Constraint-Logikprogrammierung

Slim Abdennadher

LMU München

23.01.2003

Motivation: Arithmetik

```
summe(X,Y,Z) :- Z is X+Y.
```

Prolog: Laufzeitfehler

```
summe(1,Y,3).
summe(X,2,5).
summe(X,Y,3).
```

Gewünschtes Verhalten:

```
summe(1,Y,3).
Y=2.
summe(X,2,5).
X=3.
summe(X,Y,3).
X+Y=3.
```

Motivation: Generate & Test

- Prolog:

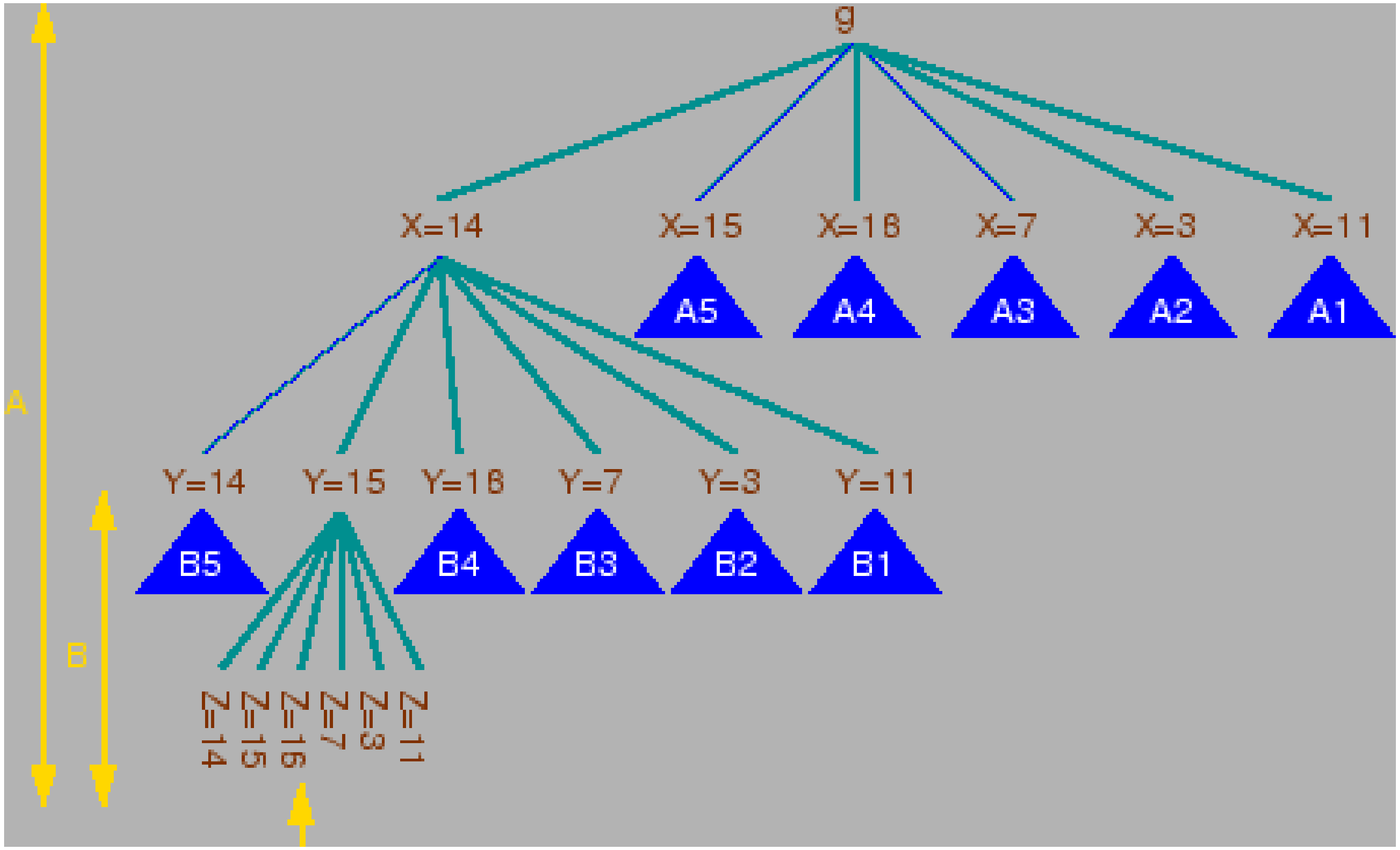
```
solution(X, Y, Z) :-
    p(X), p(Y), p(Z),
    test(X, Y, Z).

p(11). p(3). p(7). p(16). p(15). p(14).

test(X, Y, Z) :-  Y is X + 1, Z is Y + 1.
```

