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

Formale Systeme, WS 2014/2015

Praxisaufgabe 1: SAT-Solver – "Spotlight"

Abgabe der Lösungen bis zum 19.12.2014 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.

Die (erneute) Abgabe einer *eigenen* Lösung zu dieser Aufgabe aus einem früheren Semester ist zulässig. Auch in diesem Fall müssen Sie aber jetzt (im aktuellen Semester) in der Lage sein, Ihre Lösung zu erklären.

Für die vollständige Lösung dieser Praxisaufgabe erhalten Sie **20 Übungspunkte**. Bitte beachten Sie die Erläuterung zu Bonuspunkten auf der Webseite zur Vorlesung.

Das Puzzle Spotlight

Das Logik-Puzzle „Spotlight" (erfunden von E. Friedmann[1]) ist eine NP-vollständige Denksportaufgabe, die wir in dieser Praxisaufgabe mit Hilfe eines aussagenlogischen SAT(isfiability) Solvers lösen wollen.

Spotlight-Aufgaben haben ein rechteckiges *Spielbrett*, auf dem *Felder* liegen. Jedes Feld ist mit einer natürlichen Zahl und einem Pfeil markiert, der in eine horizontale, vertikale oder dialognale Richtung zeigt (s. Abb. 1(a) für ein Beispiel[2]). Die Aufgabe in Spotlight besteht darin, die Felder mit den Farben weiß und schwarz so zu *färben*, dass alle weißen Pfeile *genau* auf die angebene Anzahl von weiß gefärbten Feldern zeigen und alle schwarzen Felder auf eine Anzahl, die von dieser Angabe abweicht (s. Abb. 1(b) für die eindeutige Lösung zu Abb. 1(a)).

Das *Einzugsgebiet* eines Pfeiles wird dabei von der Richtung des Pfeiles bestimmt und erstreckt sich bis zum Spielbrettrand (s. Abb. 1(c)). Das Feld des Pfeiles selbst gehört dabei *nicht* zu seinem Einzugsgebiet.

Formale Beschreibung

Um eventuelle Mehrdeutigkeiten auszuschließen, geben wir im Folgenden eine mathematische Formalisierung der Aufgabenstellung an:

Gegeben ist ein rechteckiges Spielbrett mit Seitenlängen[3] $\mathcal{Z}, \mathcal{S} \in \mathbb{N}_{\geq 1}$. Jedem Feld (z, s) für $1 \leq z \leq \mathcal{Z}$ und $1 \leq s \leq \mathcal{S}$ ist eine Zahl $C_{z,s} \in \mathbb{N}_{\geq 0}$ und Richtungsvektor $(Z_{z,s}, S_{z,s}) \in \{-1, 0, 1\}^2 \setminus \{(0, 0)\}$ des Pfeiles zugeordnet. Der Pfeil auf Feld (z, s) zeigt somit auf das Einzugsgebiet

$$E_{z,s} := \{(z + kZ_{z,s}, s + kS_{z,s}) \mid k \in \mathbb{N}_{\geq 1}, 1 \leq z + kZ_{z,s} \leq \mathcal{Z}, 1 \leq s + kS_{z,s} \leq \mathcal{S}\}$$

von Feldern. Ist auf dem Spielbrett oben links die Koordinate $(1, 1)$, so ist beispielsweise in Abb. 1 $C_{4,1} = 3$, $(Z_{4,1}, S_{4,1}) = (-1, 1)$ und $E_{4,1} = \{(3, 2), (2, 3), (1, 4)\}$.

Aufgabe ist es, eine Menge von Feldern $\mathit{Weiß} \subseteq \{1, ..., \mathcal{Z}\} \times \{1, ..., \mathcal{S}\}$ zu finden, so dass

