CLP zum Laufen bringen

Christian Pfaller
Software & Systems Engineering
Technische Universität München
pfaller@in.tum.de

Perlen der Informatik, 22.11.2006

Warum dieser Vortrag?

- Typische Sätze aus einführender Literatur:

„This Book is about constraint programming […] The central notation is that of a constraint. Informally a constraint on a sequence of variables is a relation on their domains.“

K. Apt: *Principles of Constraint Programming*

„This tutorial provides an introduction to programming in ECLiPSe. It assumes a broad understanding of constrained optimisation problems, some background in mathematical logic and in programming languages …“

Cheadle et al.: ECLiPSe – A Tutorial Introduction

➔ Hilfreich für den Einstieg in die CLP-Programmierung?

Mein Fazit aus der Einarbeitung

- Einführungen in Sprachimplementierungen / Werkzeuge
  - Setzen Erfahrung mit (C)LP voraus
- Allgemeine Einführungen in Constraint (Logik) Programmierung
  - Zeigen kaum, wie man ein Problem tatsächlich programmiert

- Dagegen bei Java, C++, C, etc.
  - Einführung der Konzepte anhand praktischer Anwendung

Übersicht

- Einordnung CLP
- Grundkonzepte
- LP vs. CLP
- Tools
- Erweiterungen
- Literatur

Historisches

- 1963
  - I. Sutherland: SKETCHPAD - Konzept von Constraints
- 1970er Jahre
  - 1971/72: PROLOG (Alain Colmerauer / Robert Kowalski)
  - Erste experimentelle Sprachen zum Constraintlösen
  - Constraint Satisfaction Problem im Bereich der KI
- 1980er
  - Früher 80er: Erste Programmiersprachen von gewisser Bedeutung
  - Joxan Jafar und Jean-Luois Lassez (IBM):
    Constraint Logic Programming (POPL'87, München)
  - CHIP – Mehmet Dincbas, (München, ECRC) 1988
  - PROLOG-III – A. Colmerauer (Marseille) 1990
  - CLP(R) – J. Jaffar et al. (Melbourne) 1992

Vergleich mit anderen Sprachen

- Frühe Prog.-Sprachen (z. B. FORTRAN66)
  - Widerspiegelung der physikalischen Rechnerarchtektur
- Imperative Sprachen
  - Definition von Strukturen/Objekten und Prozeduren korrespondierend zu Entitäten der Anwendungsdomäne
  - Insbesondere bei OO
  - Aber: Kaum Unterstützung zur Definition von Relationen zwischen Entitäten
- Constraint Logik Sprachen
  - Definition und Auswertung von Relationen bzw. Einschränkungen

Beispiel für eine Relation

- Ohmsche Gesetz

$$U = R \cdot I$$

(Spannung = Widerstand · Stromstärke)

- In Imperativen Sprachen nicht direkt verwendbar, Konsequenzen müssen explizit formuliert werden
  - I := U / R
  - R := U / I

Sprachen-Wirrwarr

- Logikprogrammierung (LP)
  - Logik zur Problemdarstellung (Klauseln) und Lösung (Resolution und Rückwärtsschließen)
- PROLOG
  - Auf LP basierende Programmiersprache
  - Deterministische Auswertungsreiheinfolge
- Constraint-Programmierung / Constraint-Lösen
  - Berechung von Lösungen unter Einschränkungen
  - Constraints über Domänen
- Constraint-Logik-Programmierung (CLP)
  - Logikprogrammierung erweitert um Constraints
  - Wird meist als Erweiterung von PROLOG um Constraints verstanden
  - Heutige PROLOG Implementierungen umfassen meist auch CLP

Übersicht

- Einordnung CLP
- Grundkonzepte LP
- LP vs. CLP
- Tools
- Erweiterungen
- Literatur

