

Antwortmengenprogrammierung

Eine Einführung

v4hn

upLUG Potsdamer Linux User Group

3. Juli 2011

1 Einführung

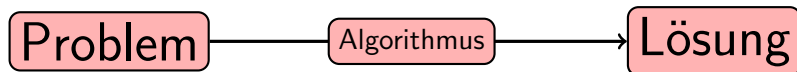
2 Modellierung

- SAT und ASP
- Regeltypen
- Grounding

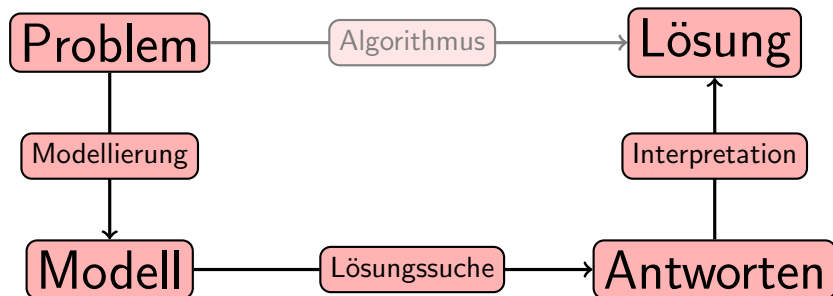
3 Anwendung

- Die Werkzeuge
- Beispiele

Traditioneller Ansatz



Deklarativer Ansatz



SAT - Erfüllbarkeit logischer Formeln

Formel

$$(a \vee b) \wedge (a \vee c)$$

erfüllende Belegungen

$$\{a \mapsto \text{wahr}, b \mapsto \text{falsch}, c \mapsto \text{falsch}\} \Rightarrow \{a\}$$

$$\{a \mapsto \text{falsch}, b \mapsto \text{wahr}, c \mapsto \text{wahr}\} \Rightarrow \{b, c\}$$

$$\{a \mapsto \text{wahr}, b \mapsto \text{wahr}, c \mapsto \text{falsch}\} \Rightarrow \{a, b\}$$

$$\{a \mapsto \text{wahr}, b \mapsto \text{falsch}, c \mapsto \text{wahr}\} \Rightarrow \{a, c\}$$

$$\{a \mapsto \text{wahr}, b \mapsto \text{wahr}, c \mapsto \text{wahr}\} \Rightarrow \{a, b, c\}$$

SAT - Erfüllbarkeit logischer Formeln

Formel

$$(a \vee b) \wedge (a \vee c)$$

erfüllende Belegungen

$$\{a \mapsto \text{wahr}, b \mapsto \text{falsch}, c \mapsto \text{falsch}\} \Rightarrow \{a\}$$

$$\{a \mapsto \text{falsch}, b \mapsto \text{wahr}, c \mapsto \text{wahr}\} \Rightarrow \{b, c\}$$

$$\{a \mapsto \text{wahr}, b \mapsto \text{wahr}, c \mapsto \text{falsch}\} \Rightarrow \{a, b\}$$

$$\{a \mapsto \text{wahr}, b \mapsto \text{falsch}, c \mapsto \text{wahr}\} \Rightarrow \{a, c\}$$

$$\{a \mapsto \text{wahr}, b \mapsto \text{wahr}, c \mapsto \text{wahr}\} \Rightarrow \{a, b, c\}$$

SAT - Erfüllbarkeit logischer Formeln

Formel

$$(a \vee b) \wedge (a \vee c)$$

erfüllende Belegungen

$$\{a \mapsto \text{wahr}, b \mapsto \text{falsch}, c \mapsto \text{falsch}\} \Rightarrow \{a\}$$

$$\{a \mapsto \text{falsch}, b \mapsto \text{wahr}, c \mapsto \text{wahr}\} \Rightarrow \{b, c\}$$

$$\{a \mapsto \text{wahr}, b \mapsto \text{wahr}, c \mapsto \text{falsch}\} \Rightarrow \{a, b\}$$

$$\{a \mapsto \text{wahr}, b \mapsto \text{falsch}, c \mapsto \text{wahr}\} \Rightarrow \{a, c\}$$

$$\{a \mapsto \text{wahr}, b \mapsto \text{wahr}, c \mapsto \text{wahr}\} \Rightarrow \{a, b, c\}$$

Antwortmengen

Hintergrund

Wir wollen nicht **alle** erfüllenden Belegungen,
sondern nur die, die auch begründet werden können!

