**Wissensrepräsentation**

Nebel/Rintanen/Wölfl — Universität Freiburg
WS 2004/2005 — Institut für Informatik

# Projekt P1

**Abgabe: Freitag, 17. Dezember 2004**

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

Dieses Projekt widmet sich dem Thema propositionale Default-Logik. Es umfasst drei Teile:

- Im ersten Teil wird ein System zum Schlussfolgern in Aussagenlogik entwickelt, das auf dem SAT-Solver *zChaff* aufbaut.
- Im zweiten und dritten Teil wird dieses System benutzt, um Erweiterungen von propositionalen Default-Theorien zu bestimmen und leichtgläubige und skeptische Schlussfolgerungen in Default-Logik durchzuführen.

*Hinweis:* Bei den Projektaufgaben geht es um die Implementierung, nicht so sehr um die theoretischen 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.

## Aufgabe P1.1 (Schließen in Aussagenlogik, 1,5 Punkte)

Implementieren Sie ein Programm zum Schließen in Aussagenlogik.
Das Programm soll für eine gegebene Wissensbasis $\Theta$ und Formel $\varphi$ ermitteln, ob $\Theta \models \varphi$ gilt oder nicht. Verwenden Sie dabei das Programm *zChaff* zum Testen der Erfüllbarkeit von aussagenlogischen Formeln in konjunktiver Normalform (siehe nächster Abschnitt).
Bekanntlich gilt $\Theta \models \varphi$ genau dann, wenn $(\bigwedge_{\vartheta \in \Theta} \vartheta) \rightarrow \varphi$ eine Tautologie ist, d. h. genau dann, wenn $(\bigwedge_{\vartheta \in \Theta} \vartheta) \wedge \neg\varphi$ *nicht* erfüllbar ist. Ihre Aufgabe besteht also im Wesentlichen darin, die Formeln in der Wissensbasis sowie die Formel $\neg\varphi$ in konjunktive Normalform zu übersetzen, die resultierende KNF-Formel für $(\bigwedge_{\vartheta \in \Theta} \vartheta) \wedge \neg\varphi$ in eine Datei zu schreiben und *zChaff* mit dieser Eingabe aufzurufen.

### zChaff

Das Programm *zChaff* kann von der Vorlesungsseite im Bereich „Übungen“ heruntergeladen werden. Es wird als `./zchaff <input file>` aufgerufen, wobei `<input file>` der Name einer Datei ist, die wie folgt aufgebaut ist:

- Die erste Zeile lautet
  `p cnf` $M$ $N$,
  wobei $M$ die Variablenzahl $N$ die Klauselzahl der Formel ist.
- Jede weitere Zeile beschreibt je eine Klausel in der Form
  $l_1\ l_2\ \ldots l_k$ `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 $(X_1 \vee X_2) \wedge (\neg X_1 \vee \neg X_2) \wedge \neg X_1$.
Ruft man *zChaff* mit dieser Eingabe auf, so wird ermittelt, dass die Formel erfüllbar ist. Für uns relevant ist für uns immer nur die letzte Zeile der *zChaff*-Ausgabe, die entweder `RESULT: SAT` oder `RESULT: UNSAT` lautet, je nach dem, ob die Formel erfüllbar ist oder nicht.
Wer das Ergebnis `RESULT: SAT` nachprüfen möchte, findet in der Ausgabe hinter der Zeile `Instance Satisfiable` eine erfüllende Belegung der Formel. In unserem Beispiel ist dies die Belegung $\{X_1 \mapsto \mathbf{F}, X_2 \mapsto \mathbf{T}\}$, die entsprechend dem Eingabeformat als `-1 2` notiert wird.

### Eingabe

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

```
# ./reason <knowledge base> <formula>
```

Dabei ist `<knowledge base>` der Name einer Datei, die eine aussagenlogische Wissensbasis enthält und `<formula>` eine aussagenlogische Formel. Formeln $\varphi$ werden durch folgende Grammatik erzeugt:[2]

| | |
|---|---|
| $\varphi \longrightarrow$ `a` \| `b` \| ... \| `z` | für Aussagevariablen |
| $\varphi \longrightarrow$ `T` \| `F` | für Wahr ($\top$) und Falsch ($\bot$) |
| $\varphi \longrightarrow$ `(`$\varphi$ `+` $\varphi$`)` | für Disjunktionen |
| $\varphi \longrightarrow$ `(`$\varphi$ `*` $\varphi$`)` | für Konjunktionen |
| $\varphi \longrightarrow$ `(`$\varphi$ `->` $\varphi$`)` | für Implikationen |
| $\varphi \longrightarrow$ `~`$\varphi$ | für Negationen |

Es müssen also nur vollständig geklammerte Formeln unterstützt werden, und die Äquijunktion $\leftrightarrow$ kann ignoriert werden. Leerzeichen zwischen den verschiedenen Bestandteilen der Formel sollen erlaubt sein.
Wissensbasis-Dateien enthalten zwei Arten von Zeilen:

- Eine Formelzeile enthält eine aussagenlogische Formel gemäß obiger Syntax; diese ist damit Teil der Wissensbasis.
- Eine Kommentarzeile besteht nur aus Leerzeichen oder hat ein `#` als erstes nicht-leeres Zeichen.

