AGH University of Science and Technology

Essential Thinking.
Introduction to Constraint Programming
Example Problems IV

Antoni Ligęza

Faculty of EEACSE
Department of Automatics

AGH'2011
Kraków, Poland

Outline

1. References
2. Introduction
3. What is CSP?
4. Backtracking Search and Heuristics
5. Exploring Problem Structure
6. Tools: Prolog clp(fd) Library
7. Further Example Problems

References

1. Stuart J. Russel, Peter Norvig: *Artificial Intelligence. A Modern Approach.* Third Edition. Pearson, Prentice Hall, Boston, 2010. `http://aima.cs.berkeley.edu/`.
2. Ivan Bratko: *Prolog Programming for Artificial Intelligence*. Fourth Edition, 2011. Pearson, Addison Wesley, 2012. `http://www.pearsoned.co.uk/HigherEducation/Booksby/Bratko/`
3. Frank van Harmelen, Vladimir Lifschitz, Bruce Porter (Eds.): *Handbook of Knowledge Representation*. Elsevier B.V., Amsterdam, 2008.
4. Michael Negnevitsky: *Artificial Intelligence. A Guide to Intelligent Systems.* Addison-Wesley, Pearson Education Limited, Harlow, England, 2002.
5. Adrian A. Hopgood: *Intelligent Systems for Engineers and Scientists.* CRC Press, Boca Raton, 2001.
6. Joseph C. Giarratano, Gary D. Riley: *Expert Systems. Principles and Programming.* Fourth Edition, Thomson Course Technology, 2005.

AGH

References

1. George Polya: *How to Solve it?*. Princeton University Press, 1945; PWN 1993. `http://en.wikipedia.org/wiki/How_to_Solve_It`.
2. John Mason, Leone Burton, Kaye Stacey: *Thinking Mathematically*. Addison-Wesley, 1985; WSiP, 2005.
3. Mordechai Ben-Ari: *Mathematical Logic for Computer Science.* Springer-Verlag, London, 2001.
4. Michael R. Genesereth, Nils J. Nilsson: *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann Publishers, Inc., Los Altos, California, 1987.
5. Zbigniew Huzar: *Elementy logiki dla informatyków*. Oficyna Wyawnicza Politechniki Wrocławskiej, Wrocław, 2007.
6. Peter Jackson: *Introduction to Expert Systems.* Addison-Wesley, Harlow, England, 1999.
7. Antoni Ligęza: *Logical Foundations for Rule-Based Systems.* Springer-Verlag, berlin, 2006.

References

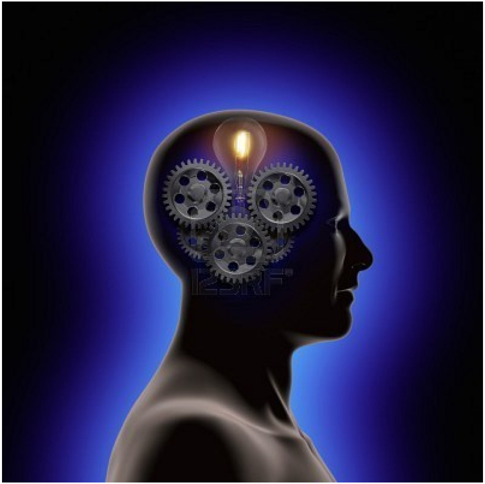
1. Krzysztof R. Apt: *Principles of Constraint Programming*. Cambridge University Press, Cambridge, UK, 2006.
2. Krzysztof R. Apt and Mark Wallace: *Constraint Logic Programming Using ECLiPSe*. Cambridge University Press, Cambridge, UK, 2006.
3. Rina Dechter: *Constraint Processing*. Morgan Kaufmann Publishers, San Francisco, CA, 2003.
4. Antoni Niederliński: *Programowanie w logice z ograniczeniami. Łagodne wprowadzenie do platformy ECLiPSe*. PKJS, Gliwice, 2010 (`http://www.pwlzo.pl/`).
5. Roman Bartak: *On-line Guide to Constraint Programming*. http://kti.mff.cuni.cz/ bartak/constraints/index.html.
6. `http://en.wikibooks.org/wiki/Prolog/Constraint_Logic_Programming`.
7. `http://eclipseclp.org/`

What is worth learning?