- Anfrage:

```
| ?- solution(X, Y, Z).
X = 14
Y = 15
Z = 16 ? ;
no
```

- 458 Schritte (alle Lösungen : 465 Schritte).

Motivation: Suchraum



Wieso Constraint-Logikprogrammierung: Logik + Steuerung

Programmierung kann unterteilt in

- “Kompetenz”-Komponente: zuständig für die Korrektheit
- “Performanz”-Komponente: zuständig für die Effizienz

In Prolog:

- der Kompetenz-aspekt: Klausellogik
- der Performanz-aspekt: SLD-Resolution

Programmierung in Prolog: Schwerpunkt auf Kompetenz, während Performanz meist automatisch .

Constraint-Logikprogrammierung (CLP):

Vorteile der deklarativen Programmierung + effizientere Programme

Wieso Constraint-Logikprogrammierung: Beliebige Bereiche

- Objekte in LP (Terme): uninterpretierte Strukturen
- Gleichheit: Syntaktische Äquivalenz
- Bedeutung zu Objekten nur mittels Klauseln.

  Beispiel: Mengen modelliert durch Listen
- Constraint-Logikprogrammierung führt allgemeine Bereiche in Logikprogrammierung ein:
  - CLP($\Re$): Constraint-Logikprogrammierung über reelle Zahlen
  - CLP(B): Constraint-Logikprogrammierung über boolesche Werte
  - CLP(FD): Constraint-Logikprogrammierung über endliche Bereiche
  - ...

CLP = Logikprogrammierung + Constraints

CLP kann als Verallgemeinerung des herkömmlichen logischen Programmierens angesehen werden

- Die Sprache wird erweitert um die Möglichkeit, Einschränkungen auf Variablen zu formulieren (z.B: in Prolog nicht möglich)
- Lösungsalgorithmus (Interpreter der logischen Programmiersprache) wird ergänzt (bzw. ersetzt) durch Mechanismen für das Sammeln und späteren Lösen von Constraints.

Von Prolog zu CLP: Arithmetik

- Prolog

```
summe(X,Y,Z) :- Z is X+Y.
```

- CLP($\Re$)

```
summe(X,Y,Z) :- Z=X+Y.
| ?- summe(1,Y,3).
Y=2.
| ?- summe(X,2,5).
X=3.
| ?- summe(X,Y,Z), X+Y>Z.
no
```

Beispiel: Fakultät

- Prolog

```
fakultaet(0,1).
fakultaet(X,Xfak) :- X>=1, Xminus1 is  X-1,
                     fakultaet(Xminus1,Xminus1fak),
                     Xfak is X*Xminus1fak.
```

- CLP($\Re$)

```
fakultaet(0,1).
fakultaet(X,X*Xminus1fak) :- X>=1,
                             fakultaet(X-1,Xminus1fak).
```

```
?- fakultaet(6,Y).
Y = 720.
?- fakultaet(X,40320).
X = 8.
```

Anwendungsbeispiel: Finanzwesen

```
%       D: Darlehenshoehe
%       T: Dauer der Rueckzahlung in Monaten
%       Z: Zinssatz pro Monat
%       R: Monatliche Rueckzahlungsrate
%       S: Restschuld nach T Monaten

mortgage(D, T, Z, R, S) :-
        T = 0, D = S.

mortgage(D, T, Z, R, S)  :-
        T > 0, T1 = T - 1, D1 = D + D*Z - R,
        mortgage(D1, T1, Z, R, S).
```

