Constraint Logic Programming
a short tutorial

Inês Dutra

ines@dcc.fc.up.pt

http://www.dcc.fc.up.pt/~ines

Departamento de Ciência de Computadores

Faculdade de Ciências da Universidade do Porto

CRACS & INESC-Porto LA

LAI-OIL, June 2010
Outline

- What is CLP?
- A little bit of history (motivation)
- Systems, applications and clients
- Variable domain
- CLP by example
Constraint Logic Programming

What is CLP?

- the use of a rich and powerful language to model optimization problems (not only...)
- modelling based on variables, domains and constraints
Motivation:

1. to offer a declarative way of modelling constraint satisfaction problems (CSP)

2. to solve 2 limitations in Prolog:
   - Each term in Prolog needs to be explicitly evaluated and is not interpreted (evaluated):
     - \( x + 1 \) is a term that is not evaluated in Prolog. It is only a syntactic representation.
     - a variable can assume one single value.
   - Uniform computation, but not that powerful: depth-first search, “generate-and-test”.

3. to integrate Artificial Intelligence (AI) and Operations Research (OR)
CLP

CLP can use Artificial Intelligence (AI) techniques to improve the search: propagation, data-driven computation, “forward checking” and “lookahead”.

Applications: planning, scheduling, resource allocation, computer graphics, digital circuit design, fault diagnosis etc.

Clients: Michelin and Dassault, French railway SNCF, Swissair, SAS and Cathay Pacific, HK International terminals, Eriksson, British Telecom etc.
CLP

- CLP joins 2 research areas:
  - Introduction of richer and powerful data structures to Logic Programming (e.g.: replace unification by efficient manipulation of constraints and domains).
  - Consistency techniques: “generate-and-test” x “constrain-and-generate”
CLP

Systems:
- Prolog III, Colmerauer
- CHIP, Dincbas and Van Hentenryck (ECRC)
- OPL, Van Hentenryck
- Xpress
- CLP(R), Jaffar, Michaylov, Stuckey and Yap (Monash)
- ECLiPSe, Wallace (IC Parc)
- Oz, Smolka (DFKI)
- clp(FD), Diaz and Codognet (INRIA, France)

The CLP(X) scheme:
- “constraint solver”: replaces simple unification.
- 2 popular domains: arithmetic and boolean.
Arithmetic domain: linear constraints

- Prolog cannot solve $x - 3 = y + 5$.
- CLP(R): first language to introduce arithmetic constraints.
- Linear arithmetic expressions composed by: numbers, variables and operators (negation, addition, subtraction, multiplication and division).
- Example: $t_1 \mathrel{R} t_2$, with $R = \{ >, \geq, =, \leq, <, = \}$
- Popular decision procedures:
  - Gauss elimination.
  - Simplex (most popular):
    - average good behavior
    - popular
    - incremental
Arithmetic Domain: Linear Constraints

- Example:
  - A meal consists of starter, main course and dessert
  - database with various kinds of food and their caloric values
  - Problem: produce a menu with light meals (caloric value < 10Kcal)
Arithmetic Domain: Linear Constraints

light_meal(A,M,D) :-
  I > 0, J > 0, K > 0,
  I + J + K <= 10,
  starter(A,I),
  main_course(M,J),
  dessert(D,K).

main_course(M,I) :-
  meat(M,I).

main_course(M,I) :-
  fish(M,I).

stater(salad,1).

stater(soup,6).

meat(steak,5).
meat(pork,7).
fish(sole,2).
fish(tuna,4).
dessert(fruit,2).
dessert(icecream,6).
Arithmetic Domain: Linear Constraints