A bit provocative position statement

- Languages — enable communication and knowledge representation; Wieviel Sprachen du sprichst, sooftmal bist du Mensch; Goethe
- Problem Solving — analytical thinking; cross-curricular competencies,
- Learning — persistent learning, quick learning, focused learning, learning on-demand, ...
- Tools — finding and using proper tools for specific problems.
  - conceptual tools,
  - software tools.

Thinking — What is the Essence of it?



An Inspiration Example

A cryptoarithemtic problem

```
   SEND
+  MORE
------
  MONEY
```

An Intuitive Introduction

```
?- [library(clpfd)].

?- X #> 3.
X in 4..sup.

?- X #\= 20.
X in inf..19\/21..sup.

?- 2*X #= 10.
X = 5.

?- X*X #= 144.
X in -12\/12.

?- 4*X + 2*Y #= 24, X + Y #= 9, [X,Y] ins 0..sup.
X = 3, Y = 6.

?- Vs = [X,Y,Z], Vs ins 1..3, all_different(Vs), X = 1, Y #\= 2.
Vs = [1, 3, 2], X = 1, Y = 3, Z = 2.
```

Basic Constraints

clp(fd) — intro

```
?- [library(clpfd)].

Expr1 #>= Expr2 % Expr1 is larger than or equal to Expr2

Expr1 #=< Expr2 % Expr1 is smaller than or equal to Expr2

Expr1 #= Expr2 % Expr1 equals Expr2

Expr1 #\= Expr2 % Expr1 is not equal to Expr2

Expr1 #> Expr2 % Expr1 is strictly larger than Expr2

Expr1 #< Expr2 % Expr1 is strictly smaller than Expr2
```

Basic Constraints

clp(fd) — intro

```
?- [library(clpfd)].

#\ Q       % True iff Q is false

P #\/ Q      % True iff either P or Q

P #/\ Q      % True iff both P and Q

P #<==> Q    % True iff P and Q are equivalent

P #==> Q     % True iff P implies Q

P #<== Q     % True iff Q implies P
```

Basic Constraints

clp(fd) — intro

```
?Var in Domain

?Vars ins Domain

 Var is an element of Domain. Domain is one of:

   Integer
        Singleton set consisting only of Integer.
   Lower .. Upper
        All integers I such that Lower =< I =< Upper.
        Lower must be an integer or the atom inf, which den
        Upper must be an integer or the atom sup, which den
   Domain1 \/ Domain2
        The union of Domain1 and Domain2.
```

Basic Constraints

clp(fd) — intro

`label(+Vars)`

`labeling(+Options, +Vars)`

```
Labeling means systematically trying out values for
the finite domain variables Vars until all of them are grou
The domain of each variable in Vars must be finite.
Options is a list of options that let you exhibit
some control over the search process.
```

clp(fd) — intro

`indomain(+Var)`

```
Bind Var to all feasible values of its domain on backtracki
The domain of Var must be finite.
```

Basic Constraints

clp(fd) — intro

```
all_different(+Vars)

all_distinc(+Vars)

Vars are pairwise distinct.

The second command has stronger propagation
(can detect inconsistency).
```

Basic Constraints

clp(fd) — intro

```
sum(+Vars, +Rel, +Expr)
```

The sum of elements of the list Vars is in relation Rel to Expr, where Rel is #=, #\=, #<, #>, #=< or #>=. For example:

```
?- [A,B,C] ins 0..sup, sum([A,B,C], #=, 100).
A in 0..100,
A+B+C#=100,
B in 0..100,
C in 0..100.
```

