Karlsruher Institut für Technologie
Institut für Theoretische Informatik
Prof. Dr. Bernhard Beckert
Dr. Thorsten Bormer, Dr. Vladimir Klebanov, Dr. Mattias Ulbrich

Formale Systeme, WS 2015/2016

Praxisaufgabe 1: Das Puzzle "Light Up" mittels SAT Solver lösen

Abgabe der Lösungen bis zum 6.1.2016 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 in der Abschlussklausur im Verhältnis 1:10 verrechnet werden.

Das Puzzle Light Up

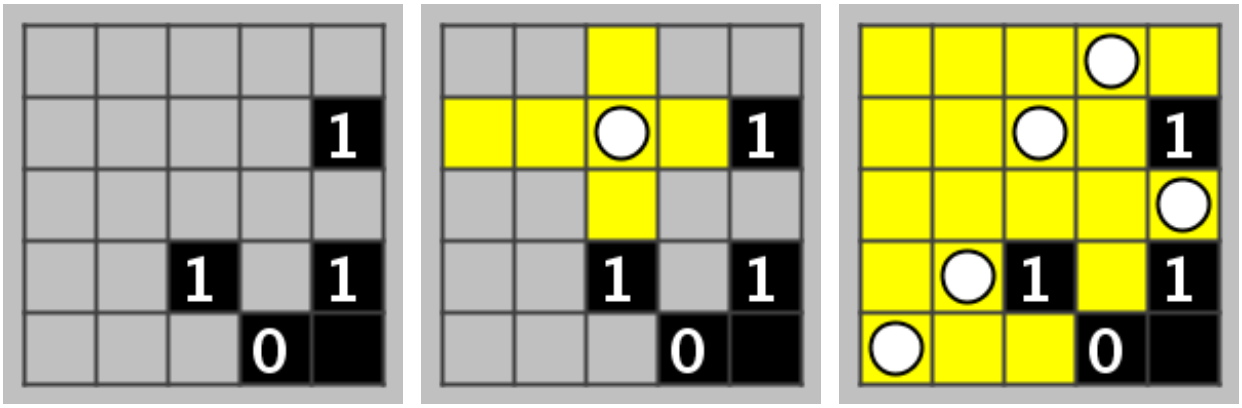
Das Logik-Puzzle „Light Up" ist eine NP-vollständige[1] Denksportaufgabe, die Sie in dieser Praxisaufgabe mit Hilfe eines Lösers für aussagenlogische Erfüllbarkeit (*engl.* satisfiability solver oder SAT solver) lösen sollen.

Die Aufgabe des Puzzles Light Up besteht darin, auf einem quadratischen Spielbrett Lampen so auf den Spielfeldern anzubringen, dass alle Felder ausgeleuchtet werden, aber zwei Lampen sich nie gegenseitig beleuchten. Auf dem Spielfeld sind Wände (*blocks*) platziert, die die Reichweite der Lampe beschränken. Auch können Wände mit einer Randbedingung versehen werden, wie viele Lampen in ihrer Umgebung aufgestellt werden müssen. Formaler beschrieben:

> **Spezifikation:** Gegeben ist ein schachbrettartiges Spielbrett der Seitenlänge N. Die Felder des Brettes sind zu Beginn entweder *leer* oder *mit einer Wand* belegt. Wände sind entweder nicht gekennzeichnet oder mit einer Ziffer zwischen 0 und 4 markiert.
>
> Platzieren Sie auf die leeren Felder Lampen, so dass gilt:
>
> - Zu jedem anfänglich leeren Feld gibt es eine Lampe, die in derselben Zeile oder derselben Spalte steht, ohne dass eine Wand zwischen dem Feld und der Lampe liegt.
> - Es gibt keine zwei Lampen, die in der selben Zeile oder Spalte stehen, ohne dass eine Wand zwischen ihnen liegt.
> - Jede mit einer Ziffer z markierte Wand grenzt *genau* an z mit Lampen besetzte Felder. Zwei Felder grenzen aneinander, wenn sie nebeneinander oder übereinander liegen (jedoch nicht diagonal zu einander).

Abbildung 1 zeigt ein Beispiel für ein Spielbrett der Größe $N = 5$. Neben der Ausgangssituation (a) ist eine Situation mit einer platzierten Lampe (b), sowie eine Lampenplatzierung, die alle Bedingungen aus der Spezifikation erfüllt, dargestellt.

