Advanced Programming Languages for AI H02A8

Gerda Janssens
Departement computerwetenschappen
200A.01.26
http://people.cs.kuleuven.be/~gerda

Overview Lecture 1

- About APLAI

- Intro: Constraint (Logic) Programming
  1. with ECLiPSe
  2. Prolog
  3. Constraint programming terminology
  4. Running examples

Starting Point

- Knowledge of Prolog
- Background in AI

    constraint propagation

    search

    condition-action rules

Aim

Study more programming languages and tools useful in the AI context

Selection for 15-16

- Constraint (Logic) Programming
  - ECLiPSe (ILOG (IBM) , OPL, Cosytec)
- Rule Based Systems
  - Constraint Handling Rules (CHR)
  - Jess (rule engine, Java, Business rules)
- (Local search)

Format 15-16

- Lectures (2 studypoints)
  - Different systems/languages
  - Different approaches

  - APLAI slides + wiki on Toledo
  - Additional material: How to use it?? http://4c.ucc.ie/~hsimonis/ELearning/index.htm
- Assignment (2 studypoints)
  - Groups of 2
  - Minimal requirements + optional part

15-16 Assignment

- For the problems at hand,
- Given the different approaches studied in APLAI,
- Explore !!!

- Minimal requirements: some tasks
- Optional: an additional task

Constraint (Logic) Programming

with ECLiPSe

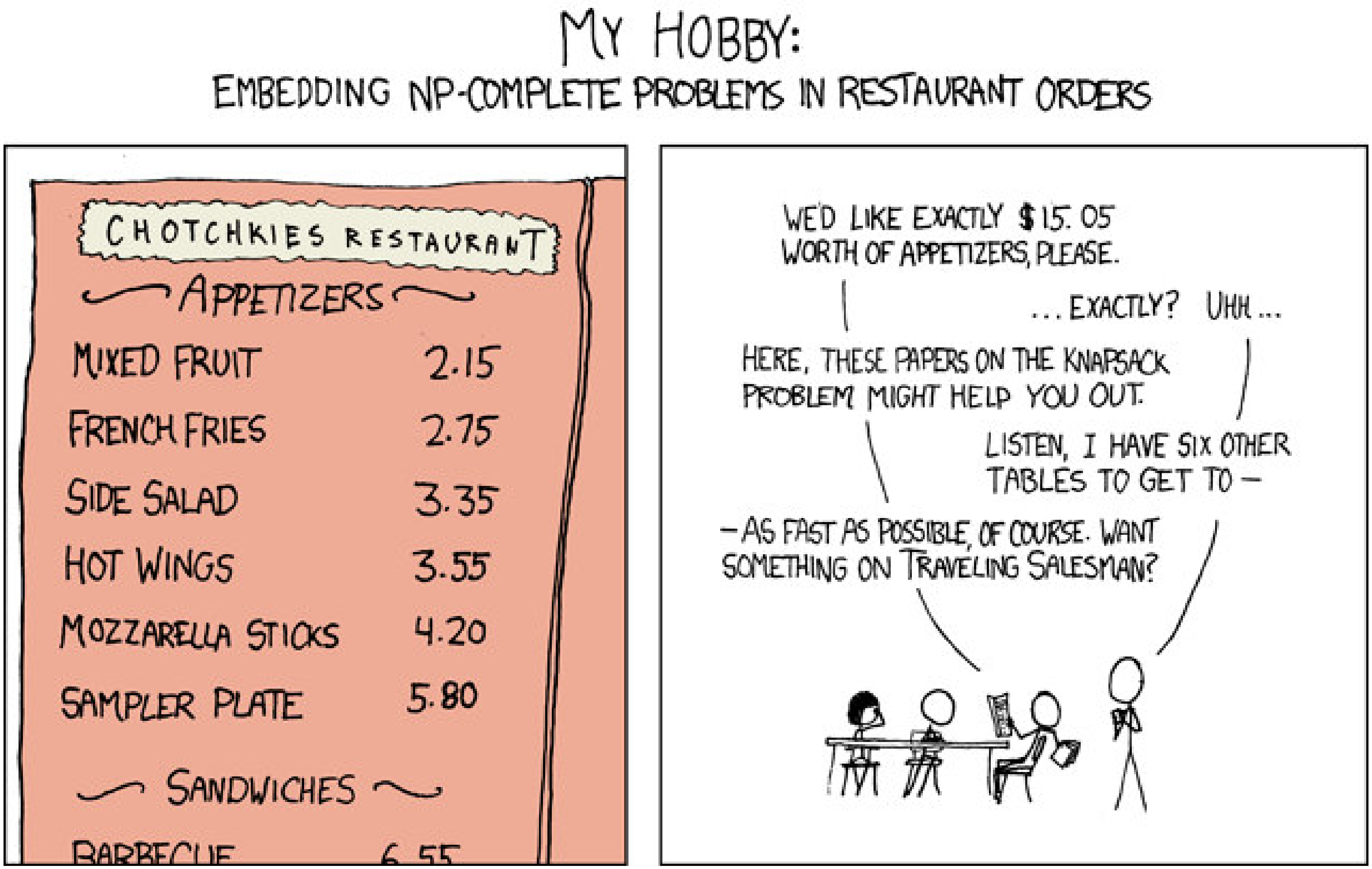
http://eclipseclp.org

constraints embedded in Prolog

Topics

- **What** constraint satisfaction problems and constrained optimisation problems are and **how** they are solved using constraint programming techniques.
- How to deal with these problems in ECLiPSe
- What support ECLiPSe provides for general and domain specific methods
