Prinzipien der Wissensrepräsentation

Prof. Dr. B. Nebel, Dr. S. Wölfl, Dr. M. Ragni
R. Mattmüller
Sommersemester 2010

Universität Freiburg
Institut für Informatik

Projekt P3

Abgabe: Mittwoch, 21. Juli 2010

Abgabe per E-Mail an `mattmuel@informatik.uni-freiburg.de`

Dieses Projekt widmet sich strikten Vererbungsnetzen. Es umfasst zwei Teile:

- Im ersten Teil wird ein Programm zum Schlussfolgern in einfachen Vererbungsnetzen mit negierten Konzepten implementiert.
- Im zweiten Teil wird dieses Programm auf Vererbungsnetze mit konjunktiven Konzepten erweitert.

Hinweis: Bei den Projekten geht es um die Implementierung, nicht so sehr um theoretische Aspekte. Wenn Sie an bestimmten Stellen nicht genau wissen, wie die Algorithmen und Definitionen aus der Vorlesung in die Praxis umgesetzt werden, können Sie daher gerne Fragen stellen – entweder per E-Mail oder in der Übungsgruppe.
Projekte können in C, C++, Java oder Python bearbeitet werden. Andere Programmiersprachen sind nach Absprache möglich; in diesem Fall bitte vor Bearbeitung des Projekts bei uns melden.
Die eingereichten Programme müssen die geforderten Ein- und Ausgabeformate verwenden, einige Tests bestehen und **ausreichend kommentiert** sein. Programme, die diesen Anforderungen nicht genügen, werden nicht akzeptiert, aber es besteht die Möglichkeit, innerhalb der Abgabefrist nachzubessern. Daher bitten wir darum, frühzeitig abzugeben, um ausreichend Zeit für Nachbesserungen zu haben.

Aufgabe P3.1 (Vererbungsnetze mit Negation, 1 Punkt)

Implementieren Sie ein Programm zum Schlussfolgern in einfachen Vererbungsnetzen mit Negation, aber ohne Konjunktion. Ihr Programm sollte das in der Vorlesung vorgestellte graphenbasierte Verfahren verwenden.

Eingabe

Das zu implementierende Programm soll wie folgt aufgerufen werden:[1]

```
# ./inheritance-simple <knowledge base> <query>
```

Dabei ist `<knowledge base>` der Name einer Datei, die die Wissensbasis enthält, und `<query>` die Anfrage an die Wissensbasis, eine **isa**-Aussage.[2] Jede Zeile der Wissensbasisdatei sollte eine **isa**-Aussage beinhalten oder eine Kommentarzeile sein, d. h. komplett aus Leerraum bestehen oder als erstes nicht-leeres Zeichen ein `#` enthalten. Kommentarzeilen werden ignoriert.
Eine **isa**-Aussage wird wie folgt notiert: `concept1 isa concept2`.
Dabei sind `concept1` und `concept2` Konzeptbeschreibungen in einer der folgenden beiden Formen:

[1] Passen Sie das Kommando entsprechend an, wenn Sie eine interpretierte Programmiersprache wie Java verwenden.

[2] Die Aussage muss üblicherweise in Anführungszeichen eingeschlossen werden, da sie Leerzeichen enthält, aber von der Shell als einzelnes Argument betrachtet werden soll.

- *Atomare Konzepte* (Konzeptnamen) sind Wörter bestehend aus Kleinbuchstaben, Ziffern und Bindestrichen, beginnend mit einem Kleinbuchstaben. Nicht erlaubt sind die Schlüsselwörter `isa`, `not` und `and`.
- *Negierte Konzepte* werden als `not concept` notiert, wobei `concept` eine Konzeptbeschreibung ist.

Ausgabe

Je nachdem, ob die Anfrage aus der Wissensbasis folgt, sollte die Ausgabe entweder `Yes.` oder `No.` lauten. Ansonsten sollten keine Ausgaben gemacht werden.

Aufgabe P3.2 (Negation und Konjunktion, 1,5 Punkte)

Erweitern Sie das Programm aus dem vorherigen Teil, so dass es auch konjunktive Konzepte `concept1 and concept2 and ... and conceptk` akzeptiert.
Konzeptbeschreibungen dürfen von Klammern umschlossen sein, damit Mehrdeutigkeiten durch Kombinationen von `and` und `not` aufgelöst werden können. Sind keine Klammern gesetzt, sollte `not` stärker binden als `and`, so dass die Konzeptbeschreibung `not a and b` äquivalent zu der Form `(not a) and b` ist, nicht aber zu `not (a and b)`.
Verwenden Sie für diesen Aufgabenteil einen SAT-Solver Ihrer Wahl[3], um die Anfrage zu beantworten. Ihr Programm sollte `inheritance` heißen und abgesehen vom verallgemeinerten Eingabeformat genauso verwendet werden und dieselben Ausgaben erzeugen wie das Programm aus dem vorigen Teil.

MiniSat

MiniSat wird mit zwei Argumenten `<input file>` und `<output file>` aufgerufen, wobei `<input file>` der Name einer wie folgt aufgebauten Datei ist:

- Die erste Zeile lautet `p cnf` M N, wobei M die Variablenzahl und N die Klauselzahl der Formel ist.
- Die weiteren Zeilen beschreiben Klauseln in der Form l_1 l_2 `...` l_k `0`, wobei die Literale l_i als Zahlen im Bereich $\{1, 2, \ldots, M\}$ (für positive Literale X_k) oder im Bereich $\{1, 2, \ldots, M\}$ (für negative Literale $\neg X_k$) angegeben werden.
- Keine Klausel darf tautologisch sein, d. h. keine Zeile darf sowohl k als auch $-k$ enthalten.

Beispiel einer Formel mit zwei Variablen und drei Klauseln:

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

Diese Eingabe repräsentiert die Formel $(X_1 \vee X_2) \wedge (\neg X_1 \vee \neg X_2) \wedge \neg X_1$.
Das Argument `<output file>` ist der Name einer Ausgabedatei. Die erste Zeile von `<output file>` nach einem Aufruf von MiniSat ist entweder `SAT` oder `UNSAT`, je nachdem, ob die Formel erfüllbar ist oder nicht.
Wer das Ergebnis `SAT` nachprüfen möchte, findet in `<output file>` unter der Zeile `SAT` eine erfüllende Belegung der Formel. In unserem Beispiel ist dies die Belegung $\{X_1 \mapsto 0, X_2 \mapsto 1\}$, die entsprechend dem Eingabeformat als `-1 2 0` notiert wird.
Beispieleingaben sind auf der Übungsseite zur Vorlesung verfügbar.

[3] Etwa MiniSat (Dokumentation, Sourcen und Binaries unter `http://minisat.se/`).