[1] Siehe dazu http://www.cs.umass.edu/~mcphailb/papers/2005lightup.pdf



(a) Das ursprüngliche Spielbrett (b) Die Ausleuchtung *einer* Lampe (c) Eine Belegung mit Lampen, die alle Bedingungen erfüllt

Abbildung 1: Ein Beispiel-Spielbrett für $N = 5$

Die Aufgabe

Die Praxisaufgabe besteht darin, ein Java-Programm zu schreiben, das

1. Light-Up-Spielbretter in aussagenlogische Klauselmengen transformiert, so dass jede erfüllende Belegung der Klauselmenge einer Lösung des Puzzles entspricht,
2. einen vorgegebenen SAT-Solver (`SAT4J`) aufruft, um eine solche erfüllende Belegung zu finden,
3. die gefundene Belegung in eine Lösung des Puzzles zurückübersetzt.

Die Bedingungen an die Lampenplatzierung müssen dazu aussagenlogisch formalisiert werden. Sie müssen entscheiden, was Sie durch AL-Variablen modellieren und die Forderungen aus der Problemstellung in Klauseln formulieren. Typischerweise wird eine Formalisierung mit $O(N^2)$ Variablen und $O(N^3)$ Klauseln auskommen.

Bitte beachten Sie: Die Klauselmenge wird von dem Java-Programm für jede Instanz (jedes Spielbrett) jeweils neu erstellt. Sie kodiert sowohl die allgemeinen Regeln des Spiels als auch die Lage der Wände auf dem gegebenen Spielbrett.

Kodierung der Spielbretter (Ein- und Ausgabe)

Spielbretter werden für diese Aufgabe durch Zeichenketten repräsentiert, die als Eingabe und Ausgabe für Ihr Java-Programm dienen. Die unten erwähnten Java-Klassen ermöglichen es Ihnen, diese Zeichenketten zu parsen und so die Spielbretter in Ihr Programm einzulesen. Auch können Sie damit die Lösung, das heißt ein Spielbrett mit platzierten Lampen, als Zeichenkette darstellen. Um die Analyse einfacher zu machen, können Spielbretter außerdem visualisiert werden.

Ihre Aufgabe ist es, die Methode `solve` in der vorgegebenen Klasse `MyLightUpSolver` zu vervollständigen. Diese Methode erwartet als Parameter einen String, der ein Spielbrett ohne Lampen beschreibt, und liefert einen String zurück, der dieses Spielbrett mit nach den vorgegebenen Regeln platzierten Lampen kodiert.

Rahmenwerk

Um Ihnen die Bearbeitung der Aufgabe zu erleichtern, stellen wir Ihnen ein Rahmenwerk zur Verfügung, so dass Sie sich für die Bearbeitung auf die Aufgabe konzentrieren können und nicht mit der Ansteuerung des SAT-Solvers beschäftigen müssen.

Auf der Seite der Vorlesung finden Sie folgende Dateien:

- `MyLightUpSolver.java`: Ein Skelett, das Sie für Ihre Implementierung der Lösung erweitern sollen. Falls Sie weitere Klassen benötigen, speichern Sie diese bitte ebenfalls in dieser Datei.
  Beachten Sie:
  - Der Name der Klasse `MyLightUpSolver`, sowie die Signatur der Methode `solve` sind fest vorgegeben, um Ihre Abgabe automatisch überprüfen zu können.
  - Die Methode `solve` muss mehrfach hintereinander für verschiedene Eingaben auf einer Instanz der Klasse `LightUpSolver` aufrufbar sein.
- `sat4j.jar`: Implementierung des SAT-Solvers SAT4J
- `lightUp.jar`: Eine Paket um:
  - Zeichenketten in Spielbrettobjekte zu konvertieren und umgekehrt (Konstruktor der Klasse `Lights`, sowie die `toString()`-Methode dieser Klasse),
  - den Status einzelner Felder abzufragen und zu setzen (Methoden `getField` und `setLight`, `setBlock`, ... der Klasse `Lights`),
  - sich Spielbretter grafisch anzeigen zu lassen (Methode `showWindow` der Klasse `Lights`),
  - Klauselmengen zu erstellen, die ausdrücken, dass genau n aus vier AL-Variablen wahr sind (Methode `Clause.exactlySetN`).

Hinweis: Die vollständige Beschreibung der Funktionalität des Pakets finden Sie auf der Webseite der Vorlesung als JavaDoc-Dokumentation.