- How to solve constraint satisfaction and constrained optimisation problems in ECLiPSe

xkcd problem



ECLiPSe subset-sum program

```
:- lib(ic).
solve(Amounts) :-
  Total = 1505,
  Prices = [215, 275, 335, 355, 420, 580],
  length(Prices, N), length(Amounts, N),

  Amounts::0..Total//min(Prices),
  Amounts * Prices #= Total,

  labeling(Amounts).
```

Impact

- Siemens: configuration of an Railway Interlocking system
- Use of CLP:
  - 60% reduction of the intitial development cost
  - 88% reduction of the yearly maintenance cost
- Declarative reading of the constraints

Starting Point

- Prolog issues
  - Unification, backtracking search, arithmetic
  - Passive vs Active constraints
- Constraint issues
  - Constraint propagation: again Passive/Active
  - Search: backtracking search and
    branch and bound search

1. ECLIPSE

ECLiPSe history

- European Computer-Industry Research Center ECRC Munich (1984): development of advanced reasoning techniques for practical problems
- CHIP (1988) incorporated constraint satisfaction into Linear Programming by using finite domain variables and relying on top-down search techniques
- ECLiPSe (1991) = CHIP + (database and parallel programming facilities)

ECLiPSe history

- 1997 interface to an external linear and mixed integer programming package
- 1999 commercial rights to IC-Parc (London)
- 2004 bought by Cisco Systems
- Still freely available for teaching and research (production planning, transportation, scheduling, bioinformatics, optimisation of contracts …);
  commercial optimisation SW by Cisco