Was vs. Wie

- R. Kowalski 1979

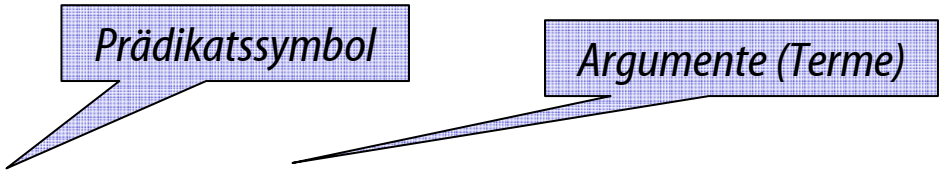
$$\text{Alogrithm} = \text{Logic} + \text{Control}$$

- Unterschied zwischen dem Was und dem Wie
  - Logische Komponente → Wissen über das Problem
  - Steuerkomponente → Auswertung des Wissens zur Problemlösung
- Ideal der LP
  - Programmierer muss nur das Wie, also das Problem beschreiben
  - Um den Programmablauf muss er sich nicht kümmern

Elemente in LP / Prolog

- LP-Programm
  - Menge von Klauseln
- Klauseln
  - Beschreiben Relationen, Sachverhalte, Zusammenhänge
  - Sind über Prädikate aufgebaut
- Prädikate
  - Logische Atome
  - Prädikate werden über Klauseln definiert
  - Klauseln eines Prädikats sind ODER-verknüpft
  - Bestehen aus Prädikatssymbol und Argumenten
  - Haben bestimmte Stelligkeit, Schreibweise name/stelligkeit

Mehr zu Klauseln



- Klauseln
  - Fakten *vater(fritz, thomas)*
    vater(thomas, maja)
  - Regeln *grossvater(Opa, Enkel)* $\leftarrow$ *vater(Opa, X)* $\wedge$ *vater(X, Enkel)*
- Regeln
  - bestehen aus Kopf (links von $\leftarrow$)
  - und Rumpf (rechts von $\leftarrow$), Konjunktion von Zielen
  - Fakten sind Regeln ohne Rumpf, bzw. der Rumpf TRUE ist
- Regeloperator $\leftarrow$
  - falls rechte Seite gilt, gilt auch linke (umgekehrte Implikation)

Symbole und Variablen

- Symbole (Konstanten) und Variablen
  - symbol (Kleinbuchst.): *fritz, maja, thomas, vater*
  - Variable (Großbuchst.): *Opa, Enkel, X*
- Variablen
  - Imperative Prog.: Verweis auf Speicherbereich
  - Hier: logische Variable im mathem. Sinn
  - Unterscheidung, ob Wert einer Variable bekannt (gebunden) oder unbekannt (ungebunden)
  - Gültigkeitsbereich von Variabeln: Nur in der jeweiligen Klausel

Syntax in Prolog

- Klausel wird mit Punkt . abgeschlossen
- Regeloperator wird durch :- dargestellt
- Logisches UND durch ein Komma ,
- Variablen beginnen mit Großbuchstaben oder Unterstrich _
- _ ist die anonyme Variable für beliebige Werte
  - wird nicht gebunden, alle Vorkommen von _ unabhängig
- Weitere Datenstrukturen, z. B.
  - Listen: [a,b,c]; [x|xs]
  - Strukturen / Tupel: (a,b)
- Anfragen werden nach ?- gestellt
- Arithmetik: +, -, *, /, mod, is

Anfragen

- Programm

```
vater(fritz, thomas).
vater(thomas, maja).
vater(fritz, anna).
grossvater(X,Y) :- vater(X,Z), vater(Z,Y).
```

- Beispiel-Anfragen

```
?-
vater(fritz,anna).
Yes
```

```
?-
vater(hans,fritz).
No
```

```
?- vater(fritz,X).
X = thomas ;
X = anna ;
No
```