Anwendungsbeispiel: Finanzwesen

```
:- mortgage(100000,360,0.01,1025,S).
S = 12625.896680787611

:- mortgage(D,360,0.01,1025,0).
D = 99648.7893560409

:- S=<0, mortgage(100000,T,0.01,1025,S).
T= 374, S = -807.964032884383

:- mortgage(D,360,0.01,R,0).
R = 0.0102861259692555055*D
```

Von Prolog zu CLP: Test & Generate

- `test(X, Y, Z)` an den Anfang verschieben:

```
solution(X, Y, Z) :-
    test(X, Y, Z),
    p(X), p(Y), p(Z).
p(11). p(3). p(7). p(16). p(15). p(14).
```

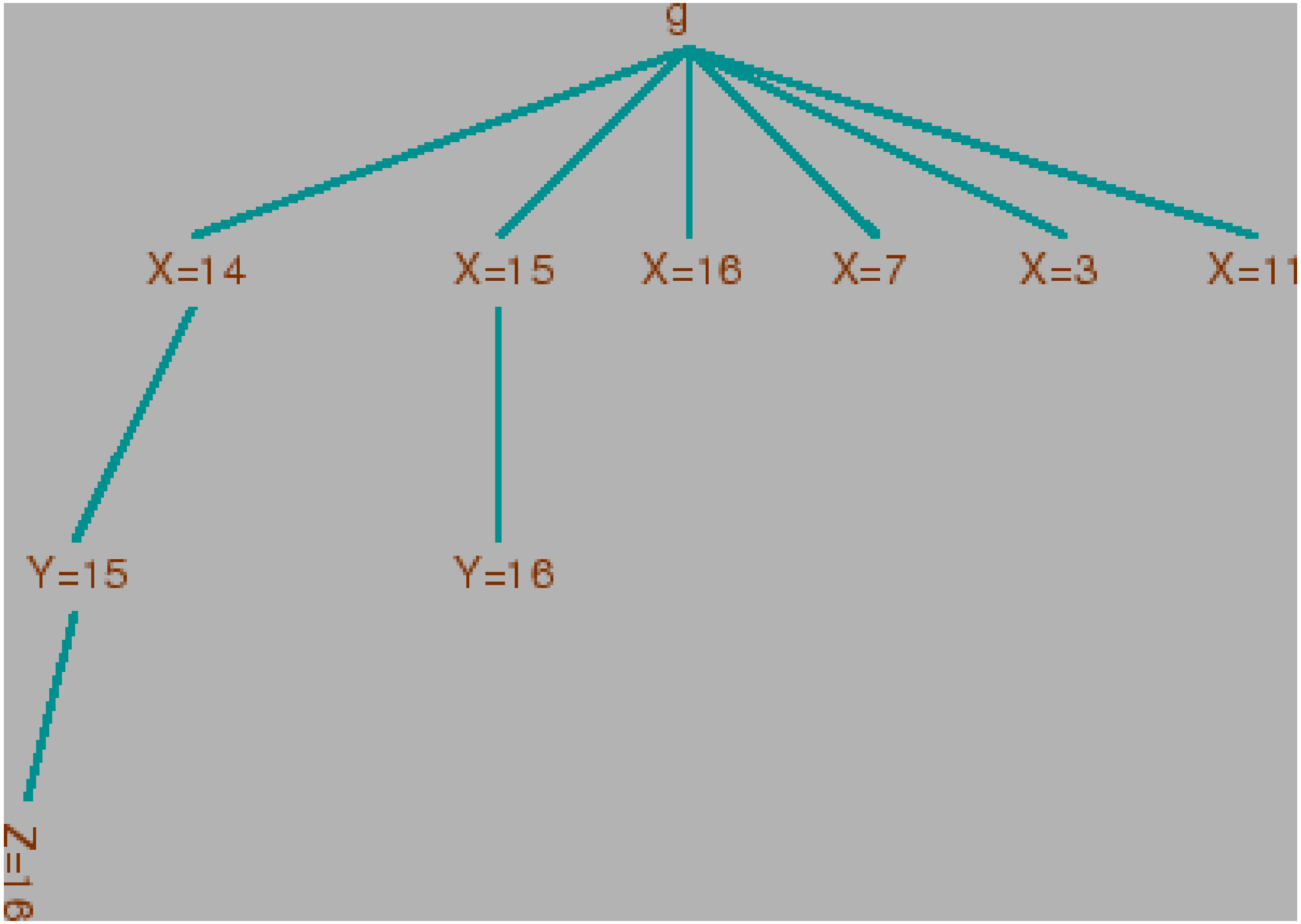
- Prolog: `test(X, Y, Z) :- Y is X+1, Z is Y+1.`

```
| ?- solution(X, Y, Z).
{INSTANTIATION ERROR: in expression}
```