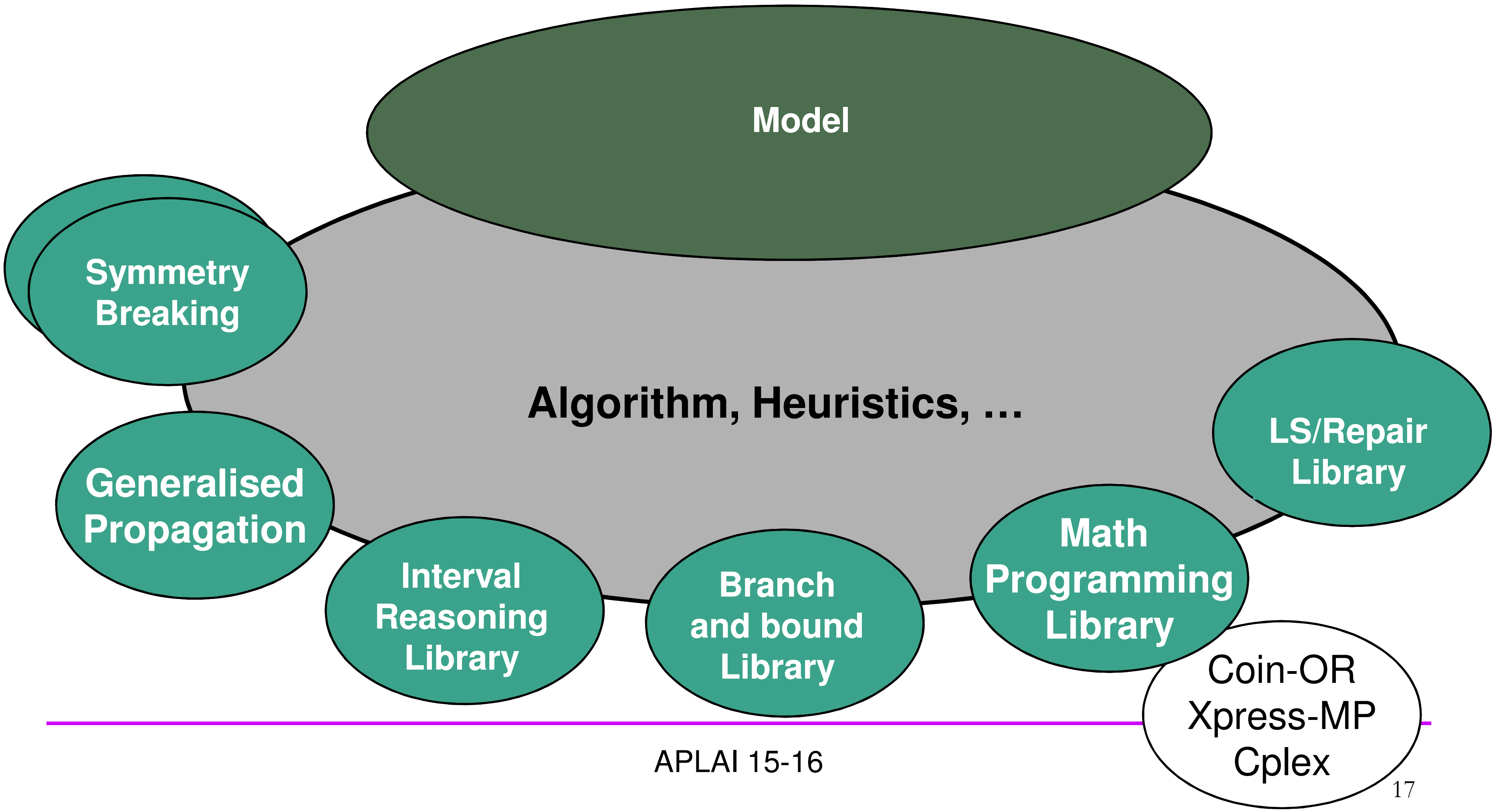
Motivation

ECLiPSe attempts to support - in some form or other - the most common techniques used in solving Constraint (Optimization) Problems:

- CP – Constraint Programming
- MP – Mathematical Programming(also LP,IP)
- LS – Local Search
- and combinations of those

ECLiPSe is built around the CLP (Constraint Logic Programming) paradigm

ECLiPSe for Modelling and Solving



ECLiPSe Usage: www.eclipseclp.org

Applications

- Developing problem solvers
- Embedding and delivery

Research

- Teaching
- Prototyping solution techniques

CLP Books

- Constraint Logic Programming using ECLiPSe, Krzysztof Apt and Mark Wallace. Cambridge University Press, 2007. ISBN-13 978-0-521-86628-6
  Plaatsingsnummer: **6 681.3*D32 ECLI/2007**
  A practical introduction to constraint programming and to ECLiPSe
- Principles of Constraint Programming, Krzysztof R. Apt, Cambridge University Press, 2003. ISBN 0 521 82583 0

Online material

- Explore the website: examples, manuals (tutorial, reference, libraries)
- http://eclipseclp.org/reports/index.html mentions books, introductory material, applications
- *ECLiPSe - from LP to CLP* by Joachim Schimpf and Kish Shen. Theory and Practice of Logic Programming / Volume 12 / Special Issue on Prolog Systems 1-2, pp 127 - 156. Copyright Cambridge University Press 2011,

CLP Books

- Programming with Constraints: an Introduction, Kim Marriott and Peter J. Stuckey, MIT Press, 1998.
- The OPL Optimization Programming Language, Pascal Van Hentenryck, MIT Press, 1999. An industrial implementation of OPL is available from the international software company ILOG. http//www.ilog.fr/products/optimization

Books

- Constraint-Based Local Search, Pascal Van Hentenryck and Laurent Michel, MIT Press, 2005. ISBN-10:0-262-22077-6 combinatorial optimization problems; combines constraint programming and local search; a programming language, COMET, that supports both modeling and search abstractions in the spirit of constraint programming.

