allgemeines

Aufgabe. Das Ziel dieser Praxisaufgabe ist, einen automatischen Spieler für eine Variante aus der Gruppe von Tetromino-Spielen (wie z.B. auch Tetris) zu entwickeln. Dafür sollen Sie das durch das Spiel gegebene Packungsproblem 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.

Tetromino. Ein Tetromino-Spiel besteht aus einem rechteckigen Spielfeld, in das nacheinander Spielsteine (die Tetrominos) von oben nach unten fallen. Erreicht ein solcher Block den unteren Rand des Spielfelds, oder trifft auf einen bereits gelegten Spielstein, so ist der aktuelle Zug beendet und der nächste Stein beginnt herabzufallen.

Ziel des Spielers ist es, diese Steine durch Verschieben und Rotieren zu einem möglichst kompakten Block zusammenzusetzen, der die volle Breite des Felds einnimmt. Ist eine Reihe des Felds vollständig gefüllt, so wird der Inhalt dieser Reihe entfernt (die weiter oben liegenden Blöcke füllen die entstehenden Lücken dabei auf).

Das allgemeine Spielprinzip und weiterführende Informationen sind z.B. unter `http://de.wikipedia.org/wiki/Tetris` aufgeführt.

Tetris kann als eine Art Planungsproblem aufgefasst werden: Gegeben ist eine Weltformalisierung, 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.

Für einige Tetromino-Probleme konnte gezeigt werden, dass sie NP-vollständig sind, wie etwa die Anzahl an vollständig gefüllten Reihen zu maximieren. [1]

Präzisierung der Aufgabe. Schreiben Sie ein Java-Programm, das einen als Eingabe dienenden Tetromino-Level samt einer Abfolge von Tetromino-Spielsteinen in eine aussagenlogische Formel übersetzt. Ein Tetromino-Level besteht dabei aus einer Vorgabe für die Belegung der Felder zu Beginn des Spiels, sowie einer Ziel-Belegung, die erfüllt sein muss, nachdem alle Spielsteine der Abfolge gelegt wurden.

Die Formel soll genau dann erfüllbar sein, wenn sich ausgehend von der Start-Belegung die Ziel-Belegung durch Platzieren der Spielsteine der vorgebebenen Abfolge erreichen lässt. In diesem Fall sollte Ihr Programm die Levellösung aus der erfüllenden Belegung der Formel ablesen.

Die Spielregeln. Für diese Formalisierungsaufgabe betrachten wir eine Tetromino-Variante mit folgenden Eigenschaften:

- Die vollständige Sequenz der herabfallenden Spielsteine ist zu Beginn des Spiels festgelegt und bekannt ("offline Version" des Spiels).
- Die Spielsteine bewegen sich nicht nach einer bestimmten Zeit selbständig nach unten, sondern diese Bewegung wird durch eine explizite Aktion ("DOWN") ausgelöst. Dies hat auch zur Folge, dass ein Spielstein beliebig oft rotiert und nach rechts und links bewegt werden kann, bevor er nach unten bewegt wird.
- Es sind nur solche Platzierungen der Steine erlaubt, die auch *nur* durch Rotieren und Links- bzw. Rechtsbewegungen direkt zu Anfang des Zuges möglich wären. (Das Einschieben von Spielsteinen, wie in Abb. 1 dargestellt, ist also nicht erlaubt.)
- Vollständig gefüllte Reihen werden nicht entfernt, sondern bleiben auf dem Spielfeld unverändert erhalten. (Dadurch entfällt die Modellierung des Nachrückens weiter oben liegender Reihen in einem solchen Fall.)

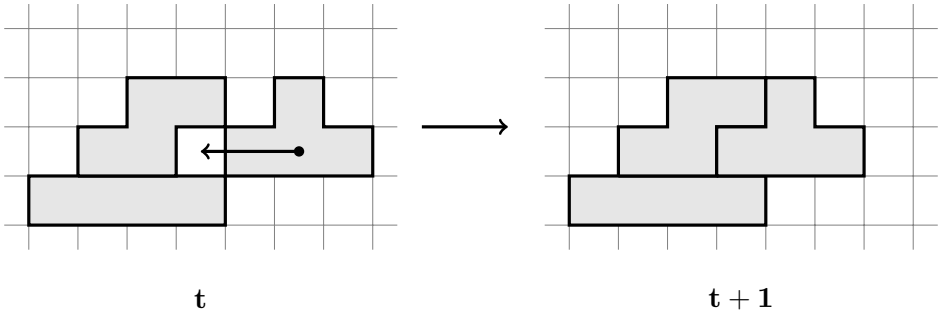

Abbildung 1: Einschieben von Blöcken ist in SATromino *nicht* erlaubt.