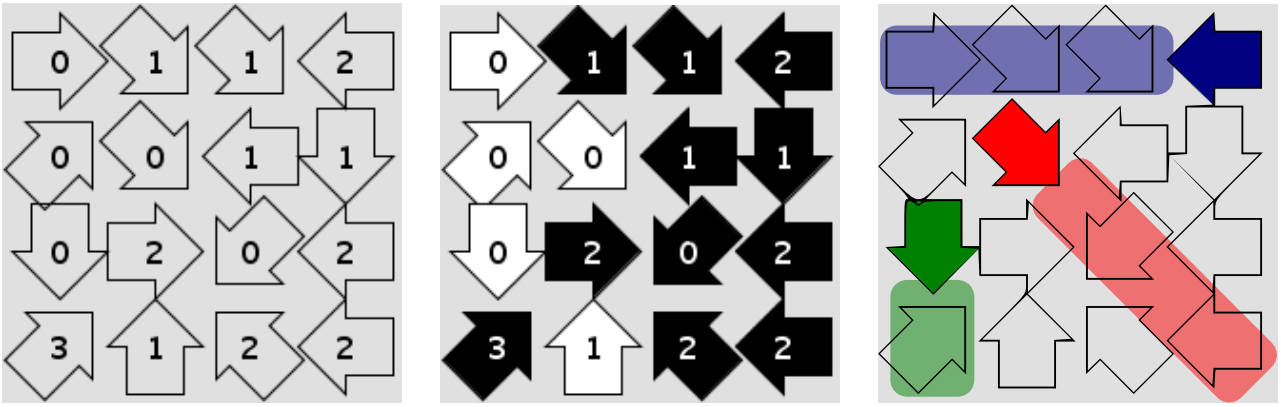
$$(z, s) \in \mathit{Weiß} \iff C_{z,s} = |E_{z,s} \cap \mathit{Weiß}| \tag{1}$$

für alle Felder (z, s) gilt.

[1] http://www2.stetson.edu/~efriedma/arrow

[2] aus der Sammlung http://www.janko.at/Raetsel/Spotlight entnommen

[3] $\mathcal{Z}$ steht für Zeile, $\mathcal{S}$ für Spalte



(a) Das ungefärbte Spielbrett (b) Dasselbe Brett mit korrekt gefärbten Feldern (c) Das Einzugsgebiet von Pfeilen

Abbildung 1: Ein Beispiel-Spielbrett

Die Aufgabe

Die Praxisaufgabe besteht darin, ein Java-Programm zu schreiben, das für ein beliebiges Spotlight-Spielbrett

1. eine Klauselmenge erzeugt, so dass jede erfüllende Belegung der Klauselmenge einer Lösung des Puzzles entspricht,
2. einen SAT-Solver mit dieser Klauselmenge als Eingabe aufruft, um zu überprüfen ob das Spielbrett eine Lösung hat,
3. die Ausgabe des Solvers in eine korrekte Färbung der Felder des Spielfeldes übersetzt.

Wir bieten Ihnen ein Rahmenwerk an (s.u.), dass Ihnen u.a. die Ansteuerung des SAT-Solvers erleichert.

Übersetzung von Spotlight nach SAT

Es ist naheliegend, die möglichen „weißen Belegungen“ des Einzugsgebietes eines Pfeiles zu betrachten. Ein weißer Pfeil, der mit der Zahl k markiert ist, muss genau k weiße Felder im Einzugsgebiet haben. Man braucht also unter anderem Formeln, die beschreiben, dass genau k Felder aus n weiß gefärbt sind. Man nennt solche Formeln „Kardinalitätsbedingungen“ (*cardinality constraints*).

Eine naive Formalisierung der Kardinalitätsbedingung ist, alle möglichen gültigen Belegungen der Felder aufzuzählen. Allerdings gibt es für ein Einzugsgebiet der Größe n genau $\binom{n}{k}$ gültige Belegungen (also Möglichkeiten, k weiß-gefärbte Felder darin zu verteilen). Der Binomialkoeffizient $\binom{n}{k}$ und somit die Größe einer solchen Formalisierung kann sehr groß werden. Zum Beispiel ist für $k = \frac{n}{2}$, der Koeffizient $\binom{n}{k} > 2^n/\sqrt{2n}$ und wächst somit exponentiell in n.

Wie kann man stattdessen die Kardinalitätsbedingung so formalisieren, dass nicht exponentiell grosse Formeln erzeugt werden? Wir machen dafür etwas scheinbar Kontraproduktives: Wir führen zusätzliche aussagenlogische Variablen ein, die Zwischenergebnisse der Berechnung enthalten. Diese Zwischenergebnisse können später mehrfach verwendet werden, ohne die Berechnung jedes mal neu auszuführen. Dieses Vorgehen[4] wird häufig bei der Kodierung von Problemen in SAT angewandt, so zum Beispiel bei der Erstellung der kurzen KNF.