2. PROLOG

Some Prolog links

- FAQ van Prolog:
  http://www.logic.at/prolog/faq/faq.html
- Online tutorials:
  Bartak , Fisher, and P. Blackburn et al.
- Books on Prolog
- The World Wide Web Virtual Library: Logic Programming

Back to Prolog: unification

```
[eclipse 2]: p(k(Z,f(X,b,Z)))=p(k(h(X),f(g(a),Y,Z))).

[eclipse 3]: p(k(Z,f(X,b,Z)))=p(k(h(X),f(g(Z),Y,Z))).
```

Use Martelli-Montanari algorithm

What with occur check??

What on failure of unification?

append.pl

```
% app(Xs, Ys, Zs) :- Zs is the result of
%     concatenating the lists Xs and Ys.

app([], Ys, Ys).
app([X | Xs], Ys, [X | Zs]) :- app(Xs, Ys, Zs).
```

Backtracking

```
[eclipse 1]: [append].
append.pl  compiled traceable 276 bytes in 0.00 seconds

Yes (0.00s cpu)
[eclipse 2]: append(Xs,Ys,[1,2,3]).

Xs = []
Ys = [1, 2, 3]
Yes (0.00s cpu, solution 1, maybe more) ? ;

Xs = [1]
Ys = [2, 3]
Yes (0.00s cpu, solution 2, maybe more) ? ;

Xs = [1, 2]
Ys = [3]
Yes (0.00s cpu, solution 3, maybe more) ? ;

Xs = [1, 2, 3]
Ys = []
Yes (0.00s cpu, solution 4)
[eclipse 3]:
```

Pure Prolog and declarative programming

- Declarative interpretation (what is being computed;meaning): a Prolog program is a set of formulas (first order logic; semantics)
- Procedural interpretation (how computation takes place; method): mgu, SLD resolution
- Declarative programming: Programs can be interpreted in a natural way as formulas in some logic. Then the results of the program computations follow logically from the program. Towards executable specifications.

Arithmetic in Prolog

- Infinitely many integer constants, also floats

  `0   -3   19      0.0 -1.344  3.333`

- Given a tree (node/3, empty): sum of its values

```
sum(empty,0).
sum(node(L,R,W), Total):-
            sum(L,TotalL),
            sum(R,TotalR),
            Total is TotalL + TotalR + W.
```

- `is/2` **the arithmetic evaluator**
  - **evaluates** the operand at the rhs
  - **unifies** the result with the operand at the lhs
  - has to be at the right place

=/2 and is/2

- `[eclipse 1]: X is 1 + 2 .`
- `[eclipse 2]: X = 1 + 2 .       % ?- X = pair(1,2).`
- `[eclipse 3]: X = 1 + 2, Y is X .`

```
+  -(binary) -(unary) * //  mod
The relation between integer division //
    and modulus operation mod is as follows:
       X =:= (X mod Y) + (X // Y) * Y
```

Arithmetic comparison predicates

- Less than `<`
- Less than or equal `=<`
- Equality `=:=`
- Disequality `=\=`
- Greater than or equal `>=`
- Greater than `>`

Ordered/2

```
% ordered(Xs) :- Xs is an =<-ordered list of numbers.
ordered([]).
ordered([H | Ts]) :- ordered(H, Ts).

% ordered(H, Ts) :- [H|Ts] is an =<-ordered list of numbers.
ordered(_, []).
ordered(H, [Y | Ts]) :- H =< Y, ordered(Y, Ts).


[eclipse 3]: ordered([1,X,6]).
instantiation fault in 1 =< X
Abort
```

Arithmetic constraints: passive

- Compare f(a,X) = f(Y,b) with 3*X < Y +2
- Constraints: restrict the values of variables
- Using comparison predicates : </2 is an arithmetic constraint
- Evaluation of a constraint
  - =/2 : can affect the variables in the constraint: active
  - </2 : cannot affect them/only for testting: passive

What if </2 were an active arithmetic constraint???

```
[eclipse 3]: ordered([2,X,2]).
X = 2
```

This behaviour is possible when using the ECLiPSe's ic library.