Die in dieser Tetromino-Variante verwendeten Spielsteine entsprechen von der Gestalt der des Tetris-Spiels; die Bezeichnung der Steine, sowie deren Rotationszustände sind in Abb. 2 Abb. 3 dargestellt. Jeder neue Spielstein erscheint an Position $x = width/2, \quad y = 0$ (mit *width* als Breite des Spielfelds). Die Figuren `S` und `Z` erscheinen im Rotationszustand `LEFT`, alle anderen im Zustand `DOWN`.

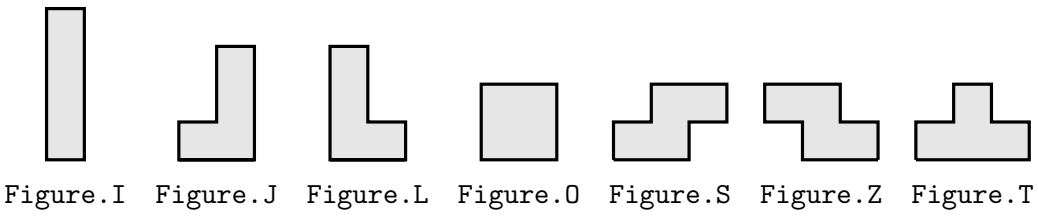

Abbildung 2: Bezeichnung der Tetrominos

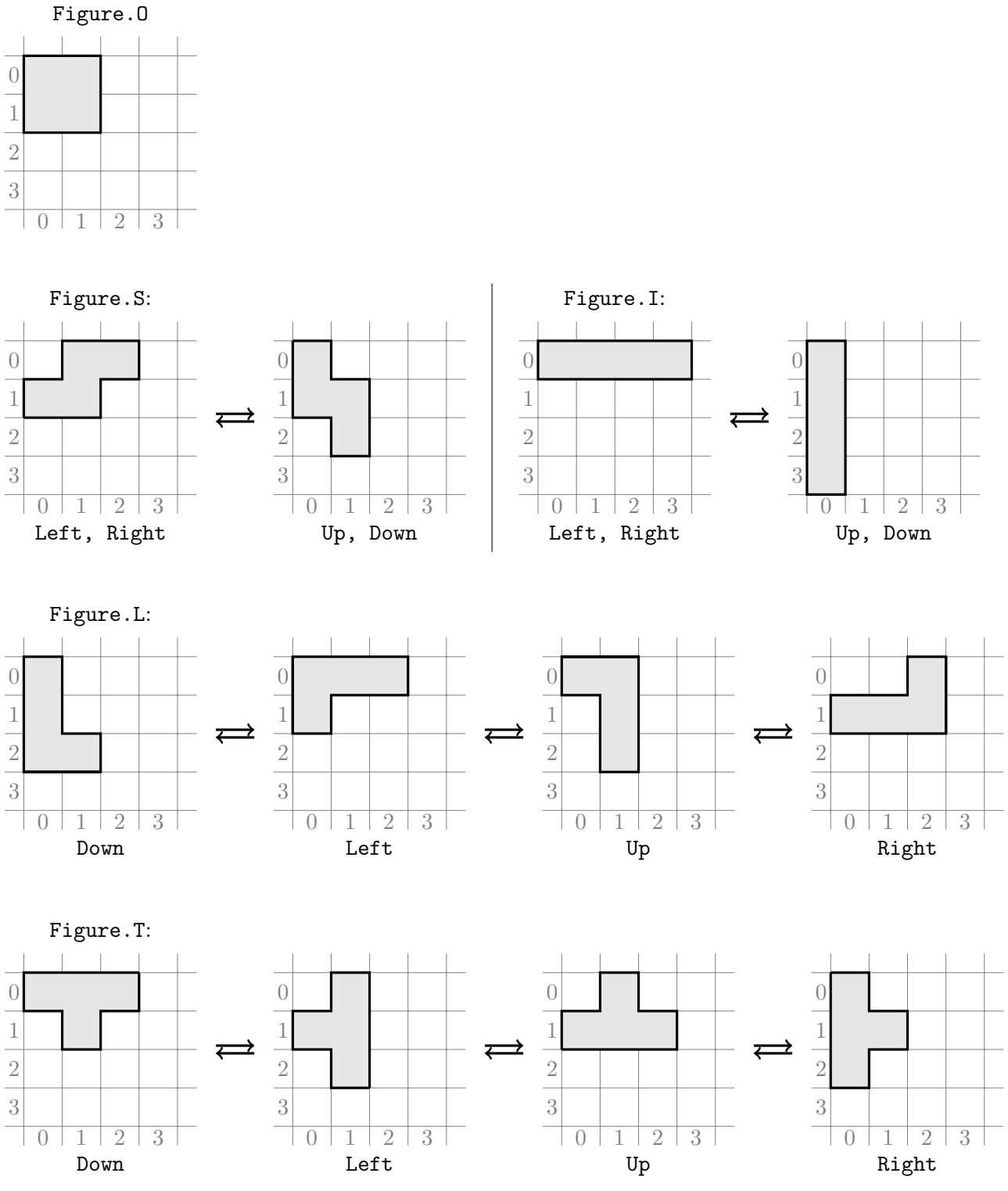

Abbildung 3: Rotation der Tetrominos

2 Technischer Rahmen

Das Rahmenwerk. Auf der Vorlesungswebseite finden Sie das Archiv `satromino.zip` mit dem Rahmenwerk. Das Rahmenwerk basiert auf der freien Java-Implementierung JETRIS[1].

Sie brauchen nur die Klasse `net.sourceforge.jetris.Solver` zu verändern. Bitte ändern Sie andere Klassen nicht ohne Rücksprache. Sollten Sie ausser an den Methoden `init` und `move` noch weitere Veränderungen an der Klasse `Solver` vornehmen, so ist folgendes zu beachten: Die Klasse `Solver` muss einen Konstruktor der Signatur `public Solver(Level lvl)` enthalten, der (wie auch bereits vorgegeben) für den Eingabelevel die Lösung des Levels in Form einer Liste von `Action`s in die statische Variable `LinkedList<Action> solutionKeys` schreibt.

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 satromino.jar` starten (bzw. mit einem Doppelklick auf die Jar-Datei, falls Ihr System das unterstützt).

Datenfluss. Beim Starten liest SATromino die Levels aus der Datei `levels.txt` im aktuellen Verzeichnis. Für jeden gefundenen Level passiert jeweils folgendes:

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

Debug-Modus. Das Rahmenwerk hat einen Debug-Modus, der viele hilfreiche Zusatzinformationen ausgibt (u.a. eine lesbare Interpretation der SAT-Solver-Ausgabe).

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 SATromino. Die äußere Syntax der `levels.txt`-Datei ist selbsterklärend. Sie können sie anhand Abb. 4 nachvollziehen. Die innere Syntax eines Sokobanlevels, bzw. des `Start`- und `Goal`-Blocks der Datei ist wie folgt:

. Ein Punkt kodiert, dass das Feld zu Beginn leer ist bzw. das Feld am Ende der Spielsequenz frei bleiben *muss*.

? Das Fragezeichen ist nur im `Goal`-Block erlaubt und zeigt an, das die Belegung des Felds am Ende der Spielsequenz irrelevant ist

+ Felder, die mit einem Plus markiert sind, *müssen* am Ende bzw. zu Anfang des Spiels gefüllt sein

Eine Lösung ist eine Zeichenkette aus Zeichen `l`, `r`, `d`, `t`, die die Bewegungen der Spielsteine nach links, rechts, unten, sowie eine Drehung ("turn") kodieren. Die Bewegung "unten" ist hier explizit notwendig, da sich die Spielsteine nicht nach einer bestimmten Zeit von selbst nach unten bewegen. Des weiteren beendet die Aktion "unten" den Spielzug, wenn der Spielstein seine endgültige Position erreicht hat.

Demnach sieht eine Lösung für den ersten Level aus der mitgelieferten Datei `levels.txt` so aus: `TLLDDDDDDDDDDTTTRRDDDDDDDDDTTTRDDDDDDDLDDDDDDDTRRRDDDDDDD`.

[1] `http://jetris.sourceforge.net/`

```
;;Name: TestLevel
;;Size: 10 x 10
;;Sequence: I,J,L,S,Z,T,O,J
;;Start
..........
..........
..........
..........
..........
.........+
;;Goal
??????????
??????????
???.??.???
++++++++++
++++.+++++
++++++++++
```

Abbildung 4: Beispiel eines Tetromino-Levels in der SATromino Eingabesprache

Grafische Oberfläche von SATromino. Der Hauptbestandteil der GUI von SATromino ist äquivalent zur ursprünglichen Version JETRIS, daher hier nur eine Erklärung der für die SAT-Aufgabe wichtigen Veränderungen: Das Auswählen der aus der Datei `levels.txt` eingelesenen Level geschieht über die Buttons `<` (vorheriger Level) und `>` (nächster Level). Entsprechend der Levelbeschreibung sind die einzelnen Felder des Spielfelds farbig markiert: grün umrandete Felder sind zum Ende des Spiels mit Blöcken zu belegen, grau hinterlegte müssen frei bleiben und blau umrandete Felder sind für die Lösung irrelevant.

