CLP - Beispiele und Anwendungen

Logikprogrammierung Constraint lösen

CLP

Mathematik Operations Research

- Logikprogrammierung: deklarativ, relationale Form, nicht determiniert (Suche), Unifikation.
  Probleme: "generate and test" ineffizient für komplexe Probleme
  Datenstrukturen: Anwendungsbereich versus Terme
  Inferenz: Unification und Resolution versus Spezialmethoden
- Constraint lösen: KI, Konsistenztechniken + Heuristic
- Operations Research: spezialisiert vs. allgemein, statisches Problem vs. inkrementelle dynamische Probleme
- Mathematik: wie OR

Constraints

- vollständige Verfahren
  - (Pseudo-)Bool: Unifikation, OR-Techniken
  - Rational: Simplex, Innere Punkte Methode, Intervalltechniken
  - ...
- unvollständige Verfahren
  - endliche Bereiche (FD): Konsistenztechniken
  - Bool:
  - Listen:
  - ...

Boolesche Constraints

Logische Schaltkreise

Halbaddierer:

`half_adder`(X,Y,X $\oplus$ Y,X $\wedge$ Y).

Addierer:

```
full_adder(X,Y,Cin,S,C1 ∨ C2) :-
        half_adder(X,Y,S1,C1),
        half_adder(S1,Cin,S,C2).
```

Anfrage und Lösung:

`?-full_adder`(X,Y,Cin,S,Cout).

S = X $\oplus$ Y $\oplus$ Cin,
Cout = X $\wedge$ Y $\oplus$ X $\wedge$ Cin $\oplus$ Y $\wedge$ Cin `yes.`

`?-full_adder`(X,Y,Cin,1,0).
X = A
Y = A $\wedge$ B $\oplus$ B
Cin = A $\wedge$ B $\oplus$ A $\oplus$ B $\oplus$ 1 `yes.`

- Aufbau und Test komplexer Schaltungen
- Verifikation durch Vergleich mit Spezifikation
- Fehlersuche

Pseudo-Boolesche Constraints

0-1 Optimierungsproblem: Zielfunktion und 0-1 Nebenbedingung

Beispiel: Radiobau

- 3 Röhrentypen (T_1, T_2, T_3); genau ein Typ
- Gehäuse entweder aus Holz (H) oder Plastik (P).
  Falls Plastik Platzmangel: Röhre T_2 und nicht Transformator (F) sondern Netzteil (S).
- T_1 arbeitet nur mit F.
- T_2 und T_3 arbeiten nur mit S.

Die Preise der Komponenten sind;

T_1	28 GE	T_2	30 GE	T_3	31 GE
F	25 GE	S	23 GE		
H	9 GE	P	6 GE		

Die anderen notwendigen Komponenten des Radios kosten

$$T_1 \text{ dann } 27 \text{ GE} \quad T_2 \text{ dann } 28 \text{ GE} \quad T_3 \text{ dann } 25 \text{ GE}$$

Zusammenbau: 10 GE
Preis: P dann 110. H dann 105. Problemstellung :

Welches Modell soll gebaut werden, um den Profit zu maximieren ?

```
assemble(T1, T2, T3, F, S, H, P) :-
                  T1 + T2 + T3 = 1,
                  H + P = 1,
                  F + S = 1,
                  P ≤ T2 * S,        % P = 1 impliziert T2 = S = 1
                  T1 ≤ F,            % T1 = 1 impliziert F = 1
                  T2 ≤ S,            % T2 = 1 impliziert S = 1
                  T3 ≤ S.            % T3 = 1 impliziert S = 1

maximal-profit(T1, T2, T3, F, S, H, M) :-
        max( 110H + 105M - ( 28T1 + 30T2 + 31T3 + 25F + 23S
                             + 9H + 6P + 27T1 + 28T2 + 25T3 + 10),
        assemble(T1, T2, T3, F, S, H, M) ).

     ?- maximal_profit(T1, T2, T3, F, S, H, P).
     T1 = 0, T2 = 0, T3 = 1, F = 0, S = 1, H = 1, P = 0.
```

Anfrage ergibt eindeutige Lösung Der Gewinn beträgt *12* Geldeinheiten.

```
     ?- assemble(T1, 1, T3, F, S, H, P).
     T1 = 0, T3 = 1, F = 0, S = 1, H + P = 1.
```