3. CONSTRAINT PROGRAMMING

Constraint Programming: a primer

- What constraints do you already know?
- A constraint is an atomic formula.
- We assume that each constraint has at least one variable.
- Each interpretation associates with a constraint a subset of the Cartesian product of the domains of its variables.
- The constraint $X < Y$ over the set of integers with as interpretation $\{ (a,b) \mid a,b \in Z \text{ and } a < b\}$

Satisfiable and solved constraints

- An assignment of values to the constraint variables satisfies a constraint (is a solution to a constraint) if the used sequence of values belongs to its interpretation.
- A constraint is satisfiable if some assignment to its variables satisfies it and is unsatisfiable if no such assignment exists.
- Over the set of reals, $X > X+Y$ is satisfiable; over the set of natural numbers, it is unsatisfiable.
- A constraint is solved if all assignments of values to its variables satisfy it. $X + Y > 0$ over the natural numbers.

Constraint satisfaction problem

- A constraint satisfaction problem, a CSP, is
  a finite sequence of *variables*,
  each ranging over a possibly different domain, and
  a finite set of *constraints*,
  each on a subsequence of the considered
  variables.
- The variables in a CSP: decision variables.
- CSPs are considered in the context of an interpretation.
- A solution to a CSP is an assignment of values to its variables that satisfies each constraint.

Consistent and solved CSPs

- A CSP is consistent (or feasible) if it has a solution and
  is inconsistent (or infeasible) if it does not.
- A CSP is solved if each of its constraints is solved.
- A CSP is failed if one of its domains is empty or one of its constraints is unsatisfiable.

- ***If a CSP is failed then it is inconsistent, but not necessarily the other way around.***

- Two CSPs are equivalent if they have the same set of solutions.

Common classes of constraints

- Uniquely determined by the considered predicates and function symbols.
- Interpretation is clear from the context.
- Classes:
  - Equality, disequality
  - Boolean
  - Linear
  - Arithmetic

Equality and disequality

- Two predicate symbols, equality = and disequality ≠
- Variables are interpreted over arbitrary domains
- What do you know about the constraints

  (solved? (un)satisfiable?)

  $x = x$ and $x \neq x$

  $x = y$ and $x \neq y$
- What is the solution(s) of the CSP

  $\langle x = y, y \neq z, z \neq u;$

  $x \in \{a,b,c\}, y \in \{a,b,d\}, z \in \{a,b\}, u \in \{b\}\rangle$

Boolean constraints

- *Constants*: true and false
- *Function* symbols: negation ~, conjunction ^, disjunction V.
- The resulting terms (s◇t) are Boolean expressions
- *Predicate* symbol: =/2 s,t Boolean expressions
  s = t s (shorthand for s = true)
- The variables are interpreted over the set of truth values {0,1}; the interpretation of the connectives is given by the standard truth tables.
- (~x) ^ (y V z) = true with as interpretation …

Linear constraints

- Over the set of integers, or the set of reals, or over a subset of one of these sets (usually an interval).
- A fixed set of numeric constants representing reals or integers.
- Linear constraint, s op t, where
  op either $<$, $\leq$, $=$, $\neq$, $\geq$, or $>$ (for reals)
  s and t linear expressions such as
  $4.x + 3.y - y$

Arithmetic constraints

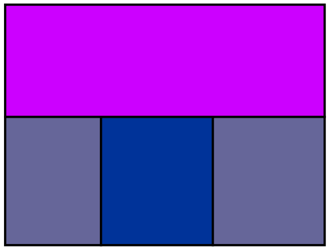
- Only difference w.r.t. linear constraints lies in the use of the multiplication symbol.
- Now also $4.x + x.y - x < y.(z+3) - 3.u$

4. RUNNING EXAMPLES

CSP examples: todo model them

- Map colouring
- SEND+MORE = MONEY
- N-queens problem

Map colouring



- A finite set of regions
- A (smaller) set of colours
- A neighbour relation between pairs of regions

Associate a colour with each region so that no two neighbours have the same colour!

Decision variables? Domains? Constraints?

SEND+MORE = MONEY

- Cryptarithmetic problem: digits are replaced by letters …
- Replace each letter by a different digit such that the sum is ok.
- Modeled by which kind of constraints???

