2nd CHR Summer School

Topics:

Introduction to Constraint Programming

Prof. Dr. Slim Abdennadher

5.9.2011 - 9.9.2011

Constraint Programming: Much Quoted Sentence

> **Constraint Programming** represents one of the closest approaches computer science has yet made to the Holy Grail of programming: **the user states the problem, the computer solves it**.

Eugene C. Freuder, Inaugural issue of the *Constraints Journal*, 1997.

Constraint programming has been identified by ACM as one of the strategic directions for computing research.

Is this possible?

- Let us try to solve the following Problem: Given a natural number N, the sum of the squares of 4 or fewer numbers is equal to N.
- Mathematical statement of the problem:

$$\forall N \exists S_1, S_2, S_3, S_4 \; N = S_1^2 + S_2^2 + S_3^2 + S_4^2$$

- Encoding the problem in a Constraint Language:

```
:- use_module(library(clpfd)).
solve(N,Sol) :-
        L = [S1,S2,S3,S4],
        N #= S1*S1 + S2*S2 + S3*S3 + S4*S4,
        labeling([],L).
```

- It works perfectly!

Constraint Programming in a Nutshell

Given a **description** of the problem in natural language, Constraint Programmers (CP)

- first **model** the problem as a **constraint problem** including **variables**, their **domains** and **constraints**
- then **resolve** this by means of **constraint solvers**

What are Constraints?

- **Variable:** A place holder for values

$$X, Y, Z, L$$

- **Function Symbol:** Mapping values to values

$$+, -, \times$$

- **Relation Symbol:** Relation between values

$$=, \neq, \leq$$

- **Primitive Constraint:** Constraint relation with arguments

$$X \leq 8$$

$$2X + Y = 10$$

What are Constraints?

- **Constraint:** a conjunction of primitive constraints

$$X \leq 5 \wedge X = Y \wedge 2X + Y = 10$$

Constraint Satisfaction

- Given a constraint C, two questions:
  - **Satisfaction:** Does it have a solution?
  - **Solution:** Give me a solution if it has one?
- **Constraint Solver:** answers the satisfaction problem

Historical Remarks

- **60s:** Early use of constraints in graphics software, e.g. SKETCHPAD.
- **60s, 70s:** Constraint networks in artificial intelligence (CSP).
- **70s:** Logic programming (Prolog).
- **80s:** Constraint logic programming (CLP).
  - First steps in CLP by Jaffar and Lassez ca. 1987
  - Finite Domain CSP studied by Mackworth ca. 1992
  - First Practical implementations by Colmerauer et. al (Prolog III) c.a. 1990
  - CHIP commercialized by COSYTEC 1990
  - ILOG SOLVER introduced 1994
  - First global constraint (all-different) c.a. 1994

⋮

Constraint Logic Programming

- **Constraint Satisfaction Problems (CSP)**: artificial intelligence (1970s)
- **Constraint Logic Programming (CLP)**: developed in the mid-1980's
  - two declarative paradigms: constraint solving and logic programming
  - more expressive, flexible, and in general more efficient than logic programs
  - **Example 1**: $X - Y = 3 \ \wedge \ X + Y = 7$ leads to $X = 5 \ \wedge \ Y = 2$
  - **Example 2**: $X < Y \ \wedge \ Y < X$ fails without the need to know values

In short

LP = Logic + Control

CP = Model + Reasoner

Constraint Satisfaction

- How do we answer the question?
  - **Generate-and-Test in LP**: impractical, facts used in passive manner only
  - **Constrain-and-generate in CLP**: use facts in active manner to reduce the search space (constraints)

The Idea

- **Example – Combination Lock:**
  0 1 2 3 4 5 6 7 8 9
  Greater or equal 5.
  Prime number.
- **Declarative problem** representation by variables and constraints:
  $x \in \{0, 1, \ldots, 9\} \ \wedge \ x \geq 5 \ \wedge \ \text{prime}(x)$
- **Constraint propagation and simplification** reduce search space:
  $x \in \{0, 1, \ldots, 9\} \ \wedge \ x \geq 5 \ \rightarrow \ x \in \{5, 6, 7, 8, 9\}$
  $x \in \{5, 6, 7, 8, 9\} \ \wedge \ \text{prime}(x) \ \rightarrow \ x \in \{5, 7\}$

Constraint Programming

- **Declarative modeling by constraints:**
  - What are Constraints?
  - **Formally**: A relation between objects (problem variables)
  - **Informally**: A statement on how the value of one or more variables restrict the possible values of others.
- **Automatic constraint reasoning:**
  Propagation of the effects of new information.
  Simplification makes implicit information explicit.
- **Solving combinatorial problems efficiently:**
  Easy Combination of constraint solving with search and optimization.