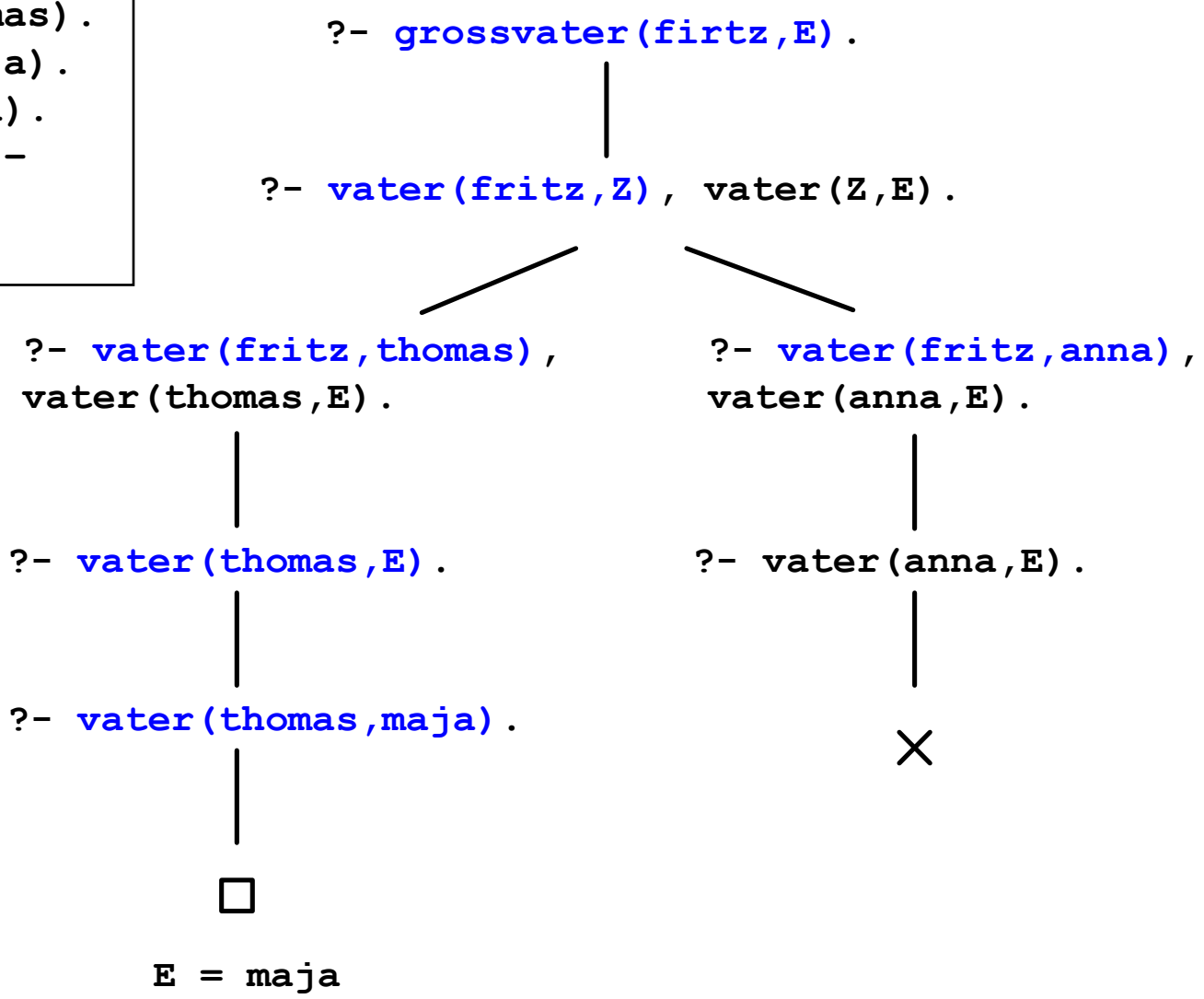
```
?-
grossvater(firtz,E).
E = maja ;
No
```

Antworten auf Anfragen: Resolution (Prinzip)