`sequence`: 0-1 Listen die keine 3 aufeinanderfolgende Nullen enthalten.

```
sequence([]).
sequence([_]).
sequence([_,_]).
sequence([Xi,Xj,Xk|L]) :-
        Xi + Xj + Xk >= 1,
        sequence([Xj,Xk|L]).
```

Queries:

?- sequence($[0,0,1,0,1,1,0]$).
$\rightsquigarrow$ yes
?- sequence($[0,1,0,0,0,1,0]$).
$\rightsquigarrow$ no
?- sequence($[A,B,\overline{A},\overline{B},A]$).
$\rightsquigarrow$ yes
?- sequence($[A,A,B,B,C,C]$).
$\rightsquigarrow$ $B+C \geq 1, A+B \geq 1$
?- sequence($[A\cdot B,\overline{A}\cdot B,C\cdot A,A{+}B,D{+}\overline{A}]$).
$\rightsquigarrow$ $B+C \geq 1, A+B \geq 1$

- Constraints werden benützt beim Programmieren und zur Lösungsbeschreibung. d.h. Untersuchen des Problems und Lösungsraums möglich.
- Während der Programmausführung werden Constraints dynamisch generiert und behandelt.
- vollständiger Löser: Regelanwendung nur zulässig wenn Constraintmenge konistent. d.h. Constraints kontrollieren Programmablauf.

Lineare Arithmetik

Beispiel: Fibonacci-Zahlen

```
fib(0, 1).
fib(1, 1).
fib(N, X1 + X2) :-
        N > 1,
        fib(N - 1, X1),
        fib(N - 2, X2).

2 ?-  fib(14,X).
X = 610
3 ?- fib(X,610).
X = 14
```

Beispiel: Darlehen

> `mortgage`(D, Laufzeit, Zinssatz, Bal, MZ) :-
>   Laufzeit = 1,
>   Bal = D $\cdot$ (1 + Laufzeit $\cdot$ $\frac{\text{Zinssatz}}{1200}$) - MZ.
> `mortgage`(D, Laufzeit, Zinssatz, Bal, MZ) :-
>   Laufzeit > 1,
>   `mortgage`(D $\cdot$ (1 + $\frac{\text{Zinssatz}}{1200}$) - MZ, Laufzeit-1, Zinssatz, Bal, MZ).

Darlehen: D, Laufzeit in Monaten: Laufzeit,
Jahreszinssatz in %: Zinssatz (monatlich abgerechnet),
Monatliche Zahlung: MZ,
RestSchuld: Bal.

Finanziere ein Darlehen von 100000 zu 12 Prozent für 15 Jahre.

> `?-mortgage`(100000, 180, 12, 0, MZ).
> MZ = 1200.17 `yes`.

Zusammenhang Darlehen und monatliche Zahlung bei Dauer 180 Monate und 12 % Zinssatz.

> `?-mortgage`(D, 180, 12, Bal, MZ).
> D = 0.166783 * Bal + 83.3217 * MZ `yes`.

Bidirektional und Constraints als Ausgabe

kombinatorische Probleme

Beispiel: SEND + MORE = MONEY

```
solve([S, E, N, D, M, O, R, Y]) :-
        constraints([S, E, N, D, M, O, R, Y]),
        gen_diff_digits([S, E, N, D, M, O, R, Y]).

constraints([S, E, N, D, M, O, R, Y]) :-
        S >= 0, E >= 0, N >= 0, D >= 0, M >= 0, O >= 0, R >= 0, Y >= 0,
        S <= 9, E <= 9, N <= 9, D <= 9, M <= 9, O <= 9, R <= 9, Y <= 9,
        S >= 1, M >= 1,
        C1 >= 0, C2 >= 0, C3 >= 0, C4 >= 0,
        C1 <= 1, C2 <= 1, C3 <= 1, C4 <= 1,
        M = C1,
        C2 + S + M = O + C1 * 10,
        C3 + E + O = N + 10 * C2,
        C4 + N + R = E + 10 * C3,
        D + E = Y + 10*C4,
        bit(C1), bit(C2), bit(C3), bit(C4).

bit(0).
bit(1).

gen_diff_digits(L) :-
        gen_diff_digits(L, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]).
gen_diff_digits([], _).
gen_diff_digits([H | T], L) :-
        select(H, L, L2), gen_diff_digits(T, L2).
select(H, [H | T], T).
select(H, [H2 | T], [H2 | T2]) :-
        select(H, T, T2).
```

Reelle Lösung als Approximation für ganzzahlige Lösung.

