Karlsruher Institut für Technologie
Institut für Theoretische Informatik
Prof. Dr. Bernhard Beckert
Dr. Alexander Weigl, Samuel Teuber

Formale Systeme, WS 2022/2023

Praxisaufgabe 1: Das Puzzle „Kuromasu“ mittels SAT-Solver lösen

Abgabe der Lösungen bis zum 08.01.2023 über das
ILIAS-Portal zur Vorlesung (unter ”Klausurbonus Übungsaufgaben”)

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 Verhältnis 1:10 als Bonuspunkte auf die bestandene Abschlussklausur angerechnet (max. ein Notenschritt Verbesserung).

Das Puzzle Kuromasu

Das Logik-Puzzle „Kuromasu“ 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. Kuromasu wird auf einem rechteckigen Spielbrett gespielt. Manche Felder sind dabei mit Zahlen versehen (Zahlenfeld). Die Aufgabe ist nun, für ein Spielbrett eine Schwarz-Weiß-Einfärbung zu finden.

Im Einzelnen sind folgende Bedingungen von einer Einfärbung zu erfüllen:

1. Zwei schwarze Felder sind niemals horizontal oder vertikal direkt benachbart.
2. Alle Zahlenfelder sind weiß.
3. Für jedes Zahlenfeld ist die in ihm enthaltene Zahl gleich der Anzahl der von ihm aus sichtbaren weißen Felder. Ein Feld ist von einem anderen Feld aus sichtbar, wenn die Felder in der gleichen Zeile oder der gleichen Spalte sind und kein schwarzes Feld zwischen ihnen liegt. Dabei ist die Sichtbarkeits-Relation reflexiv, d.h. jedes Feld ist von sich selbst aus sichtbar.
4. Jedes weiße Feld ist von jedem anderen weißen Feld aus erreichbar. Dabei ist die Erreichbarkeits-Relation der transitive Abschluss der Sichtbarkeits-Relation.

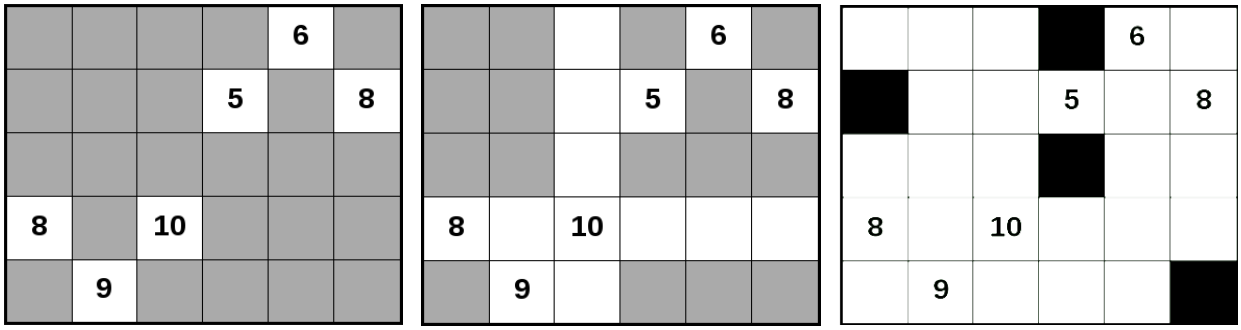
Weitere Informationen finden Sie auf Wikipedia (EN).

Die Aufgabe

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

1. Kuromasu-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,

[1] Siehe dazu https://www.jstage.jst.go.jp/article/ipsjjip/20/3/20_694/_article



(a) Das ursprüngliche Spielbrett (b) Eine Zwischenlösung (c) Eine gültige Belegung von weißen und schwarzen Feldern

Abbildung 1: Ein Beispiel-Spielbrett der Größe 5×6

3. die gefundene Belegung in eine Lösung des Puzzles zurückübersetzt.

Die Bedingungen müssen dazu aussagenlogisch formalisiert werden. Sie müssen entscheiden, was Sie durch AL-Variablen modellieren und die Forderungen aus der Problemstellung in Klauseln formulieren. Achten Sie bitte darauf, dass Ihre Lösung nachvollziehbar und verständlich ist.

Hinweis: Die Erreichbarkeitsrelation für weiße Feldern kann man über eine Erreichbarkeit in k Schritten definieren: Ein Feld A ist von einem Feld B genau dann in k Schritten erreichbar, wenn es ein Feld C gibt, das von B in $k-1$ Schritten erreichbar und direkt benachbart zu A ist.

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.

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.