Programming with Constraints versus Imperative Paradigm

Imperative Paradigm

- A program is viewed as operating on a store which holds the **(concrete) values** of the variables (e.g. `X=3`).
- The program progresses and achieves the results **by changing the values** of variables in the store.

Constraint Programming

- The store holds the **possible values** of the variables, i.e. the variables could be **partially** bound (eg. `X < 4`).
- The program does not change the value of a variable, instead it **reduces the possible values** of it.

Crypto-Arithmetic Puzzles

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

- Replace each letter by a different digit
- So that the equation above is correct
- The numbers should not start with a 0.

Crypto-Arithmetic Puzzles: Java Program

```
class sendmoremoney
{
 public static void main(String args[])
 {
  int num = 99999999;
  int s,e,n,d,m,o,r,y, count = 0;
  while(num > 0)
  {
   int num2 = num;
   --num;

   y = num2 % 10; num2 /= 10;
   r = num2 % 10; num2 /= 10;
   o = num2 % 10; num2 /= 10;
   m = num2 % 10; num2 /= 10;
   d = num2 % 10; num2 /= 10;
   n = num2 % 10; num2 /= 10;
   e = num2 % 10; num2 /= 10;
   s = num2 % 10; num2 /= 10;
```

Crypto-Arithmetic Puzzles: Java Program

```
    if(s == e || s == n || s == d || s == m || s == o || s == r ||
       s == y || e == n || e == d || e == m || e == o || e == r ||
       e == y || n == d || n == m || n == o || n == r || n == y ||
       d == m || d == o || d == r || d == y || m == o || m == r ||
       m == y || o == r || o == y || r == y || s == 0 || m == 0)
     continue;
    int send = s * 1000 + e * 100 + n * 10 + d;
    int more = m * 1000 + o * 100 + r * 10 + e;
    int money = m * 10000 + o * 1000 + n * 100 + e * 10 + y;

    if(send + more == money)
     System.out.println(" A solution "
          + send + " " + more + " " + money);
   }
  }
 }
```

Crypto-Arithmetic Puzzles: Constraint Program

A CLP Program Modeling the Problem

```
solve(Vars):-
        Vars=[S,E,N,D,M,O,R,Y], % variable generation
        Vars :: 0..9,
        alldifferent(Vars),     % constraint generation
        S #\= 0,
        M #\= 0,
                  1000*S+100*E+10*N+D
                + 1000*M+100*O+10*R+E
        #= 10000*M+1000*O+100*N+10*E+Y.
```

Solving the Puzzle

```
?- 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 ?

?- solve([S,E,N,D,M,O,R,Y]), labeling([],[S,E,N,D,M,O,R,Y]).

D = 7, E = 5, M = 1, N = 6,
O = 0, R = 8, S = 9, Y = 2 ?
```

Constraint Propagation using Interval reasoning

Constraint propagation is an **inference rule** for finite domain problems that reduces the domains of variables:

- Given the following constraints:

$$\begin{aligned} X &\in \{0,\ldots,9\} \\ Y &\in \{0,\ldots,9\} \\ X+Y &= 9 \\ 2X+4Y &= 24 \end{aligned}$$

- We can conclude:

  Eq.2 $\Rightarrow X \in \{0,\ldots,9\}, Y \in \{2,\ldots,6\}$
  Eq.1 $\Rightarrow X \in \{3,\ldots,7\}, Y \in \{2,\ldots,6\}$
  Eq.2 $\Rightarrow X \in \{4,\ldots,6\}, Y \in \{3,4\}$
  Eq.1 $\Rightarrow X \in \{5,6\}, Y \in \{3,4\}$
  Eq.2 $\Rightarrow X \in \{6\}, Y \in \{3\}$

SEND + MORE = MONEY (Propagation)

- The equation below initiates **propagation**

  1000*S + 100*E + 10*N + D + 1000*M + 100*O + 10*R + E = 10000*M + 1000*O + 100*N + 10*E + Y

- Transform the equation into a simpler (equivalent) one:

  1000*S + 91*E + D + 10*R = 9000*M + 900*O + 90*N + Y

- M should be less than or equal to
  $\frac{1}{9} * max(S) + \frac{91}{9000} * max(E) + \frac{1}{9000} * \max(D) + \frac{1}{900} * max(R) - \frac{1}{10} * min(O) - \frac{1}{100} * min(N) - \frac{1}{9000} * min(Y)$

- Given the current domain this implies that
  $M \leq \frac{9}{9} + \frac{91*9}{9000} + \frac{9}{9000} + \frac{9}{900} - \frac{0}{10} - \frac{0}{100} - \frac{0}{9000} = 1.102$

- Thus the domain of M is updated to `[1..1]`

- S should be greater than $9 * min(M) + \frac{9}{10} * min(O) + \frac{9}{100} * min(N) + \frac{1}{1000} * min(Y) - \frac{91}{1000} * max(E) - \frac{1}{1000} * max(D) - \frac{1}{100} * max(R)$

- Given the current domain, we infer that $S \geq 8.082$

Modelling: How to Model a Problem?

- **Choose the variables** that will be used to represent the parameters of the problem (this may be straightforward or difficult).
- **Choose the corresponding domains for variables**.
- **Model the idealized relationships** between these variables using the primitive constraints available in the domain.
- **Several models** for a problem
  - **Different variables**
  - **Different constraints**
- **Efficiency** depends on
  - **Number of variables**
  - **Type of constraints**

Modelling Example – Simple Allocation Problem

- 4 **workers**: w_1, w_2, w_3, w_4
- 4 **products**: p_1, p_2, p_3, p_4
- **A worker is allocated a product and vice versa**.

Profit of worker w_i with product p_j is given by

	p_1	p_2	p_3	p_4
w_1	7	1	3	4
w_2	8	2	5	1
w_3	4	3	7	2
w_4	3	1	6	3

Problem is solved when total profit is at least 19.

Modelling with Boolean Variables

- **Operations Research** method: 16 Boolean variables B_{ij}
- $B_{ij} = 1$: worker i is allocated to product j

Model with **linear arithmetic constraints**:

- **Worker w_i is allocated to a single product:**

$$B_{i1} + B_{i2} + B_{i3} + B_{i4} = 1$$

- **Product p_j is allocated to a single worker:**

$$B_{1j} + B_{2j} + B_{3j} + B_{4j} = 1$$

- **Total profit:**

$$\begin{aligned} P \quad = \quad & 7 * B_{11} + B_{12} + 3 * B_{13} + 4 * B_{14} \\ & + 8 * B_{21} + 2 * B_{22} + 5 * B_{23} + B_{24} \\ & + 4 * B_{31} + 3 * B_{32} + 7 * B_{33} + 2 * B_{34} \\ & + 3 * B_{41} + B_{42} + 6 * B_{43} + 3 * B_{44} \end{aligned}$$

Modelling with Boolean Variables: CLP Program

```
assignment(List) :-
      List = [B11,B12,B13,B14,B21,B22,B23,B24,
              B31,B32,B33,B34,B41,B42,B43,B44],
      domain(List,0,1),
      B11 + B12 + B13 + B14 #= 1, B21 + B22 + B23 + B24 #= 1,
      B31 + B32 + B33 + B34 #= 1, B41 + B42 + B43 + B44 #= 1,
      B11 + B21 + B31 + B41 #= 1, B12 + B22 + B32 + B42 #= 1,
      B13 + B23 + B33 + B43 #= 1, B14 + B24 + B34 + B44 #= 1,
      P #= 7 * B11 + B12 + 3 * B13 + 4 * B14
         + 8 * B21 + 2 * B22 + 5 * B23 + B24
         + 4 * B31 + 3 * B32 + 7 * B33 + 2 * B34
         + 3 * B41 + B42 + 6 * B43 + 3 * B44,
      P #>= 19,
      labeling([],List).
```