Ansatz

- Jedes Element in einer AM muss eine Begründung haben.
- Zwei Elemente in einer AM dürfen sich nicht gegenseitig begründen.

Beispiele

arbeiten \leftarrow *zwangsarbeit*.

freizeit \leftarrow *wetter_schlecht*.

arbeiten \leftarrow *wetter_schlecht*.

wetter_schlecht \leftarrow *freizeit*.

Ein kleines Programm

voegel.lp

```
vogel(tweety).  
fliegt(tweety).  
pinguin(tux).  
vogel(X) :- pinguin(X).  
:- pinguin(X), fliegt(X), not hat_jetpack(X).  
hat_jetpack(X) :- pinguin(X), fliegt(X).  
0 {pinguin(X)} 1 :- vogel(X).
```

Fakten

```
vogel(tweety).  
fliegt(tweety).  
pinguin(tux).
```

Es wird festgelegt, dass tweety ein Vogel und tux ein Pinguin ist und dass tweety fliegen kann.

Normale Regeln

```
vogel(X) :- pinguin(X).  
hat_jetpack(X) :- pinguin(X), fliegt(X).
```

Wenn X ein Pinguin ist, dann ist X auch ein Vogel.

Wenn X ein fliegender Pinguin ist, dann hat X ein Jetpack.

Widersprüche

```
:- pinguin(X), fliegt(X), not hat_jetpack(X).
```

Es ist ein Widerspruch, wenn X ein fliegender Pinguin ohne Jetpack ist.

Fakten

```
vogel(tweety).  
fliegt(tweety).  
pinguin(tux).
```

Es wird festgelegt, dass tweety ein Vogel und tux ein Pinguin ist und dass tweety fliegen kann.

Normale Regeln

```
vogel(X) :- pinguin(X).  
hat_jetpack(X) :- pinguin(X), fliegt(X).
```

Wenn X ein Pinguin ist, dann ist X auch ein Vogel.

Wenn X ein fliegender Pinguin ist, dann hat X ein Jetpack.

Widersprüche

```
:- pinguin(X), fliegt(X), not hat_jetpack(X).
```

Es ist ein Widerspruch, wenn X ein fliegender Pinguin ohne Jetpack ist.

Fakten

```
vogel(tweety).  
fliegt(tweety).  
pinguin(tux).
```

Es wird festgelegt, dass tweety ein Vogel und tux ein Pinguin ist und dass tweety fliegen kann.

Normale Regeln

```
vogel(X) :- pinguin(X).  
hat_jetpack(X) :- pinguin(X), fliegt(X).
```

Wenn X ein Pinguin ist, dann ist X auch ein Vogel.

Wenn X ein fliegender Pinguin ist, dann hat X ein Jetpack.

Widersprüche

```
:- pinguin(X), fliegt(X), not hat_jetpack(X).
```

Es ist ein Widerspruch, wenn X ein fliegender Pinguin ohne Jetpack ist.

Auswahlregeln

```
0 {pinguin(X)} 1 :- vogel(X).
```

Wenn X ein Vogel ist, dann müssen minimal 0 und maximal 1 der folgenden Aussagen wahr sein: X ist ein Pinguin.

Gewichtete Regeln

```
brauche_lasttier :-  
    20 [ nimm_mit(schwert)=5, nimm_mit(ruestung)=10,  
        nimm_mit(nahrung)=5, nimm_mit(truhe)=20 ].
```

Wenn das Gesamtgewicht mindestens 20 beträgt, dann wird ein Lasttier zum Tragen benötigt.

Auswahlregeln

```
0 {pinguin(X)} 1 :- vogel(X).
```

Wenn X ein Vogel ist, dann müssen minimal 0 und maximal 1 der folgenden Aussagen wahr sein: X ist ein Pinguin.

Gewichtete Regeln

```
brauche_lasttier :-  
    20 [ nimm_mit(schwert)=5, nimm_mit(ruestung)=10,  
        nimm_mit(nahrung)=5, nimm_mit(truhe)=20 ].
```

Wenn das Gesamtgewicht mindestens 20 beträgt, dann wird ein Lasttier zum Tragen benötigt.