Probelauf

Die von uns bereitgestellte Datei `MyLightUpSolver.java` enthält eine minimale Implementierung, die einige Beispiele zur Verwendung der von `lightUp.jar` angebotenen API zeigt und sich zudem dazu eignet, die Konfiguration Ihrer Java-Umgebung zu überprüfen.

Diese Implementierung lässt sich wie folgt in lauffähigen Code übersetzen, sofern sich alle oben angegebenen Dateien im gleichen Verzeichnis befinden:

```
javac -cp sat4j.jar:lightUp.jar MyLightUpSolver.java
```

Die `main`-Methode der Klasse `LightUpSolver` erwartet eine Beschreibung des Spielbretts in Textform als einzigen Kommandozeilenparameter.

Starten Sie das kompilierte Programm wie folgt:

```
java -cp .:sat4j.jar:lightUp.jar MyLightUpSolver 5:i1g1a1c0B
```

dann sollten sich nacheinander je ein Fenster mit der graphischen Darstellung des Spielfelds ohne, sowie mit einer Lampe öffnen (es handelt sich hierbei um modale Fenster, sodass sie den Ablauf des Programms blockieren, bis sie geschlossen werden). Auf der Kommandozeile sollte nach vollständiger Ausführung etwa folgende Ausgabe erscheinen:

```
1
SAT: solving took 5 ms.
5:fLb1g1a1c0B
```

Der SAT Solver SAT4J

Für diese Aufgabe verwenden wir das Werkzeug SAT4J[2]. SAT4J ist eine reine Java-Implementierung eines SAT-Solvers. SAT4J wird in vielen Java-basierten Systemen eingesetzt, u.a. im Alloy Analyzer[3] oder auch im Eclipse Framework[4].

Eine Klausel wird in SAT4J als Liste von Ganzzahlen repräsentiert, dabei entsprechen positive (negative) Zahlen positiven (negativen) Literalen. Beispielsweise entspricht die aussagenlogische Formel $(\neg P_1 \vee P_2) \wedge (P_1 \vee \neg P_3 \vee \neg P_2)$ etwa folgenden Aufrufen der Methode `addClause` eines `SATSolver`-Objekts `solver`:

```
solver.addClause(-1, 2)
solver.addClause(1, -3, -2)
```

Die Erfüllbarkeit der so gesetzten Klauselmenge überprüft man mittels SAT-Solver durch Aufruf der Methode `solve` der Klasse `SATSolver` – diese Methode liefert `null` zurück, falls die Klauselmenge unerfüllbar ist; gibt es eine erfüllende Interpretation, so wird ein Array von Ganzzahlen zurückgegeben – 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.

Abgabe der Lösung

Bitte reichen Sie den Quelltext Ihres Java-Programms (*eine* Datei als ASCII-Text) bis spätestens 6.1.2016, 23:59 Uhr im **ILIAS-Portal** unter dem Punkt "Praxisblatt 1" ein. **Auch für Programme, die nicht vollständig korrekte Ergebnisse liefern, werden Bonuspunkte anteilig vergeben.**

Testen Ihrer Implementierung Vor der Abgabe Ihrer Lösung im ILIAS-Portal haben Sie die Möglichkeit die Implementierung einerseits anhand einiger Beispielbretter zu überprüfen, die wir auf der Webseite der Vorlesung für Sie zur Verfügung gestellt haben.

Auf der Webseite stellen wir desweiteren ein System bereit, das automatisiert einige Tests durchführt. Wenn Sie eine Lösung hochgeladen haben, werden wir diese auf unserem System übersetzen und auf einer erweiterten Auswahl von Beispielen laufen lassen. Danach wird Ihnen ein Ergebnisprotokoll per E-Mail zugeschickt. **Zur Bewertung ziehen wir aber ausschließlich die Abgaben im ILIAS-System heran, nicht die auf dem Testsystem.**

Ihr Code wird auf unserem Server im „Sandkasten-Modus" ausgeführt werden, d.h., ihm werden viele Rechte (z.B. Zugriff auf das Dateisystem, Netzwerk, etc.) fehlen. Bitte beachten Sie das bei Ihrer Implementierung. Die Aufgabe kann gut ohne solche Zugriffe gelöst werden.

[2] http://www.sat4j.org/
[3] http://alloy.mit.edu/alloy4/
[4] http://mail-archive.ow2.org/sat4j-dev/2008-03/msg00001.html