[1] Passen Sie hier und im Folgenden die Kommandos entsprechend an, wenn Sie eine interpretierte Programmiersprache wie Java verwenden.
[2] Formeln, die auf der Kommandozeile übergeben werden, müssen in Anführungszeichen eingeschlossen werden, wenn sie Zeichen enthalten, die für die Shell eine besondere Bedeutung haben (`*`, `>` und Klammern).

### Ausgabe

Das Programm sollte entweder „`Yes.`“ oder „`No.`“ ausgeben, je nach dem, ob die übergebene Formel aus der Wissensbasis folgt oder nicht. Bei ungültiger Eingabe sollte eine Fehlermeldung ausgegeben werden. Weitere Ausgaben sollten nicht erzeugt werden, um die Überprüfung des Programms zu vereinfachen.

### Beispiel

Nehmen wir an, die Datei `socrates.kb` enthalte folgende Wissensbasis:

```
# Is Socrates mortal? We know that he is human (h)
# and that humans are mortal (m).
h
(h -> m)
```

Dann sollte der Aufruf
`./reason socrates.kb m`
die Ausgabe `Ja.` erzeugen, denn aus $h$ und $h \to m$ folgt $m$.

## Aufgabe P1.2 (Bestimmen von Erweiterungen, 2 Punkte)

Implementieren Sie ein Programm, dass alle Erweiterungen einer propositionalen Default-Theorie bestimmt.

### Eingabe

Das zu implementierende Programm soll wie folgt aufgerufen werden:
`# ./extensions <default theory>`
Dabei ist `<default theory>` der Name einer Datei, die eine Default-Theorie $\langle D, W\rangle$ beschreibt. Solche Dateien sind beliebige Abfolgen von Default-Regeln, Hintergrundwissen und Kommentaren nach folgender Syntax:

- Eine Default-Regel aus $D$ wird als Zeile
  `D` $\alpha$ $\beta$ $\gamma$
  notiert. Dabei ist $\alpha$ die Vorbedingung, $\beta$ die Konsistenzbedingung und $\gamma$ die Konklusion der Default-Regel. Die einzelnen Formeln werden wie bei der ersten Aufgabe angegeben.
- Eine Formel mit Hintergrundwissen aus $W$ wird als Zeile
  `W` $\alpha$
  notiert. Die Formel $\alpha$ wird wie bei der ersten Aufgabe angegeben.
- Eine Kommentarzeile besteht nur aus Leerzeichen oder hat ein `#` als erstes nicht-leeres Zeichen.

### Ausgabe

Das Programm sollte in jeder Zeile seiner Ausgabe je eine Erweiterung der Default-Theorie beschreiben. Ansonsten sollten außer bei einer fehlerhaften Eingabe keine weiteren Ausgaben gemacht werden.
Da Erweiterungen unendliche Mengen sind, können sie natürlich nicht explizit in der Ausgabe angegeben werden. Stattdessen sollte eine Erweiterung durch

die Nummern der Default-Regeln repräsentiert werden, die in ihr angewandt wurden. Wenn $\Delta$ also z. B. eine Erweiterung besitzt, in der die zweite, dritte und fünfte Default-Regel der Eingabedatei angewandt wurde (und sonst keine weitere Regel), dann sollte die Ausgabe die Zeile

```
2 3 5
```

enthalten.

### Beispiel

Nehmen wir an, die Datei `ex3_2.dt` enthalte folgende Default-Theorie (s. Aufgabe 3.2).

```
# Defaults:
D a e e
D h ~l ~l
D T a a
# Background knowledge:
W h
W (e -> l)
```

Dann sollte der Aufruf `./extensions ex3_2.dt` die folgende Ausgabe erzeugen:

```
1 3
2 3
```

Dies entspricht den zwei Erweiterungen, in denen die erste und dritte bzw. die zweite und dritte Default-Regel angewandt wurden (siehe Lösung zu Aufgabe 3.2).

## Aufgabe P1.3 (Schließen in Default-Logik, 0,5 Punkte)

Implementieren Sie ein Programm zum leichtgläubigen und skeptischen Schließen in propositionalen Default-Theorien.

### Eingabe

Das zu implementierende Programm soll wie folgt aufgerufen werden:

```
# ./defreason <default theory> <formula>
```

Dabei ist `<default theory>` wie in der vorherigen Aufgabe der Name einer Datei, die eine Default-Theorie beschreibt, und `<formula>` wie in der ersten Aufgabe eine propositionale Formel.

### Ausgabe

Das Programm soll genau zwei Ausgaben erzeugen. Die erste Zeile sollte

```
Formula is credulous conclusion: Yes.
```

oder

```
Formula is credulous conclusion: No.
```

lauten, je nach dem, ob die Formel leichtgläubig aus der Default-Theorie folgt oder nicht.
Analog gibt es für die zweite Zeile die Möglichkeiten:

```
Formula is skeptical conclusion: Yes.
Formula is skeptical conclusion: No.
```