- Unifikation
  - Substitution von Variablen durch Terme
  - Zwei Terme sind unifizierbar, wenn durch Substitution eine syntaktische Gleichheit erreicht wird
- Ableitung
  - Wiederholtes Ersetzen von Zieltermen durch Klauseln aus dem Programm
  - Ableitung ist (endlich) gescheitert falls keine Unifikation mit einer Klausel des Programms mehr möglich ist.
- Anfrage ist ableitbar, wenn eine Ableitung möglich ist (LP-Kalkül)
  - Freie Wahl der zu subst. Variablen und ersetzenden Klauseln

Resolution: Beispiel

```
vater(fritz, thomas).
vater(thomas, maja).
vater(fritz, anna).
grossvater(X,Y) :-
vater(X,Z),
vater(Z,Y).
```



Übersicht

- Einordnung CLP
- Grundkonzepte
- LP vs. CLP
- Tools
- Erweiterungen
- Literatur

Neues durch Constraints

- LP / Prolog
  - *Algorithm = Logik + Control*
  - Basierend auf syntaktischer Gleichheit
  - Nur Symbole, keinen Domänen Typen etc.
- CLP
  - Control = Reasoning + Search
  - Auswahl guter Strategien zur Constriant-Propagierung
  - Wahl von Suchstrategien und Heuristiken zum schnellen Finden von Lösungen
  - Domänenspezifische Constraints, effizienter auswertbar
  - Rechnen mit symbolischen Werten, erfordert keine voll instanzierten Variablen

Sudoku

- Problem:
  - 81 Variablen
  - 27 Gruppen (Zeilen, Spalten, 3x3-Felder) á 9 Variablen sind je mit Permutation von [1..9] zu belegen

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

- In Prolog (pure Logik)
  - ```
    permute_all_nine(X) :-
        permutation([1,2,3,4,5,6,7,8,9],X).
    ```
- Problem lässt sich relativ „wörtlich“ formulieren
    ```

Sudoku (pure Logik)

```
sudoku(L) :-

        [A1,A2,A3, A4,A5,A6, A7,A8,A9,
         B1,B2,B3, B4,B5,B6, B7,B8,B9,
         C1,C2,C3, C4,C5,C6, C7,C8,C9,

         D1,D2,D3, D4,D5,D6, D7,D8,D9,
         E1,E2,E3, E4,E5,E6, E7,E8,E9,
         F1,F2,F3, F4,F5,F6, F7,F8,F9,

         G1,G2,G3, G4,G5,G6, G7,G8,G9,
         H1,H2,H3, H4,H5,H6, H7,H8,H9,
         I1,I2,I3, I4,I5,I6, I7,I8,I9] = L,

        permute_all_nine([A1,A2,A3,A4,A5,A6,A7,A8,A9]),   % Row1
        permute_all_nine([B1,B2,B3,B4,B5,B6,B7,B8,B9]),   % Row2
        permute_all_nine([C1,C2,C3,C4,C5,C6,C7,C8,C9]),   % Row3
        permute_all_nine([D1,D2,D3,D4,D5,D6,D7,D8,D9]),   % Row4
        permute_all_nine([E1,E2,E3,E4,E5,E6,E7,E8,E9]),   % Row5
        permute_all_nine([F1,F2,F3,F4,F5,F6,F7,F8,F9]),   % Row6
        permute_all_nine([G1,G2,G3,G4,G5,G6,G7,G8,G9]),   % Row7
        permute_all_nine([H1,H2,H3,H4,H5,H6,H7,H8,H9]),   % Row8
        permute_all_nine([I1,I2,I3,I4,I5,I6,I7,I8,I9]),   % Row9

        permute_all_nine([A1,B1,C1,D1,E1,F1,G1,H1,I1]),   % Column1
        ...
```

ziemlich langsam ...

Sudoku (CLP)