SMM: representation 1

- 1 equality constraint

$$1000.S + 100.E + 10.N + D$$
$$+ 1000.M + 100.O + 10.R + E$$
$$= 10000.M + 1000.O + 100.N + 10.E + Y,$$

- 2 disequality constraints: $S \neq 0$, $M \neq 0$
- And 28 disequality constraints $x \neq y$ for x,y ranging over the set {S,E,N,D,M,O,R,Y}

SMM representation 2

- Instead of 1 equality: five simpler ones
- Introducing per column a carry variable ranging over {0,1}

$$D + E = 10.C1 + Y,$$
$$C1 + N + R = 10.C2 + E,$$
$$C2 + E + O = 10.C3 + N,$$
$$C3 + S + M = 10.C4 + O,$$
$$C4 = M$$

SMM representation 3

- Instead of the 28 disequality constraints
- alldifferent([S,E,N,D,M,O,R,Y]).

- Book p. 196: built-in of the ic_library
- From within ECLiPse: query
  help(alldifferent).
  help(ic:alldifferent/1).

SMM as Integer Linear Programming

- Finding (optimal) integer solutions to linear constraints over the reals.
- For each variable x in {S,E,N,D,M,O,R,Y} and each digit d: a 0/1 variable is_xd
- Each variable x from {S,E,N,D,M,O,R,Y} takes one value: $\sum_d$ is_xd = 1 (d from 0 to 9)
- All different values: for all d
  $\sum_x$ is_xd ≤ 1 (x ∈ {S,E,N,D,M,O,R,Y})
- x = $\sum_d$ (d . is_xd) (d from 0 to 9)

N-queens problem

- Place n queens on the n x n chess board, where $n \geq 3$, so that they do not attack each other.
- Representation 1: using n^2 0/1 variables x_ij, where $i,j \in [1..n]$, representing one field
- Representation 2: using linear constraints and n variables x_i with domain [1..n]. x_i denotes the position of the queen in the ith column.

Representation 1

- Exactly one queen per row:
  for each i $\sum_j x\_ij = 1$ (j from 1 to n)
- Exactly one queen per column:
  for each j $\sum_i x\_ij = 1$ (i from 1 to n)
- At most one queen per diagonal:
  $x\_ij + x\_kl \leq 1$ for $i,j,k,l \in [1..n]$ such that $i \neq k$ and $|i-k| = |j-l|$
- Disadvantage??

Representation 2

- n variables x_i with domain [1..n], denoting the position of the queen in the ith column.
- Implies : no two queens in the same column.
- For $i \in [1..n]$ and $j \in [1..i-1]$
  - At most one queen per row: $x_i \neq x_j$
  - At most one queen per SE-NW diagonal $x_i - x_j \neq i - j$
  - At most one queen per SW-NE diagonal $x_i - x_j \neq j - i$

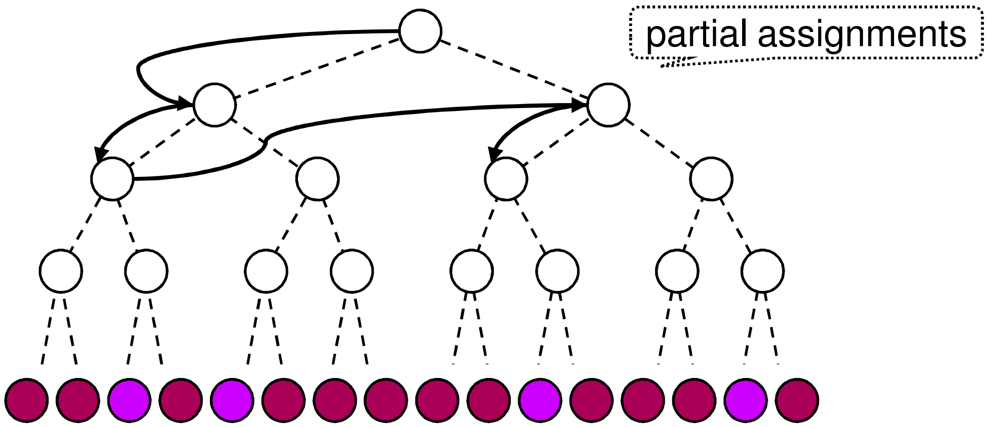
Constraint Optimization Problems: examples

- Also a cost function mapping the values of the x_i variables to a real number
- Optimal: here minimal
- Huge area:
  - The knapsack problem
  - A coins problem
  - The facility location problem