- Intermediate results = compute states.
- 2 components: constraint store and continuation of objectives.
  I + J + K <= 10, I > 0, J > 0, K > 0 ♦ starter(A,I),
  main_course(M,J), dessert(D,K).
- A = salad, I = 1, 1 + J + K <= 10, 1>0, J>0, K>0 ♦
  main_course(M,J), dessert(D,K).
- A = salad, I = 1, M=M1, J=I1, 1 + J + K <= 10, 1>0, J>0, K>0 ♦
  meat(M1,I1), dessert(D,K).
- A = salad, I = 1, M=steak, J=5, M1=steak, I1 = 5, 1 + 5 + K <= 10, 1>0, 5>0, K>0 ♦
  dessert(D,K).
- A = salad, I = 1, M=steak, J=5, M1=steak, I1 = 5, D=fruit, K = 2, 1 + 5 + 2 <= 10, 1>0, 5>0, 2>0 ♦.
Arithmetic Domain: Linear Constraints

Inconsistent derivation:

A = pasta, I = 6, M=steak, J=5, M1=steak, I1 = 5, 6 + 5 + K
<= 10, 5>0, 6>0, K > 0 ◇ dessert(D,K).
Example: multiply 2 complex numbers: \((R1 + I*I1) \times (R2 + I*I2)\).

\[
\text{zmul}(R1, I1, R2, I2, R3, I3) :-
\]
\[
R3 = R1 \times R2 + I1 \times I2,
\]
\[
I3 = R1 \times I2 + R2 \times I1.
\]

Query: \(\diamond \text{zmul}(1,2,3,4,R3,I3)\)

Equations become linear.

Solution: \(R3 = -5, I3 = 10\) (definite solution)

Query: \(\diamond \text{zmul}(1,2,R2,I2,R3,I3)\)

Solution:

\[
I2 = 0.2*I3 - 0.4*R3
\]
\[
R2 = 0.4*I3 + 0.2*R3
\]

yes (undefined solution)
Arithmetic Domain: non-linear Constraints

- Same example: multiply 2 complex numbers:
  \[(R1 + I*I1) \times (R2 + I*I2)\]

- Query: \(\Diamond \ zmul(R1,2,R2,4,-5,10), \ R2 < 3.\)

- CLP(R): (do not solve non-linear equations)
  
  \[
  \begin{align*}
  R1 &= -0.5*R2 + 2.5 \\
  3 &= R1*R2 \\
  R2 &< 3 \\
  \text{Maybe}
  \end{align*}
  \]

- applications of non-linear equations: computational geometry and financial applications (various algorithms used).
Boolean domain

- **Main Application:** digital circuit design (hardware verification) and theorem proof.

- **Boolean terms:** truth values (F - False or T - True), variables, logical operators, one single constraint: equality.

- **Various uniication algorithms for boolean constraints.**

- **Solution:** provides a decision procedure for propositional calculus (NP-complete).
Boolean domain

Example: full adder (operators # (xor), * (and), + (or))

\[
\text{add}(I1, I2, I3, O1, O2) :-
\]
\[
X1 = I1 \# I2,
A1 = I1 \ast I2,
O1 = X1 \# I3,
A2 = I3 \ast X1,
O2 = A1 + A2.
\]

Query: \( \Diamond \text{add}(a, b, c, O1, O2) \)

Solution: \( O1 = a + b + c, \quad O2 = (a \land b) + (a \land c) \# (b \land c) \)
Consistency techniques

- Eliminate inconsistent 'labellings' by constraint propagation
  information about the values of variables.
- exemplos: arc-consistency, forward checking, generalized propagation.
- Example: task scheduling.

```
T2                  before(T1,T2).
/                   before(T1,T3).
/                   before(T2,T6).
T1                  before(T3,T5).
\       T6          before(T4,T5).
\       /            before(T5,T6).
T3  ---  T5         notequal(T2,T3).
/                   
T4
```
Consistency Techniques

Example: \( T_1 \in \{1, 2, 3, 4, 5\}, T_2 \in \{1, 2, 3, 4, 5\} \).