Auf der Seite der Vorlesung finden Sie ein Archiv mit folgenden Dateien:

- `MyKuromasuSolver.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 `MyKuromasuSolver`, sowie die Signatur der Methode `solve()` und der Konstruktor sind fest vorgegeben, um Ihre Abgabe automatisch überprüfen zu können.
- `kuromasu-1.4.1-all.jar`: Eine Java-Bibliothek (inkl. SAT4J) um:
  - Spielbrettobjekte abzurufen
    (Methoden `Kuromasu.getNumberConstraints`, `Kuromasu.getWidth`, `Kuromasu.getHeight`),
  - den Status einzelner Felder abzufragen und zu setzen
    (Methoden `getField` und `setField` in der Klasse `Solution`),
  - sich Spielbretter grafisch anzeigen zu lassen
    (Methode `Solution.print(System.out)`)

Hinweis: Die vollständige Beschreibung der Funktionalität des Pakets finden Sie auf der Webseite der Vorlesung als JavaDoc-Dokumentation. Ihre Implementierung von `MyKuromasuSolver` muss von

`edu.kit.iti.formal.KuromasuSolver` erben. Dadurch erhalten Sie Zugriff auf folgende Membervariablen:

`game : Kuromasu` beinhaltet die Informationen zum aktuellen Spielbrett.

`solution : Solution` Ein initialisiertes Objekt zum Speichern Ihrer Lösung.

`solver : ISolver` eine Instanz des SAT-Solver (SAT4J). Eine Dokumentation des Interfaces findet sich in der SAT4J-Dokumentation

Probelauf

Nach dem Entpacken des bereitgestellten Archivs sollten Sie einen Ordner mit folgenden Dateien vorfinden:

- kuromasu-1.4.1-all.jar
- riddles/
- Makefile
- MyKuromasuSolver.java

Testen Sie Ihre Systemkonfiguration, indem Sie die Implementierung von `MyKuromasuSolver` zu übersetzen versuchen:

```
javac -cp kuromasu-1.4.1-all.jar MyKuromasuSolver.java
```

Sie können einen Testlauf starten mit:

```
java -cp .:kuromasu-1.4.1-all.jar edu.kit.iti.formal.kuromasu.KuromasuTest
```

Dabei wird Ihre Implementierung auf alle Spielbretter aus dem Ordner `riddles/` angewendet. Sie sollten mehrere Ausgaben folgender Art sehen:

```
== sat.6x5.1110 ======================================================
XXXXXX
XXXXXX
XXXXXX
XXXXXX
XXXXXX
Riddle is satisfible
Your answer is UNKNOWN
```

Weiße Spielfelder erscheinen als „w“, schwarze als „b“ und Felder ohne Angabe als „X“. In dieser Beispielausgabe gibt das System an, dass dieses Puzzle lösbar ist, aber die Implementierung dazu keine Angabe gemacht hat.

Tipp: Beginnen Sie Ihre Tests mit wenigen Spielbrettern im Ordner `riddles` und legen Sie nach und nach weitere Dateien dort hinein. Sollten Sie Problemfälle Ihrer Implementierung erkennen, erzeugen Sie sich für diese Fälle minimale Spielbretter, um diese Probleme isoliert zu lösen.

Kodierung der Spielbretter

Spielbretter werden in Dateien gepeichert und können über das bereitgestellte Framework eingelesen werden. Das Format ist zeilenbasiert. Am Anfang der Zeile steht ein Schlüsselwort „h“, „w“ oder „c“ gefolgt von einer positiven Zahl. Dabei gibt

h $\langle N \rangle$ die Höhe des Spielbrettes, sowie

w $\langle N \rangle$ die Breite des Spielbrettes an.

c $\langle row \rangle$ $\langle column \rangle$ $\langle value \rangle$ definiert ein Zahlenfeld in der entsprechenden Spalte, Reihe und Wertigkeit (Indexierung beginnt bei 0).

Das Spielbrett aus Abbildung 1 wäre wie folgt kodiert:

```
w 6
h 5
c 0 4 6
c 1 3 5
c 1 5 8
c 3 2 10
c 3 0 8
c 4 1 9
```

Im Archiv sind bereits einige Spielbretter mitgeliefert, das Rahmenwerk lädt diese für Sie.