Statt also die Kardinalitätsbedingungen für je ein n und k von Grund auf zu formalisieren, wollen wir Kardinalitätsbedingungen für kleinere n dafür wiederverwenden. Wir führen für jedes Feld (z, s) und jede

[4] In der Programmierwelt bekannt als *memoization*.

Zahl $k \in \{0, \ldots, n\}$ und jede Richtung r eine neue aussagenlogische Variable $P_{r,z,s,k}$ ein. Diese kodiert die Kardinalitätsbedingung, dass von Feld (z, s) in Richtung r ausgehend *genau* k Felder des Einzugsgebietes weiß sind. Damit diese Variablen die soeben erklärte Bedeutung erhalten, müssen sie durch zusätzliche aussagenlogische Formeln *axiomatisiert* werden. Dies bedarf je einer Äquivalenz pro zusätzliche Variable. Überlegen Sie sich nun, wie zum Beispiel in Abbildung 1(c) für den blauen Pfeil die Bedeutung der Kardinalitätsbedingungsvariablen $P_{\mathit{left},1,4,2}$ unter Zuhilfenahme von $P_{\mathit{left},1,3,1}$ und $P_{\mathit{left},1,3,2}$ ausgedrückt werden kann.

Durch die Einführung von $O(\mathcal{Z} \cdot \mathcal{S} \cdot \max(\mathcal{Z}, \mathcal{S}))$ vielen Zusatzvariablen und ihren Definitionen wird das Problem polynomiell kodiert. Der von uns verwendete SAT-Solver findet so sehr viel schneller eine Lösung als für die exponentielle Aufzählung. Eine weitere Optimierung kann erzielt werden, wenn nicht alle möglichen Zusatzvariablen und Definitionen erzeugt werden, sondern nur diejenigen, die während der Übersetzung des ursprünglichen Problems auftreten („Erzeugung bei Bedarf“).

Nebenbemerkung: Das Problem bleibt trotz polynomieller Kodierung weiterhin NP-hart. Nur bieten wir dem SAT-Solver durch die kürzere Kodierung mehr Angriffsfläche, um geschickte Optimierungen zu finden.

Ressourcenverbrauch

Bei unseren Experimenten mit einer explizit aufzählenden Übersetzung hat der von der Java Virtuellen Maschine standardmäßig zur Verfügung gestellte Speicher bereits bei einer Spielbrettgröße von 16×16 nicht mehr ausgereicht. Bei Verwendung der Parameter[5] `-XX:+UseConcMarkSweepGC -Xmx1536m` wurde die Spielbrettgröße 16×16 (innerhalb von einer Minute) noch bewältigt, 17×17 aber nicht mehr. Bei polynomieller Übersetzung wird selbst ein Spielbrett der Größe 30×30 mit `SAT4J` in wenigen Minuten (0,5 Minuten beim Aufzählungskodieren und 2 Minuten beim Sortieren) kodiert und gelöst. **Um die Bonuspunkte für die Aufgabe zu erhalten, muss Ihre Übersetzung nicht unbedingt polynomiell sein.** Wenn Ihre Implementierung alle Beispiele bis zur Größe 15×15 innerhalb je 30 Minuten lösen kann, ist das hinreichend.

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 auf der Seite zu dieser Aufgabe folgende Dateien:

- `SpotlightSolver.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.
- `sat4j.jar`: Diese Datei benötigen Sie, wenn Sie den SAT-Solver SAT4J verwenden wollen (empfohlen!)
- `spotlight.jar`: Eine Packet um Spielbretter und deren Felder einzulesen, zu manipulieren, auszugeben und anzuzeigen, Klauseln und Formeln zu behandeln, und SAT-Solver anzusteuern.

  Auf der Seite haben wir auch die JavaDoc-Dateien des Pakets bereitgestellt. Die Dokumentation ist auf Englisch gehalten. Dabei wird ein Spielbrett als *board*, ein einzelnes Feld als *field* bezeichnet. Die Pfeile, mit denen die Felder markiert sind, heißen *arrows*, die Zahlen darin *constraints*. Die Einzugsbereiche der Felder heißen *regions*.