\textbf{before}(T_1, T_2) \rightarrow \text{apply consistency:}

- \( T_1 \in \{1, 2, 3, 4\} \)
- \( T_2 \in \{2, 3, 4, 5\} \)

\begin{itemize}
  \item Value 5 removed from \( T_1 \), because there is no value in \( T_2 \) that can satisfy \((T_1=5) < T_2\).
  \item Value 1 removed from \( T_2 \), for the same reason.
\end{itemize}
Arc Consistency

- \( T_1 \in \{1, 2\}, T_2 \in \{2, 3, 4\}, T_3 \in \{2, 3\}, T_4 \in \{1, 2, 3\}, T_5 \in \{3, 4\}, T_6 \in \{4, 5\}. \)

- Value 2 from \( T_1 \) is chosen (\( T_1 \) is “labelled” with value 2).

- Using propagation: \( T_2 \in \{3, 4\} \) and \( T_3 \in \{3\} \)

- \( T_2 \in \{4\} \) from notequal(\( T_2, T_3 \)).

- Finally: \( T_1 \in \{2\}, T_2 \in \{4\}, T_3 \in \{3\}, T_4 \in \{1, 2, 3\}, T_5 \in \{4\}, T_6 \in \{5\} \)
Infinite Domains

- Utilization of maximum and minimum values to solve constraints in the linear domain.
- E.g.: x,y,z with domains [1..10] with constraint $2x + 3y + 2 < z$
- Removing inconsistencies values:
  - 10 is the largest value for $z$, then: $2x + 3y < 8$
  - 1 is the smallest possible value for $y$, then: $2x < 5$
  - $x$ can only assume values \{1,2\}
  - $3y < 6$, $y < 2$, $y \in \{1\}$
  - $z > 7$, $z \in \{8,9,10\}$
Basic programming techniques

- Define problem variables and their domains.
- Establish the constraints between variables.
- Search for solution.

?- [X,Y,Z]::1..10,
   2 * X + 3 * Y + 2 #< Z,
   indomain(X), indomain(Y), indomain(Z).
Algorithms and other examples

Forward Checking: Example

\( n = 8 \)

(1) \( V1 = 1 \)  \( \implies \)  \( V2 = \{3,4,5,6,7,8\} \)
\( V3 = \{2,4,5,6,7,8\} \)
\( V4 = \{2,3,5,6,7,8\} \)
\( V5 = \{2,3,4,6,7,8\} \)
\( V6 = \{2,3,4,5,7,8\} \)
\( V7 = \{2,3,4,5,6,8\} \)
\( V8 = \{2,3,4,5,6,7\} \)

(2) \( V2 = 3 \)  \( \implies \)  \( V3 = \{5,6,7,8\} \)
\( V4 = \{2,6,7,8\} \)
\( V5 = \{2,4,7,8\} \)
\( V6 = \{2,4,5,8\} \)
\[ V7 = \{2, 4, 5, 6\} \]
\[ V8 = \{2, 4, 5, 6, 7\} \]
Algorithms

- infinite domains: linear programming, simplex, revised simplex, convex hull, Gauss elimination.
- finite domains: forward checking, lookahead, arc-consistency.

For finite domains, 2 problems:

- choice of variable:
  - most-constrained: smallest domain
  - most constraining: mostly constrains domains of other variables

- choice of a value for a variable:
  - first-fail principle.
  - least constraining: value that constrains less sets of values of other variables
Map Coloring

Map Coloring: 3 colors

+-------------+
  \leftarrow\rightarrow
  \text{blue} \quad \text{red} \quad \text{green}

\begin{align*}
\text{A} & \quad \text{B} \\
\text{C} & \quad \text{D} \\
\text{E} & \quad \text{F}
\end{align*}
Heuristics

- most-constrained variable: allows to solve the n-queens problem with n equals 100
- pure forward checking: only solves for n = 30
- least-constraining: solves for n = 1000
A comparative study of eight constraint programming languages, by Antonio Fernandez and Pat Hill, 2000


Constraint logic programming: A survey J. Jaffar, M. J. Maher, 1994