Mehr Prolog

Prolog bietet mehr an, z.B.:

- Prädikate zum Testen und Manipulieren der Struktur der Terme;
- Mehr meta-logische Prädikate z.B. zum Testen des Zustands der Ableitung;
- Mehr extra-logische Prädikate, die Seiteneffekte bei ihrer "Ableitung" haben, z.B für Ein- und Ausgabe oder für die Schnittstelle zum Betriebssystem.
- Es gibt Prolog-Erweiterungen, z.B. für Constraint-Programmierung...

6 Constraint-Programmierung

- Alle in einem rein logischen Programm manipulierten Objekte sind syntaktische Konstruktionen (Terme), denen keine Semantik zugewiesen wird. Man sagt, dass die Funktor-Symbolen nicht-interpretiert (d.h die dargestellten Objekte nicht-interpretierte Strukturen) sind:
  - ▷ Das Ziel `X=2+3` bewirkt nur die Bindung von `X` an den Term `2+3`, weil das Funktionssymbol `+` nicht interpretiert wird.

Constraints

- Zwei Objekte sind in LP nur dann gleich, wenn sie syntaktisch gleich sind.
- Idee: erweitere die rein syntaktische Gleichheit in LP zur Gleichung, die zu lösen ist:
  - ▷ Das Ziel `X+Y=8,X-Y=2` erhält in einer CP-Sprache die Antwort `X=5,Y=3`.
- Gleichungen spezifizieren implizit Relationen zwischen semantischen Objekten. Im Allgemeinen heißen Relationen zwischen semantischen Objekten Constraints.

Constraint-Programmierung als Erweiterung der logischen Programmierung

- Die Constraints werden als syntaktisch ausgezeichnete Prädikate dargestellt, und statt mittels SLDNF-Resolution durch spezielle Algortithmen über bestimmte Wertebereiche mit Hilfe eines Constraint-Löser gelöst.
- LP ist CP, wobei das einzige Constraint die syntaktische Gleichheit zwischen Termen ist, und der Unifikationsalgorithm zur Lösung solcher Constraints benutzt wird.

Constraint-Löser

- Programmierung mit Constraints ist in beliebiegen Programmiersprachen möglich, vorausgesetzt dass ein Constraint-Löser, evt. als eine Erweiterungs-Bibliothek zur Verfügung gestellt wird, z.B. für Java, C++, Prolog, Lisp.
- Meist werden LP-Sprachen um Constraints erweitert $\longrightarrow$ Constraint-Logikprogrammierung, z.B. Eclipse-, Sicstus-, SWI-Prolog, Mozart/Oz, etc.
- Effektive Constraint-Löser gibt für verschiedene Constraint-Arten, z.B.:
  - ▷ Logische Formeln über boolsche Variable
  - ▷ Interval-Constraints über endliche Bereiche
  - ▷ Lineare Gleichungssysteme über reele Zahlen

Constraint-Löser

Von einem Constraint-Löser zu erfüllenden Berechnungsdienste:

- Konsistenztest(Erfüllbarkeittest) (*satisfiability/consistency test*): sind die Constraints erfüllbar? Ein Constraint-Löser ist vollständig, wenn er die Erfüllbarkeit jeder beliebiegen Menge von Constraints entscheiden kann.
- Vereinfachung: Die Constraints in eine einfachere Normalform darstellen können. Zur effizienten Vereinfachung soll der Löser inkrementell sein: Vereinfachung der Constraints zusammen mit einem neu hinzukommendes Constraint ohne die Vereinfachung der bisherigen Constraints.
- Determination: Erkennen, wenn eine Variable nur noch einen bestimmten Wert haben kann. (z.B. $X \geq 1$, $X \leq 1 \Rightarrow X = 1$.)

Vorteile der Constraint-Logikprogrammierung

- Zusätzlich zur erhöhten Deklarativität kann CP die Effizienz der LP-Sprachen erhöhen.
- Constraints können benutzt werden, um den "Nicht-Determinismus" der LP-Suche nach Lösungen einzuschränken, indem man Teilbäume von der Suche ausschliesst, die keine Lösung der Constraints enthalten können (durch Konsistenzteste).