- CLP($\Re$): `test(X, Y, Z) :- Y = X+1, Z = Y+1.`

```
?- solution(X, Y, Z).
Z = 16
Y = 15
X = 14 ?;
No
```

CLP: Suchraum



CLP über endliche Bereiche

- CLP($\Re$)

```
solution(X, Y, Z) :-
      test(X, Y, Z),
      p(X), p(Y), p(Z).
p(11). p(3). p(7). p(16). p(15). p(14).
test(X, Y, Z) :- Y = X+1, Z = Y+1.
```

- CLP(FD)

```
solution(X, Y, Z) :-
      [X,Y,Z] in [3,7,11,14,15],
      test(X, Y, Z),
      labeling([X,Y,Z]).
test(X, Y, Z) :- Y = X+1, Z = Y+1.
```

Kryptoarithmetische Rätsel

		S	E	N	D
+		M	O	R	E
=	M	O	N	E	Y

```
solve(S,E,N,D,M,O,R,Y) :-
        [S,E,N,D,M,O,R,Y] in 0..9,
        S≠0, M ≠0,
        alldifferent([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
```

Reduktion des Suchraums durch Constraint-Propagierung:

```
S=9, M=1, O=0, E in 4..7, N in 5..8, [D,R,Y] in 2..8
```

,

`labeling([S,E,N,D,M,O,R,Y]).`

Lösung durch Suche: `S=9, E=5, N=6, D=7, M=1,O=0, R=8, Y=2`

Anfänge der Constraint-Programmierung I

1963	Sutherland, Sketchpad, Graphiksystem für geometrisches Zeichnen
1970	Montanari, Formalisierung der Constraint-Netzwerke
1970	Fikes, REF-ARF, Sprache für ganzzahlige, lineare Gleichungen
1972	Colmerauer sowie R.A. Kowalski, Prolog
1977	Mackworth, Algorithmen für Constraint-Netzwerke
1978	Lauriere, Alice, Sprache für kombinatorische Probleme
1979	Borning, Thinglab, interaktives Graphiksystem
1980	Steele, Constraints, erste Constraint-Sprache, in LISP

Anfänge der Constraint-Programmierung II

1982	Colmerauer, Prolog II, Prolog mit Gleichheitsconstraints
1984	Eclipse Prolog, ECRC München und später IC-PARC London
1985	Sicstus Prolog, SICS
1987	Ait-Kaci, Austin, Texas, Life, Prolog mit Gleichheitsconstraints
	Jaffar , Lassez, Monash Univ. Melbourne, CLP-Schema
	Jaffar, CLP(R), Monash Univ. Melbourne, arithmetische Constraints
1988	Hentenryck, CHIP, ECRC München, endliche Bereiche
	Voda, Vancouver, Trilogy, CLP-ähnlich mit Ganzzahlarithmetik
	Older, Ottawa, Bell-Northern Research, Intervalle
	Aiba, Tokyo, ICOT, nichtlineare Gleichungssysteme
	Leler, Termersetzungssprache zum Schreiben von Constraintlösern
	Colmerauer, Prolog III, Univ. Marseille, Constraints über Listen

Kommerzielle Anwendungen

- Lufthansa: Personalplanung nach Störungen.
- Renault: Kurzfristige Fertigungsplanung.
- Airbus: Layout der Pilotenkabine.
- Siemens: Verifizierung von Schaltkreisen.
- Caisse d'epargne: Portfolio Management.

In Entscheidungsunterstützungssystemen zum Planen, Konfigurieren, für Design, Analyse.

Wollen Sie mehr?

Im Sommersemester 2003

Vorlesung: Constraint-Programmierung: Theorie und Anwendung