Modelling with Boolean Variables: Program Execution

```
:- assignment([B11,B12,B13,B14,B21,B22,...]).
B11 = 0,   B11 = 0,   B11 = 0,   B11 = 1
B12 = 0,   B12 = 0,   B12 = 1,   B12 = 0
B13 = 0,   B13 = 0,   B13 = 0,   B13 = 0
B14 = 1,   B14 = 1,   B14 = 0,   B14 = 0
B21 = 1,   B21 = 1,   B21 = 1,   B21 = 0
B22 = 0,   B22 = 0,   B22 = 0,   B22 = 1
B23 = 0,   B23 = 0,   B23 = 0,   B23 = 0
B24 = 0,   B24 = 0,   B24 = 0,   B24 = 0
B31 = 0,   B31 = 0,   B31 = 0,   B31 = 0
B32 = 0,   B32 = 1,   B32 = 0,   B32 = 0
B33 = 1,   B33 = 0,   B33 = 1,   B33 = 1
B34 = 0,   B34 = 0,   B34 = 0,   B34 = 0
B41 = 0,   B41 = 0,   B41 = 0,   B41 = 0
B42 = 1,   B42 = 0,   B42 = 0,   B42 = 0
B43 = 0,   B43 = 1,   B43 = 0,   B43 = 0
B44 = 0,   B44 = 0,   B44 = 1,   B44 = 1
```

- Four solution to the problem
- **28 derivation steps in total**, where there is a choice of a value for a variable.

Modelling with Variables for the Workers

- 4 variables: W_1, W_2, W_3, W_4
- If worker i is allocated to product j, then W_i equals j.
- **Different value for each variable:**

$$W_i \neq W_j \text{ for all } 1 \leq i \neq j \leq 4$$

Modelling by the constraint `all_distinct`

```
all_distinct([W1,W2,W3,W4])
```

- **Profit:**
  - profit_i: Profit of worker i for every product.
  - $\text{profit}_i[j]$: Profit, if product j is done by worker i
  - Profit of worker i: $\text{profit}_i[W_i]$

Modelling by the constraint `element`

```
element(W1,[7,1,3,4],WP1)
```

($\text{profit}_1[1] = 7$, $\text{profit}_1[2] = 1$, $\text{profit}_1[3] = 3$, $\text{profit}_1[4] = 4$)

Modelling with Variables for the Workers (3)

`element(?X,+List,?Y)`

- `X`, `Y`: integers or domain variables
- `List`: list of integers or domain variables
- `element(?X,+List,?Y)`: True if the `X`-th element of `List` is `Y`.
- Operationally, the domains of `X` and `Y` are constrained s.t. for every element from the domain of `X`, there is a compatible element from the domain of `Y`, and vice versa.

Query: `Value #> 5, element(N,[7,8,4,3],Value).`

Answer: `Value in 7..8, N in 1..2`

Modelling with Variables for the Workers (4)

```
assignment(W1,W2,W3,W4) :-
        W1 in 1..4,
        W2 in 1..4,
        W3 in 1..4,
        W4 in 1..4,
        all_distinct([W1,W2,W3,W4]),
        element(W1,[7,1,3,4],WP1),
        element(W2,[8,2,5,1],WP2),
        element(W3,[4,3,7,2],WP3),
        element(W4,[3,1,6,3],WP4),
        P #= WP1 + WP2 + WP3 + WP4,
        P #>= 19,
        labeling([],[W1,W2,W3,W4]).
```

Modelling with Variables for the Workers (5)

```
:- assignment(W1,W2,W3,W4).

W1 = 1,    W1 = 2,    W1 = 4,    W1 = 4
W2 = 2,    W2 = 1,    W2 = 1,    W2 = 1
W3 = 3,    W3 = 3,    W3 = 2,    W3 = 3
W4 = 4,    W4 = 4,    W4 = 3,    W4 = 2
```

14 derivation steps in total

Modelling with Variables for the Products (1)

```
assignment(T1,T2,T3,T4) :-
        T1 in 1..4,
        T2 in 1..4,
        T3 in 1..4,
        T4 in 1..4,
        all_distinct([T1,T2,T3,T4]),
        element(T1,[7,8,4,3],TP1),
        element(T2,[1,2,3,1],TP2),
        element(T3,[3,5,7,6],TP3),
        element(T4,[4,1,2,3],TP4),
        P #= TP1 + TP2 + TP3 + TP4,
        P #>= 19,
        labeling([],[T1,T2,T3,T4]).
```

Modelling with variables for the products (2)

```
:- assignment(T1,T2,T3,T4).
```

```
T1 = 1,    T1 = 2,    T1 = 2,    T1 = 2
T2 = 2,    T2 = 1,    T2 = 3,    T2 = 4
T3 = 3,    T3 = 3,    T3 = 4,    T3 = 3
T4 = 4,    T4 = 4,    T4 = 1,    T4 = 1
```

7 derivation steps in total

Reason: Better propagation because profit depends more on the product than on the worker

`TP1 in 3..8, TP2 in 1..3, TP3 in 3..7, TP4 in 1..4`

$$TP_1 \geq \min(P) - \max(TP_2) - \max(TP_3) - \max(TP_4)$$

yields $TP_1 \geq 5$. The constraint `element(T1,[7,8,4,3],TP1)` then reduces the domain of `T1` to `1..2`.