$\Longrightarrow$ CP wird eingesetzt, um kombinatorische Probleme zu lösen, die meist exponentielle Komplexität haben.

6.1 Das Berechnungsmodell der Constraint-Programmierung

Syntax einer CP-Sprache:

Atome:	A, B	$::=$	$p(t_1, \ldots, t_n)$
Constraints:	C, D	$::=$	$c(t_1, \ldots, t_n) \mid C \wedge D$
Ziele:	G, H	$::=$	$\top \mid \bot \mid A \mid C \mid G \wedge H$
Klauseln:	K	$::=$	$A \leftarrow G$
Programme:	P	$::=$	$\{K_1, \ldots, K_m\}$

Berechnungszustände

- Ein Zustand ist ein Paar $< G, C >$, wobei G ein Ziel und C ein Constraint.
- G heißt Zielspeicher (die noch zu lösende Ziele), C heißt Constraintspeicher (die bereits aufgetretene Constraints).
- Ein Anfangszustand ist ein Zustand der Form $< G, true >$.
- Ein Zustand heißt erfolgreicher Endzustand, falls er von der Form $< \top, C >$ ist.
- Ein Zustand heißt erfolgloser Endzustand, falls er von der Form $< G, false >$ ist.

CP-Kalkül

Entfalten	$$\frac{(B \leftarrow H) \in P \qquad (B = A) \wedge C \text{ ist erfüllbar}}{< A \wedge G, C > \mapsto_{Entfalten} < H \wedge G, (B = A) \wedge C >}$$
Scheitern	$$\frac{\text{Es gibt keine Klausel } (B \leftarrow H) \in P, \text{ so dass } (B = A) \wedge C \text{ erfüllbar ist}}{< A \wedge G, C > \mapsto_{Scheitern} < \bot, false >}$$
Vereinfachen	$$\frac{C \wedge D_1 \equiv D_2}{< C \wedge G, D_1 > \mapsto_{Vereinfachen} < G, D_2 >}$$

Die Antwort einer CP-Berechnung

- Die Antwort einer Berechnung, die einen erfolgreichen Endzustand $< \top, C >$ erreicht, ist C.
- Eine Antwort heißt bestimmt (*definite*), wenn er eine Gleichung `X=Konstante` für jede Variable in der Anfrage enthält.
  - ▷ Z.B. `X+Y=10,X-Y=6` liefert die Antwort `X=8,Y=2`.
- Im Allgemeinen kann eine Antwort unbestimmt (*indefinite*) sein, d.h. eine undenliche Menge von Lösungen repräsentieren.
  - ▷ Z.B. liefert `X`$\leq$`Y,Y`$\leq$`Z,Z`$\leq$`X` die Antwort `X=Y=Z`.

6.2 Lineare arithmetische Constraints

Arithmetische Ausdrücke:

t ::= Zahl | Variable | $t_1 \odot t_2$ mit $\odot \in \{+, -, *, /\}$

Linearität:

- Höchstens ein Term einer Multiplikation enthält eine Variable.
- Der Teiler in einer Division enthält keine Variable.

Arithmetische Constraints:

C ::= true | false | $C \wedge C$ | $t_1 \ \mathcal{R} \ t_2$ mit $\mathcal{R} \in \{<, \leq, =, >, \geq, \neq\}$

Beispiel: Menu-Berechner

```
appetiser(radishes,1). appetiser(pasta,6).
meat(beef,5). meat(pork,7).
fish(sole,2). fish(tuna,4).
dessert(fruit,2). dessert(icecream,6).

main(M,I) :- meat(M,I).
main(M,I) :- fish(M,I).

lightmeal(A,M,D) :-
   I>0, J>0, K>0,
   I+J+K <= 10,
   appetiser(A,I), main(M,J), dessert(D,K).
```