Constraints über endliche Bereiche

```
sendmory([S,E,N,D,M,O,R,Y]) :-
        generate_constraints([S,E,N,D,M,O,R,Y],[C1,C2,C3,C4 ]),
        enumerate([C1,C2,C3,C4,S,E,N,D,M,O,R,Y]).

generate_constraints([S,E,N,D,M,O,R,Y],[C1,C2,C3,C4]) :-
        [S,E,N,D,M,O,R,Y] :: [0 .. 9],
        [C1,C2,C3,C4] :: [0 .. 1],
        alldiff([S,E,N,D,M,O,R,Y],
        S ≠ 0,
        M ≠ 0,
                  C1 = M ,
         C2 + S + M = O + 10·C1,
         C3 + E +  O = N + 10·C2,
         C4 + N +  R = E + 10·C3,
              D +  E = Y + 10·C4.

enumerate([V|Vs]) :-
        indomain(V),
        enumerate(Vs).

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

Lokale Konsistenz: Eliminieren unmöglicher Werte aus der Domain der Variablen. Bsp: SMM
Da M $\neq$ 0, C1 = M, und C1 entweder 0 oder 1 ist, ist M = C1 = 1. Der Wert 1 wird aus der Domain aller Variablen {S,E,N,D,O,R,Y} eliminiert wegen dem `alldiff`-Constraint.
Da M = C1 = 1 reduziert

$$\mathrm{C2} + \mathrm{S} + \mathrm{M} = \mathrm{O} + 10\cdot\mathrm{C1}$$

zu

$$\mathrm{C2} + \mathrm{S} = \mathrm{O} + 9 \quad .$$

Da $O + 9 \geq 9$ muss $S \geq 8$ sein, da $C2 \leq 1$. Die Domain von S ist reduziert auf {8,9}.
Die Domain von O ist reduziert zu {0}, da $C2 + S \leq 10$. O muss kleiner als 2 sein aber 1 fehlt schon.
Endzustand:

$$\mathrm{S}{=}9,\ \mathrm{M}{=}1,\ \mathrm{O}{=}0,\ \mathrm{C1}{=}1,\ \mathrm{C2}{=}0,\ \text{and } \mathrm{E,N,D,R,Y} \in \{2,\ldots,8\}$$

Beim Aufzählen wählen wir einen möglichen Wert einer Variablen mit Wertebereich mehr als einem Element und Fixieren diese Variable auf den gewählten Wert. Mit dieser Information werden die Domains der anderen Variablen weiter reduziert bis entweder einer Lösung gefunden wird oder Inkonsistenz erkannt wird (Backtracking).

```
alldiff([]).
alldiff([X|Y]) :-
        outof(X,Y),
        alldiff(Y).

outof(X,[]).
outof(X,[F|T]) :-
        X ≠ F,
        outof(X,T).
```

Quadratische Anzahl an Constraints!!!
Algorithmische Behandlung: Sobald eine Variable einen Wert hat entferne diesen Wert aus der Domain der anderen (dieselbe Pruningeigenschaft, keine quadratische Anzahl an Constraints).
Bessere Pruningmöglichkeit durch bessere Algorithmen...

Globale Constraints

für Probleme die in der Praxis häufig auftreten.

- `cumulative` Resourcen-Constraints in Schedulingproblemen
- `among` Generalisierung von `atleast` und `atmost` Constraints. In einer Liste von Constraints müssen mindestens (dürfen höchstens) n Constraints wahr sein.
- `diffn` Generalisierung von `alldiff`
- `cycle` ...

Vorteile

- Algorithmen (OR) für spezielle Probleme. Effizienz erhöht, trotzdem deklarativ.
- Durch Integration in Sprache Kombination möglich. Komplexe Probleme mit diversen Teilproblemen können effizient behandelt werden.

Propagierungsbasierter Ansatz bedingt Wissen über Problembereich und über Lösungsverfahren zur Modellierung.
Anwendungsbereiche: car sequencing, Scheduling, Verschnittprobleme, Testpatterngenerierung, Färben von Graphen, microcode label assignment, Finanzentscheidungen, Planung, ...

Problemlösen mit Konsistenztechniken in CLP

Generelles Schema