  Einige Klassen wollen wir ganz kurz erwähnen:

[5] die wir auch auf unserem Server für Ihre Implementierung verwenden werden

– Die Klasse `SATSolver` kapselt Ihren Aufruf an den SAT-Solver. Sowohl SAT4J als auch MiniSat werden unterstützt. Sie geben nur die einzelnen Klauseln Ihrer Formalisierung an und können darauf den SAT-Solver agieren lassen.
– Die Klasse `Board` kapselt ein Spielbrett. Spielbretter können aus Beschreibungsdateien oder -zeichenketten erzeugt werden und Bretter können in einem Fenster (mit Färbung) angezeigt werden, was bei der Fehlersuche sehr hilfreich sein kann.
– Die Klasse `Field` beschreibt ein einzelnes Feld auf einem Brett. Felder können insbesondere gefärbt werden.
– Die Klasse `Formula` und ihre Unterklassen erlauben es auf einfache Art aussagenlogische Formeln zu erzeugen. Falls Sie über die Teilformeln iterieren wollen, um die Formel zu verarbeiten, können Sie einen `FormulaVisitor` implementieren. Beispielsweise können Sie die Formel so in ihre kurze konjunktive Normalform überführen.

Die SAT-Solver

Wir wollen später konkrete Spotlight-Puzzles in aussagenlogische Formelmengen übersetzen, für die wir dann eine erfüllende Belegung suchen. Für diese Aufgabe wollen wir zwei Werkzeuge betrachten: MiniSat und SAT4J.

MiniSat Das Werkzeug MiniSat[6] ist ein für seine Leistung preisgekrönter SAT-Solver, der gleichzeitig ein sehr simples Werkzeug mit einer überschaubaren Schnittstelle ist.

Auf der Homepage von MiniSat können Sie die aktuelle Version herunterladen und übersetzen. Die Anforderungen an zusätzliche Bibliotheken sind sehr gering, so dass die meisten Systeme eine Übersetzung gestatten.

SAT4J SAT4J ist eine reine Java-Implementierung eines SAT-Solvers. SAT4J wird in vielen Java-basierten Systemen eingesetzt, u.a. im Alloy Analyzer[7] oder auch im Eclipse Framework[8].

Da der Solver in Java implementiert ist, ist er naturgemäß etwas langsamer als MiniSAT. Dafür ist er Bestandteil des Rahmenwerks, das wir Ihnen anbieten und Sie müssen ihn nicht selbst herunterladen und übersetzen.

Ein-/Ausgabeformat

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

In diesem Format werden Variablen durch Zahlen und Klauseln durch Zahlenfolgen repräsentiert: Eine DIMACS-Klausel ist eine leerzeichenseparierten Liste von Literalen (von 0 verschiedene ganze Zahlen) gefolgt von einer abschließenden 0. Negative Zahlen stehen dabei für die negierten Variablen. Beispielsweise entspricht die Eingabe

```
-1 2 0
1 -3 -2 0
```

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

Die Solver liefern folgendes Ergebnis zurück: Ist die Klauselmenge unerfüllbar, so lautet das Ergebnis `UNSAT`. Für den Fall der Erfüllbarkeit der Klauselmenge lautet es `SAT` zusammen mit einer Beschreibung der erfüllenden Interpretation. Dabei werden diejenigen AL-Variablen, die als wahr interpretiert werden,

[6] http://minisat.se/
[7] http://alloy.mit.edu/alloy4/
[8] http://mail-archive.ow2.org/sat4j-dev/2008-03/msg00001.html

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 19. Dezember, 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.

Des Weiteren stellen wir ein System bereit, das automatisiert einige Testfälle durchführt. Um dieses System nutzen zu können, registrieren Sie sich mit Ihrer Matrikelnummer unter:

`http://formal.iti.kit.edu/teaching/FormSysWS1415/praxis1-testen/`

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 auf der Seite ein Ergebnisprotokoll angeboten. Indem Sie erneut den Quelltext hochladen, aktualisieren Sie Ihre Abgabe. **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.