Beispiel: erfolgreiche Berechnung

$$
\begin{array}{ll}
 & < \texttt{lightmeal(A,M,D);true} > \\
\mapsto & < \texttt{I>0,J>0,K>0,I+J+K<=10,appetiser(A,I),main(M,J),dessert(D,K);true} > \\
\mapsto^* & < \texttt{appetiser(A,I),main(M,J),dessert(D,K);I>0,J>0,K>0,I+J+K<=10} > \\
\mapsto & < \texttt{main(M,J),dessert(D,K);A=radishes,I=1,J>0,K>0,1+J+K<=10} >, \\
\mapsto & < \texttt{meat(M1,I1),dessert(D,K);} \\
 & \qquad\qquad\qquad\qquad \texttt{M=M1,J=I1,A=radishes,I=1,J>0,K>0,1+J+K<=10} > \\
\mapsto & < \texttt{dessert(D,K);M=beef,J=5,M1=beef,I1=5,A=radishes,I=1,K>0,1+5+K<=10} > \\
\mapsto & < \top\texttt{;D=fruit,K=2,M=beef,J=5,M1=beef,I1=5,A=radishes,I=1} >
\end{array}
$$

Antwort: `A=radishes,M=beef,D=fruit`.

Beispiel: gescheiterte Berechnung

$<$ lightmeal(A,M,D);true $>$

$\mapsto$ $<$ I>0,J>0,K>0,I+J+K<=10,appetiser(A,I),main(M,J),dessert(D,K);true $>$

$\mapsto$ $<$ appetiser(A,I),main(M,J),dessert(D,K);I>0,J>0,K>0,I+J+K<=10 $>$

$\mapsto$ $<$ main(M,J),dessert(D,K);A=pasta,I=6,J>0,K>0,6+J+K<=10 $>$,

$\mapsto$ $<$ meat(M1,I1),dessert(D,K);
M=M1,J=I1,A=pasta,I=6,J>0,K>0,6+J+K<=10 $>$

$\mapsto$ $<$ dessert(D,K);M=beef,J=5,M1=beef,I1=5,A=pasta,I=6,K>0,6+5+K<=10 $>$

$\mapsto$ $< \perp, false >$

Beispiel: Finanzberater...

```
darlehen(Darlehenshoehe,Monate,Zinssatz,Rate,Restschuld):-
    Monate=0,Darlehenshoehe=Restschuld

darlehen(Darlehen,Monate,Zinssatz,Rate,Restschuld):-
    Monate>0,
    Monate1=Monate-1,
    Darlehen1=Darlehen+Darlehen*Zinssatz-Rate,
    darlehen(Darlehen1,Monate1,Zinssatz,Rate,Restschuld).
```

Beispiel: Finanzberater...

Sicstus-Prolog Syntax

```
darlehen(Darlehenshoehe,Monate,Zinssatz,Rate,Restschuld):-
    {Monate=0},Darlehenshoehe=Restschuld.

darlehen(Darlehens,Monate,Zinssatz,Rate,Restschuld):-
    {Monate>0,
    Monate1=Monate-1,
    Darlehens1=Darlehens+Darlehens*Zinssatz-Rate},
    darlehen(Darlehens1,Monate1,Zinssatz,Rate,Restschuld).
```

Beispiel: Finanzberater...

- Welcher Restschuld bleibt nach 30 Jahren für ein Darlehen von €200000 bei einer Rate von €1000 und einem monatlichen Zinssatz von 0.4%.

```
?- darlehen(200000,360,0.004,1000,S).
S = 39570.50372439585
```

- Wie hoch ist die Rate, um das Darlehen in 30 Jahren vollständig zu zahlen?

```
 ?- darlehen(200000,360,0.004,X,0.0).
X = 1049.3307086826705
```

- Wieviele Monate muss man zahlen, um die Schulden zu bezahlen bei einer monatlicher Rate von €1000?

```
darlehen(200000,X,0.004,1000,S),{S<0.0}.
S = -836.0650151022006, X = 404.0
```

Beispiel: Finanzberater...