The SEND+MORE=MONEY Example

Code

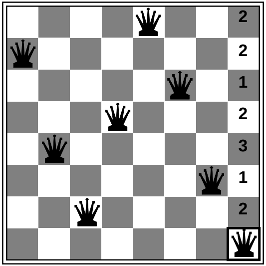
```
:- use_module(library(clpfd)).

puzzle([S,E,N,D] + [M,O,R,E] = [M,O,N,E,Y]) :-
        Vars = [S,E,N,D,M,O,R,Y],
        Vars ins 0..9,
        all_different(Vars),
                  S*1000 + E*100 + N*10 + D +
                  M*1000 + O*100 + R*10 + E #=
        M*10000 + O*1000 + N*100 + E*10 + Y,
        M #\= 0, S #\= 0.

?- puzzle(As+Bs=Cs), label(As).
As = [9, 5, 6, 7],
Bs = [1, 0, 8, 5],
Cs = [1, 0, 6, 5, 2] ;
false.
```

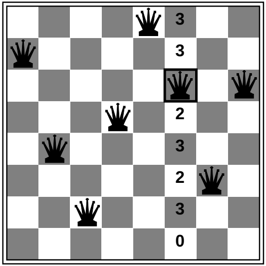
Bidirectional Factorial

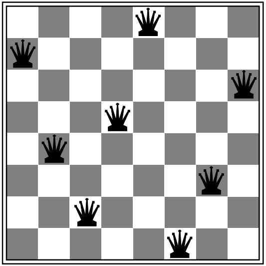
Code

```
:- use_module(library(clpfd)).

n_factorial(0, 1).
n_factorial(N, F) :- N #> 0, N1 #= N - 1, F #= N * F1, n_factorial

?- n_factorial(47, F).
F = 258623241511168180642964355153611979969197632389120000000000 ;
false.

?- n_factorial(N, 1).
N = 0 ;
N = 1 ;
false.

?- n_factorial(N, 3).
false.
```

N-Queens







Examples

A cryptoarithemtic problem

```
   SEND
+  MORE
------
  MONEY
```

Examples

A cryptoarithemtic problem

```
   DONALD
+  GERALD
  ------
   ROBERT
```

Examples

A cryptoarithemtic problem

```
      CROSS
+     ROADS
   ------
     DANGER
```

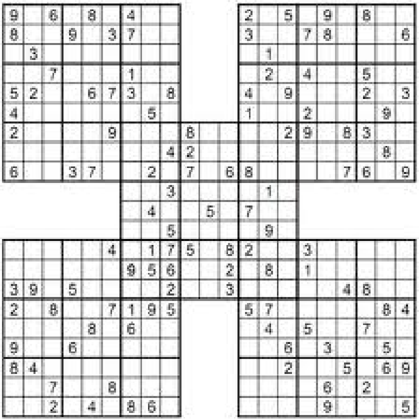
Enoxamples

A cryptoarithemtic problem

```
     TEN
+    TEN
   FORTY
 ------
   SIXTY
```

A Generic Problem Example

A Generic Problem Example



Another Example: The Zebra Puzzle

a) Norweg zamieszkuje pierwszy dom;
b) Anglik mieszka w czerwonym domu;
c) Zielony dom znajduje się po lewej stronie domu białego;
d) Duńczyk pija herbatkę;
e) Palacz Rothmansów mieszka obok hodowcy kotów;
f) Mieszkaniec żółtego domu pali Dunhille;
g) Niemiec pali Marlboro;
h) Mieszkaniec środkowego domu pija mleko;
i) Palacz Rothmansów ma sąsiada, który pija wodę;
j) Palacz Pall Malli hoduje ptaki;
k) Szwed hoduje psy;
l) Norweg mieszka obok niebieskiego domu;
m) Hodowca koni mieszka obok żółtego domu;
n) Palacz Philip Morris pija piwo;
o) W zielonym domu pija się kawę.