```
problem(<Variablen>) :-
        <Constraintgenerierung>, % test
        <Aufzählen>. % generate
```

Constraintgenerierung

```
disjunct(Si,Di,Sj,Dj) :- Sj >= Si + Di.
disjunct(Si,Di,Sj,Dj) :- Si >= Sj + Dj.
```

fruehe Wahl (sehr wahrscheinlich falsch), nicht deterministisch

```
disjunct(Si,Di,Sj,Dj) :- among(1,[Sj >= Si + Di,Si >= Sj + Dj],1)
```

Redundante Constraints: Einbringen von zusätzlicher Information die mehr Pruning bewirkt.

Aufzählen

```
labeling([]).
labeling([X|Y]) :-
        indomain(X),        % deleteff(Var,[X|Y],Rest),
                            % indomain(Var)
        labeling(Y).        % labeling(Rest).
```

- Basisaufzähler
- mit generellen Heuristiken
- mit speziellen Heuristiken

Optimieren

Nur durch "branch and bound".
1.) Berechne Lösung (Aufzählen)
2.) Verbiete Lösungen mit schlechteren oder gleichen Zielfunktionswerten.

solange bis Problem inkonsistent. Letzte Lösung war Optimallösung.

2-dimensionales Verschnittproblem

Schneidemaschine: Holzplatte in Bretter
N verschiedene Arten von Brettern:
M Konfigurationen: bestimmt Anzahl und Art von Brettern aus Platte.

$S_{i_j}, 1 \leq i \leq M, 1 \leq j \leq N$: Anzahl der Bretter j aus Konfiguration i.
$R_j, 1 \leq j \leq N$: benötigte Anzahl von Brettern j.
$W_i, 1 \leq i \leq M$: Abfall in Konfiguration i.
Problem: Schneide benötigte Bretter, minimiere Abfall.
Bsp: 6 Arten von Brettern, 72 Konfigurationen, 4 Platten.
Bemerkung: Konfugrationen mussten auch berechnet werden. Andere Aspekte eines Problems mit logischer Programmiersprache lösen.

Lösung 1 :
$X_i : 1 \leq i \leq M = 72$: Anzahl der Platten geschnitten nach Konfiguration i.

4 Platten:

$$X_1 + \cdots + X_{72} = 4$$

Alle Bretter:

$$X1 * S_{1_j} + \cdots + X_{72} * S_{72_j} \geq R_j \qquad 1 \leq j \leq N = 6$$

Abfall:

$$\min(X_1 * W_1 + \cdots + X_{72} * W_{72})$$

```
solve([X1,...,X72],Abfall) :-
  X1 + ... + X72 = 4,
  X1*S1,1 + ... + X72*S72,1 >= R1,
  ...
  X1*S1,6 + ... + X72*S72,6 >= R6,
  Abfall = X1*W1 + .... + X72*W72,
  labeling([X1,...,X72]).
```

Lösung 2 :
$X_i : 1 \leq i \leq 4$: Konfiguration von Platte i.

$$1 \leq X_i \leq 72$$

$T_{i_j} : 1 \leq i \leq 4, 1 \leq j \leq 6$: Anzahl von Brettern j aus Platte i
$A_i : 1 \leq i \leq 4$: Abfall aus Platte i.

`element`-Constraint: `element(X,List,E)`
Das `X`-te elemnt von `List` ist `E`.

Berechne T_{i_j}:

$$\texttt{element}(X_i, [S_{1_j}, \ldots, S_{72_j}], T_{i,j}) \qquad 1 \leq i \leq 4, 1 \leq j \leq 6$$

Berechne A_i:

$$\texttt{element}(X_i, [W_1, \ldots, W_72], A_i) \qquad 1 \leq i \leq 4$$
$$\text{Abfall } = A_1 + A_2 + A_3 + A_4.$$

Alle Platten:

$$T_{1_j} + T_{2_j} + T_{3_j} + T_{4_j} \geq R_j \qquad 1 \leq j \leq 6$$

Keine symmetrischen Lösungen (Trick!):

$$X_1 \leq X_2, X_2 \leq X_3, X_3 \leq X_4$$

```
solve([X1,...,X4],Abfall) :-
  ....
  labeling([X1,...,X4]).
```

Suchraum von ca. 10^{43} auf 10^7
keine symmetrischen Lösungen

6 min Lösung 1 versus
1.5 min Lösung 2 versus
3 sek Lösung 2 + Domainsplitting