```
sudoku(L) :-

        [A1,A2,A3, A4,A5,A6, A7,A8,A9,
         B1,B2,B3, B4,B5,B6, B7,B8,B9,
         C1,C2,C3, C4,C5,C6, C7,C8,C9,

         D1,D2,D3, D4,D5,D6, D7,D8,D9,
         E1,E2,E3, E4,E5,E6, E7,E8,E9,
         F1,F2,F3, F4,F5,F6, F7,F8,F9,

         G1,G2,G3, G4,G5,G6, G7,G8,G9,
         H1,H2,H3, H4,H5,H6, H7,H8,H9,
         I1,I2,I3, I4,I5,I6, I7,I8,I9] = L,

        L in 1..9,

        all_different([A1,A2,A3,A4,A5,A6,A7,A8,A9]),   % Row1
        all_different([B1,B2,B3,B4,B5,B6,B7,B8,B9]),   % Row2
        ...
        label(L).
```

Unterschied

- LP allein
  - Belegt Variablen nacheinander mit Werten
  - Prüft, ob eine Variablenbelegung einen 9er-Permuation ist
  - Falls nicht, neuer Versucht mit neuen Werten
- CLP
  - Leget die endliche Domäne `1..9` für Variablen fest
  - Nutzt vorhandenen `all_different()` Constraint für endliche Domains (FD)
  - Propagiert zunächst Constraints für alle Variablen, welche sich aus `all_different()` ergeben
  - Sucht erst am Ende mit `label()` nach passenden Belegunge

Übersicht

- Einordnung CLP
- Grundkonzepte
- LP vs. CLP
- Tools
- Erweiterungen
- Literatur

SWI-Prolog

- http://www.swi-prolog.org/
  - Universiteit van Amsterdam, Jan Wielemaker
  - LGPL
  - Linux / Win / Mac / Source
- CLP(FD), CLP(R) und einige weitere sowie CHR
- Rudimentärer API für Java, auch C++
- Editor / IDE für Win, Emacs mode, Debugger
- Oft in Tutorials etc. verwendet
- Gut dokumentiert
- Wird laufend weiterentwickelt

ECLiPSe

- www.eclipse-clp.org
  - Früher kommerziell, seit 10/06 Open Source
  - „Cisco-style Mozilla Public License“
  - Linux, SunOS, Win
- Viele Constraint-Bibliotheken, Suchstrategien
- Effizienter als SWI
- API für Java, C++
- Emacs Mode, GUI für Debugging und weitere Entwicklungs-Tools (z. B. Prädikat-Browser)
- Viel Dokumentation
- Wird laufend weiterentwickelt

Übersicht

- Einordnung CLP
- Grundkonzepte
- LP vs. CLP
- Tools
- Erweiterungen
- Literatur

Erweiterungen

- Constraint-Handling-Rules
  - Erweiterung um eigene Constraints zu spezifizieren
- Funktional-Logische-Sprachen
  - Kombination von Logik-Programmierung und Funktionaler Programmierung
  - Curry
    - Constraint-Programmierung über CHR
    - PAKCS - http://www.informatik.uni-kiel.de/~pakcs
      Münster Curry Compiler
      http://danae.uni-muenster.de/~lux/curry/
  - Toy
    - Constraintlöser für endliche Domains: CFLP(FD)
    - http://www.fdi.ucm.es/profesor/fernan/TOY/
    - Gutes Tutorial

Übersicht

- Einordnung CLP
- Grundkonzepte
- LP vs. CLP
- Tools
- Erweiterungen
- Literatur

Einsteiger-Literatur

- J. R. Fisher: prolog :- tutorial
  http://www.csupomona.edu/~jrfisher/www/prolog_tutorial/contents.html
- P. Blackburn, J. Bos, K. Striegnitz: Learn Prolog Now!
  http://www.coli.uni-saarland.de/~kris/learn-prolog-now/
- K. Marriott and P.J. Stuckey: Programming with Constraints. An Introduction. MIT Press, 1998
- T. Frühwirth, S. Abdennadher: Constraint Programmierung. Springer, 1997
- U. Nilsson and J. Maluszynski: Logic, Programming and Prolog (2ED)
  http://www.ida.liu.se/~ulfni/lpp/