Der aktuelle Level kann dann durch Anwählen des `Solution`-Buttons mit Hilfe des SAT-Solvers und Ihrer Implementierung gelöst werden. Die gefundene Lösung wird anschließend in der GUI wiedergegeben.

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 $1 \leqslant y \leqslant \mathit{height}$.[2]

Wir benutzen folgende aussagenlogische Variablen (Atome):

- B_t^s ist wahr gdw. der aktuell zu platzierende Block in Schritt s vom Typ t ist.
- R_d^s ist wahr gdw. der Block in Schritt s sich im Rotationszustand d befindet.
- $P_{x,y}^s$ ist wahr gdw. sich der Spielstein im Schritt s an der Stelle x, y im Feld befindet.
- $C_{x,y}^s$ ist wahr gdw. in Schritt s das Feld x, y gefüllt ist.

Das Zeitmodell. Einem Schritt entspricht das direkte Platzieren eines Spielsteins an eine gültige Position innerhalb des Spielfelds – die einzelnen Zwischenschritte wie z.B. das Fallen des Blocks werden dabei nicht modelliert, um die Aufgabe zu vereinfachen. Allerdings müssen die Spielsteine so platziert werden, dass es möglich ist, die Positionierung der Spielsteine durch einzelne Aktionen (z.B. in der grafischen Oberfläche) nachzuvollziehen.[3]

Die initiale Konfiguration gilt als Zeitpunkt $t = 0$. Zu diesem Zeitpunkt ist das Spielfeld nur mit den aus der Levelbeschreibung vorgegebenen Feldern gefüllt – der erste Spielstein ist zum Zeitpunkt $t = 1$ zu platzieren. Im nächsten Zustand zum Zeitpunkt $t = 2$ ist dann der zweite Spielstein zu platzieren, usw. Der finale Zustand ist mit der Länge der vorgegebenen Blocksequenz N im Zeitpunkt $t = N$ erreicht.

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(-B(t,s), -P(x,y,s), C(x,y,s));
```

dass die Klausel $\{\neg B_t^s, \neg P_{x,y}^s, C_{x,y}^s\}$ Teil der SAT-Solver-Eingabe wird. Diese entspricht der Teilformel

$$B_t^s \wedge P_{x,y}^s \rightarrow C_{x,y}^s \ .$$

Diese Formel drueckt aus, dass das Platzieren eines Block des Typs t im Schritt s auf Feld (x, y) dazu führt, dass das Feld zum Zeitpunkt s gefüllt ist.

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

- Der initiale Zustand, also die Konfiguration des Spielfeldes zum Zeitpunkt $t = 0$. Diese ist durch die Leveleingabe gegeben.

[2] 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.

[3] In der vorgegeben Implementierung wird Ihre Lösung so interpretiert, dass eine Position des Spielsteins durch beliebiges Drehen und Rechts-/Linksbewebung zu Anfang eines Spielzugs erreicht wird. Die Level sind so gestaltet, dass eine Kollision mit belegten Feldern bei diesen Aktionen nicht auftritt und somit ignoriert werden kann.

- Der finale Zustand. Ihre Formel muss fordern, dass der Level im Zustand $t = N$ tatsächlich gelöst ist. Auch diese Bedingung ist durch die Leveleingabe gegeben.
- 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

```
clause(-C(x,y,s), C(x,y,s+1));
```

Sie entspricht der Formel

$$C^s_{x,y} \rightarrow C^{s+1}_{x,y}$$

und besagt, dass einmal gefüllte Felder über den Spielverlauf gefüllt bleiben.

4 Hintergrund: Die SAT-Solver MiniSat und SAT4J

Für diese Aufgabe haben Sie die Wahl zwischen den SAT-Solvern MiniSat[4] 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.

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

5 Hinweise

Verwenden Sie die Datenstrukturen `int[] arrX` und `int[] arrY` der Klasse `Figure` aus der Implementierung, um die Klauseln aus der Gestalt der Spielsteine zu generieren. Durch Aufruf der Methode `rotationRight` eines `Figure`-Objekts werden diese beiden Arrays so verändert, dass sie die Koordinaten der gefüllten Felder des Spielsteins nach einer Rechtsdrehung enthalten.

Literatur

[1] Erik Demaine, Susan Hohenberger, and David Liben-Nowell. Tetris is hard, even to approximate. In Tandy Warnow and Binhai Zhu, editors, *Computing and Combinatorics*, volume 2697 of *Lecture Notes in Computer Science*, pages 351–363. Springer Berlin / Heidelberg, 2003.