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(new VecInt(new int[]{-1, 2}));
solver.addClause(new VecInt(int[]{1, -3, -2}));
```

Die Erfüllbarkeit der so gesetzten Klauselmenge überprüft man mittels SAT-Solver durch Aufruf der Methode `findModel()` der Klasse `ISolver` – 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. Das zurückgelieferte Modell ist partiell, nur AL-Variablen, für die der SAT-Solver eine Entscheidung getroffen hat, sind aufgeführt. Ist eine Variable nicht im Modell enthalten, ist diese beliebig belegbar.

Abgabe der Lösung

Bitte reichen Sie den Quelltext Ihres Java-Programms (*eine* Datei als UTF-8 Text) bis spätestens 08.01.2023, 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.**

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

Timeouts Wir werden Ihre Lösung mit einer Reihe von nicht-öffentlichen Instanzen evaluieren. Für eine effiziente Evaluation werden wir für die Ausführung Ihrer Lösung einen Timeout setzen. Dieser ist ein Vielfaches der Musterlösung und bereitet nur für Programme mit exponentieller Laufzeit oder Speicherplatzverbrauch Probleme. Um die volle Punktzahl zu erhalten, sollten Sie deshalb mindestens sicherstellen, dass Ihre Lösung die öffentlichen Instanzen in weniger als 3 Minuten lösen kann (dies ist noch keine hinreichende Bedingung!).

Exceptions Ihre Implementierung sollte unter keinen Umständen eine Ausnahmen werfen. Dies bedeutet, dass Randfälle stehts berücksichtigt werden sollten in der Programmierung (z.B. `IndexOutOfBounds` oder `ContradictionException` (SAT4j)). Eine geworfene Ausnahme wird als Programmfehler bewertet.

Scoreboard

Zusätzlich zur Möglichkeit (wie oben beschrieben) lokal zu testen, bieten wir Ihnen auch ein Online Scoreboard zum Vergleich Ihrer Lösungen an. Vor der Abgabe Ihrer Lösung im ILIAS-Portal ermöglicht Ihnen dieses Scoreboard Ihre Lösung auf nicht-öffentlichen Benchmark-Instanzen zu testen und Ihre Lösung (anonymisiert) mit den Lösungen anderer Studierender zu vergleichen.
Das Scoreboard finden Sie auf:

https://flavium.teuber.dev/

Bitte beachten Sie:

- Um Punkte zu erhalten **müssen** Sie Ihre Abgabe im ILIAS hochladen (Abgaben auf flavium werden nicht berücksichtigt)
- Die Umgebung verbietet die Nutzung von Pakten wie `java.io.File`, `java.net` etc.
- Bitte nutzen Sie nur eine Klasse (ggf. mit Inner Classes)
- Das Scoreboard ist unabhängig vom Buchpreis

Häufig gestellte Fragen

Was macht man mit der `ContradictionException`? SAT4J führt bereits beim Hinzufügen von Klauseln oder komplexeren Bedingungen Minimierungen durch. Im Wesentlichen wird dabei Unitpropagation durchgeführt, also eindeutige Belegung von Variablen propagiert. Wird beim hinzufügen einer Klausel durch Unitpropagation die leere Klausel erzeugt, gibt es eine `ContradictionException`. Gleiches passiert bei `addLeast(lits, k)`, wenn weniger Literale (nach Unitpropagation) übrig bleiben als die Mindestanzahl k fordert.

`ContradictionException` deutet immer auf Unerfüllbarkeit hin und tritt dabei häufig im Zusammenhang mit Programmierfehler auf. Es ist ratsam Randfälle eher programmatisch abzufangen, als diese Ausnahme zu provozieren und zu ignorieren.

Wie schnell muss mein Programm sein? Die Praxisaufgabe zielt darauf ab, dass die Transformation des Spielbrettes in eine Klauselmenge in polynomineller Laufzeit stattfinden kann. Im Allgemeinen sollte (für größere Instanzen) die Laufzeit des SAT-Solvers dominieren. Dies bedeutet, dass Sie die Kombinatorik immer geschickt in kodieren sollten. Für die Bonuspunkte ist die Laufzeit zweitrangig, solange es Ihr Programm innerhalb des Timeouts (meistens 5 min.) durch den Testkorpus schafft.

Wie wird bewertet? Aufgrund der zahlreichen Abgaben wird versucht die Bewertung stark zu automatisieren. Dafür ist die Freiheit von Seiteneffekten, z. B. `System.exit` entscheidend. Bei der automatischer Bewertung wird Ihr Programm gegen einen geheimen Testkorpus evaluiert. Sollte Ihr Programm den Testkorpus nicht überleben werden Punkte anteilig den erfolgreichen (un-)lösbaren Puzzlen vergeben. In Fällen in denen dies starke Benachteiligung ist, kann eine manuelle Sichtprüfung erfolgen. Für die manuelle Prüfung ist es wichtig, dass das Programm nachvollziehbar ist. Vor allem die Kodierung der Bedingungen in Formeln (und Klauseln) ist dann entscheidend. Alle Abgaben werden auf Plagiate im Jahrgang und zu älteren Jahrgängen geprüft.

Nur Programme, die den Testkorpus bestehen nehmen am Laufzeitvergleich für den Buchpreis statt.

Woher weiß das Framework, ob eine Instanz sat/unsat ist? Unser Framework verwendet dazu den Dateinamen. Beginnt er mit "u" gibt es keine Lösung zum Puzzle.

Gibt es Tricks um Fehler im Programm zu finden?

- Schreiben Sie die AL-Formeln explizit auf, und verwenden Tools wie wolframalpha, python boolean library oder z3 um Ihre Kodierung in die Klauselform zu überprüfen.
- Debuggen Sie Ihr Programm mit einem lösbaren Puzzle und beobachten Sie dabei den Ausdruck `solver.findModel()`. Sobald die Rückgabe zu `null` (unsatisfiable) haben Sie die Stelle der den Widerspruch erzeugt gefunden. Wenn Sie die konkrete Lösung kennen, können Sie diese auch direkt abprüfen im Debugger abprüfen mit `solver.findModel(new VecInt(new int[]...))` als *Watch-Expression*.
- Versuchen Sie Ihr Programm einfach zu halten und den einzelnen Bedingungen unabhängig von einander abzuarbeiten.