Modelling Comparison

A difficult task

- **Efficiency**
  - **Propagation of the constraints**
  - **Number of variables** (in general: the less the better)
- **Flexibility**: Additional constraints
  - **Depends on the type of the constraints**

Modelling Comparison: Our Example

- Ensure, that never worker 1 is allocated product 1 and worker 4 is allocated product 4.
  - **Boolean constraints**: $B_{11} + B_{44} \leq 1$
  - More difficult in the other models
- Worker 3 is allocated a product with a number greater than the number of the product done by worker 2
  - **Variables for the workers**: $W_3 > W_2$
  - More difficult with Boolean constraints
    $B_{31} = 0 \wedge B_{32} \leq B_{21} \wedge B_{33} \leq B_{21} + B_{22} \wedge B_{34} \leq B_{21} + B_{22} + B_{23} \wedge \ldots$
  - Even more difficult with variables for the products

Solution:

Combination of different modellings: redundant constraints.

Modelling with Combined Approach

```
assignment(W1,W2,W3,W4) :-
        W1 in 1..4, W2 in 1..4, W3 in 1..4, W4 in 1..4,
        all_distinct([W1,W2,W3,W4]),
        element(W1,[7,1,3,4],WP1),
        element(W2,[8,2,5,1],WP2),
        element(W3,[4,3,7,2],WP3),
        element(W4,[3,1,6,3],WP4),
        P #= WP1 + WP2 + WP3 + WP4, P #>= 19,
        T1 in 1..4, T2 in 1..4, T3 in 1..4, T4 in 1..4,
        all_distinct([T1,T2,T3,T4]),
        element(T1,[7,8,4,3],TP1),
        element(T2,[1,2,3,1],TP2),
        element(T3,[3,5,7,6],TP3),
        element(T4,[4,1,2,3],TP4),
        P #= TP1 + TP2 + TP3 + TP4, P #>= 19,
        iff(W1,1,T1,1), iff(W1,2,T2,1), iff(W1,3,T3,1), iff(W1,4,T4,1),
        iff(W2,1,T1,2), iff(W2,2,T2,2), iff(W2,3,T3,2), iff(W2,4,T4,2),
        iff(W3,1,T1,3), iff(W3,2,T2,3), iff(W3,3,T3,3), iff(W3,4,T4,3),
        iff(W4,1,T1,4), iff(W4,2,T2,4), iff(W4,3,T3,4), iff(W4,4,T4,4),
        labeling([],[W1,W2,W3,W4]).
```

5 derivation steps in total

Aspects of Constraint Logic Programming

Theoretical

Logical Foundation – First-Order Logic

Conceptual

Sound Modeling

Practical

Efficient Algorithms/Implementations

Combination of different Solvers

Assets of Constraint Programming

Adaption and combination of existing efficient algorithms from

- **Mathematics**
  - Operations research
  - Graph theory
  - Algebra
- **Computer science**
  - Finite automata
  - Automatic proving
  - Artificial Intelligence
- **Economics**
- **Linguistics**

Application Domains

- **Modeling**
- **Executable Specifications**
- **Solving combinatorial problems**
  - Scheduling, Planning, Timetabling
  - Configuration, Layout, Placement, Design
  - Analysis: Simulation, Verification, Diagnosis
- **Artificial Intelligence**
  - Machine Vision
  - Natural Language Understanding
  - Temporal and Spatial Reasoning
  - Theorem Proving
  - Qualitative Reasoning
  - Robotics
  - Agents
  - Bioinformatics

Research Applications

- **Computer Science:** Program Analysis, Robotics, Agents
- **Molecular Biology, Biochemistry, Bioinformatics:** Protein Folding, Genomic Sequencing
- **Economics:** Scheduling
- **Linguistics:** Parsing
- **Medicine:** Diagnosis Support
- **Physics:** System Modeling
- **Geography:** Geo-Information-Systems

Commercial Applications

- **Lufthansa:** Short-term staff planning.
- **Hongkong Container Harbor:** Resource planning.
- **Renault:** Short-term production planning.
- **Nokia:** Software configuration for mobile phones.
- **Airbus:** Cabin layout.
- **Siemens:** Circuit verification.
- **Caisse d'epargne:** Portfolio management.