- Welches ist das Verhältnis zwischen Darlehenshöhe und Rate, wenn man in 30 Jahren die Schulden komplett Zahlen möchte?

```
darlehen(D,360,0.004,R,0.0).
{R=0.005246653543413345*D}
```

- Bei welchem Zinssatz bezahlt man die Schuld in genau 20 Jahren?

```
darlehen(200000,360,Z,1000,0.0).
```

...kann nicht von einem linearer Gleichungslöser behandelt werden. Grund: die im zweiten Schritt aufgestellte Gleichung ist nicht mehr linear:

$$D_1 = D + D \cdot Z - R$$
$$D_2 = D_1 + D_1 \cdot Z - R$$
$$\vdots$$

6.3 Constraints über endliche Wertebereiche (FD-Constraints)

Constraints:

$$\begin{aligned} \text{C} \quad ::= \quad & \text{true} \mid \text{false} \mid C \wedge C \mid X \text{ in } n..m \\ & \mid X \text{ in } [k_1, \ldots, k_n] \mid X + Y = Z \mid X \ \mathcal{R} \ Y \end{aligned}$$

mit $n, m, k_1, \ldots, k_l \in \mathbb{N}$

$\mathcal{R} \in \{<, \leq, =, >, \geq, \neq\}$

X, Y, Z Variablen oder ganze Zahlen

Beispiel

FD-Constraints in Sicstus-Prolog:

```
| ?- X in 1..5, T in 3..13, X+Y #= T.

X in 1..5,
T in 3..13,
Y in -2..12 ?

yes
```

Constraint Satisfaction Probleme

Ein CSP ist definiert durch:

- Eine Menge von Variablen $X_1 \in D_1, \ldots, X_n \in D_n$, mit $D_1, \ldots, D_n$ endlichen Wertebereichen
- Eine Menge von Constraints die von den Variablen zu erfüllen sind.

Eine Lösung eines CSP ist eine Variablenbelegung, die die Constraints erfüllt.

Lösung eines CSP mit CP(FD)

Der Ansatz zur Lösung eines CSP mit Constraints über endliche Bereiche besteht typischerweise aus drei Komponenten:

1. Deklaration der Wertebereiche der Variablen.
2. Aufsetzen der Constraints
3. Suche einer Lösung (Backtracking, Branch-and-Bound)

Beispiel: CSP Problem

- Finde eine Belegung, so dass die Folgende Berechnung wahr ist!

$$\begin{array}{cccccc} & & S & E & N & D \\ & & M & O & R & E \\ \hline = & M & O & N & E & Y \end{array}$$

- Suchraum hat die Größe 10^8. Exhaustive Suche ist praktisch nicht einsetzbar.

- Ein Mensch würde das Problem lösen, indem er Constraints dynamisch ableitet. Z.B. muss M gleich 1 sein. Es folgt, dass S gleich 9 oder 8 ist. U.s.w...

Beispiel:

1. Ansatz (Sicstus-Prolog):

```
solve(S,E,N,D,M,O,R,Y) :-
        domain([S,E,N,D,M,O,R,Y], 0, 9),
        S#>0, M#>0,
        all_different([S,E,N,D,M,O,R,Y]),
                        1000*S + 100*E + 10*N + D
        +               1000*M + 100*O + 10*R + E
        #= 10000*M + 1000*O + 100*N + 10*E + Y.
```

```
| ?- solve(S,E,N,D,M,O,R,Y).
M = 1, O = 0, S = 9, E in 4..7, N in 5..8,
D in 2..8, R in 2..8, Y in 2..8
```

Beispiel:

2. Lösung (Sicstus-Prolog) mit Suche: Zum Enumerieren benutzt man das Prädikat (hier `labeling`), das Variablen alle Werte aus ihren Variablenbereichen nach einer vorgegebenen Strategie zuordnet.

```
solve(S,E,N,D,M,O,R,Y) :-
        domain([S,E,N,D,M,O,R,Y], 0, 9),
        S#>0, M#>0,
        all_different([S,E,N,D,M,O,R,Y]),
                        1000*S + 100*E + 10*N + D
        +               1000*M + 100*O + 10*R + E
        #= 10000*M + 1000*O + 100*N + 10*E + Y,
        labeling([], [S,E,N,D,M,O,R,Y]).
```

```
| ?- solve(S,E,N,D,M,O,R,Y).
D = 7, E = 5, M = 1, N = 6, O = 0, R = 8, S = 9, Y = 2
```