... und ein paar Extras

Minimierung

```
#minimize [ laufe(berlin, muenchen)= 588,  
            laufe(muenchen, hannover)=636,  
            laufe(berlin, hannover)=282, ... ].
```

Gib nur die Antwortmenge zurück, in der am wenigsten gelaufen wird.

Abkürzungen

```
alter(0..130).  $\implies$   
  alter(0).alter(1). ...
```

```
farbe(gruen;gelb;rot).  $\implies$   
  farbe(gruen). farbe(gelb). farbe(rot).
```

```
hat_beruf(X, B) : beruf(B)  $\implies$   
  hat_beruf(X, baecker), hat_beruf(X, maurer), ...
```

... und ein paar Extras

Minimierung

```
#minimize [ laufe(berlin, muenchen)= 588,  
            laufe(muenchen, hannover)=636,  
            laufe(berlin, hannover)=282, ... ].
```

Gib nur die Antwortmenge zurück, in der am wenigsten gelaufen wird.

Abkürzungen

```
alter(0..130).  $\implies$   
  alter(0).alter(1). ...
```

```
farbe(gruen;gelb;rot).  $\implies$   
  farbe(gruen). farbe(gelb). farbe(rot).
```

```
hat_beruf(X, B) : beruf(B)  $\implies$   
  hat_beruf(X, baecker), hat_beruf(X, maurer), ...
```

Grounding

Prinzip

Das gegebene Programm enthält noch Variablen. Um es effizient lösen zu können, müssen diese Variablen mit allen möglichen Werten belegt werden.

```
vogel(tweety).fliegt(tweety).pinguin(tux).
vogel(X) :- pinguin(X).
:- pinguin(X), fliegt(X), not hat_jetpack(X).
hat_jetpack(X) :- pinguin(X), fliegt(X).
0 {pinguin(X)} 1 :- vogel(X).
```

voegel.ground

```
vogel(tweety). fliegt(tweety). pinguin(tux).
vogel(tux).
:- pinguin(tweety), not hat_jetpack(tweety).
hat_jetpack(tweety) :- pinguin(tweety).
0{pinguin(tweety)}1.
```

gringo & clasp

gringo

gringo berechnet das Grounding des eingegebenen Programmes und gibt das Ergebnis auf STDOUT aus.

- gringo2 ist ziemlich eingeschränkt
- gringo3 wesentlich freier, kann aber in Endlosschleifen laufen...

clasp

clasp liest ein gegroundetes Programm ein und berechnet Antwortmengen.

usecase

```
$ gringo programm.lp instanz.lp | clasp
```

Sudoku

sudoku.lp

```
pos(0..8). input(1..9). block(0..2).
1{in(X, Y, Z): input(Z)}1 :- pos(X), pos(Y).
1{in(X,Y, Z): pos(Y)}1 :- pos(X), input(Z).
1{in(X,Y, Z): pos(X)}1 :- pos(Y), input(Z).
1{in(X,Y, Z): pos(X) : X/3 == Xb : pos(Y) : Y/3 == Yb}1
```

sudoku_instanz.lp

```
in(1,0, 3). in(3,1, 1). in(4,1, 9). in(5,1, 5).
in(1,2, 9). in(2,2, 8). in(7,2, 6). in(0,3, 8).
in(4,3, 6). in(0,4, 4). in(5,4, 3). in(8,4, 1).
in(4,5, 2). in(1,6, 6). in(6,6, 2). in(7,6, 8).
in(3,7, 4). in(4,7, 1). in(5,7, 9). in(8,7, 5).
in(7,8, 7).
```

The End

Danke für die Aufmerksamkeit :-)

Weitere Informationen:

<http://potassco.sourceforge.net/>

<http://cs.uni-potsdam.de/wv/>

Kontakt:

Email: <me@v4hn.de>

Jabber: V4hn@jabber.ccc.de

Mord in der Familie

Wer ist der Mörder?

Eine Familie besteht aus Vater, Mutter, Tochter und Sohn.
Ein Mitglied der Familie hat ein zweites ermordet. Ein weiteres Mitglied war Helfer und das letzte Mitglied hat das Verbrechen beobachtet.

- Der Helfer und der Zeuge sind verschiedengeschlechtlich.
- Der Älteste und der Zeuge sind verschiedengeschlechtlich.
- Der Jüngste und das Opfer sind verschiedengeschlechtlich.
- Der Helfer ist älter als das Opfer.
- Der Vater ist der Älteste.
- Der Mörder ist nicht der Jüngste.