Solving CSPs and COPs

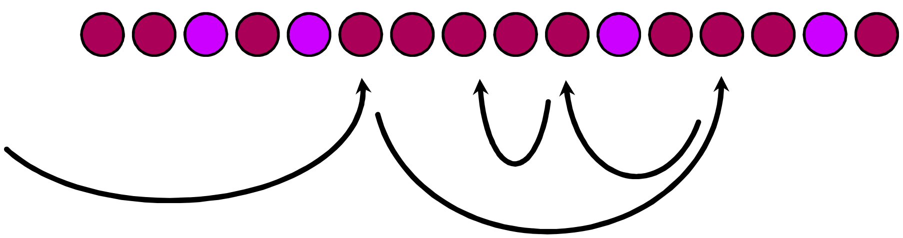
- Domain specific methods are preferred
  - Systems of linear equations over reals : methods from linear algebra
  - Systems of linear inequalities over reals: methods from the area of Linear Programming
- Also general methods are needed
  - Developed in area of Constraint Programming
  - Based on search
    - Local search
    - Top-down search

Exploring search spaces



CLP Tree search:

- constructive
- partial/total assignments
- systematic
- complete or incomplete



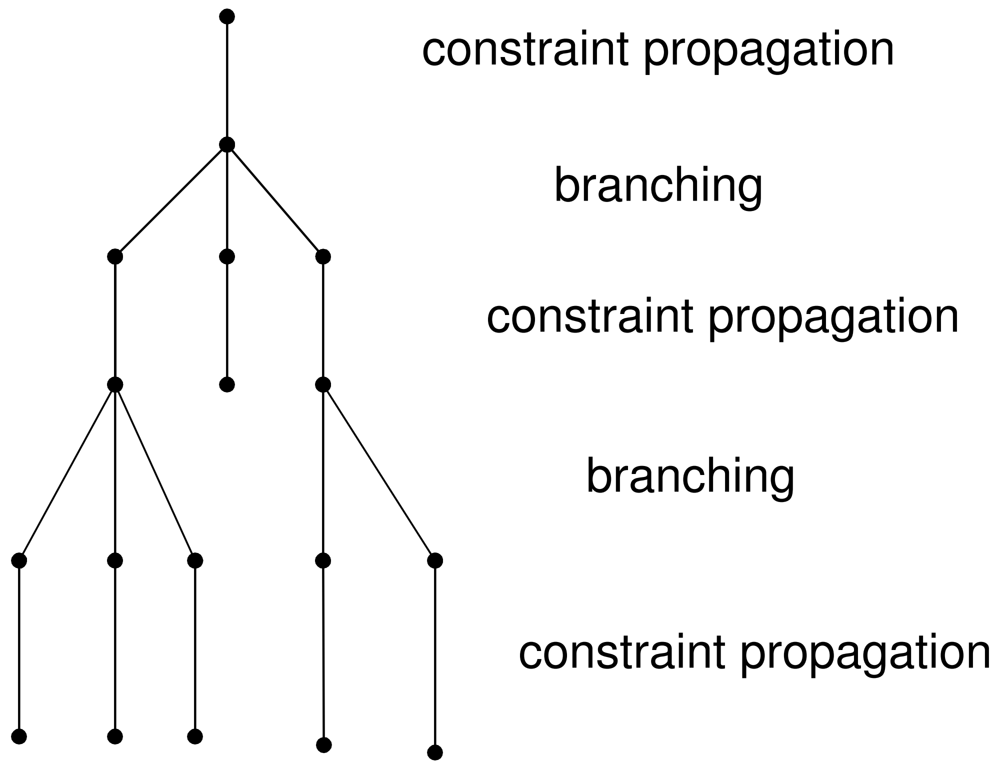
“Local” search:

- move-based (trajectories)
- only total assignments
- usually random element
- incomplete

Top-down search for CSPs

- Combined with a branching (splitting) strategy and constraint propagation.
- *Role of branching?*
- To split a CSP into 2 or more CSP's whose union is equivalent to the initial CSP
- *Role of propagation?*
- To transform a CSP into one that is equivalent but simpler
- *Alternating* propagation and branching

Search tree generated on the fly



Leaves: CSP's that are solved or obviously inconsistent (e.g. empty domain)
Backtracking search

Organisation of backtracking search

- Ordering the decision variables according to some variable choice heuristics
- Branching through splitting the domain of a variable, e.g. labelling splits a finite domain into the singleton domains
- Some value choice heuristics for the ordering of the values in each domain, e.g. bisection which is used for interval domains or finite set domains.

Domain reduction by propagation

- Removal of values that do not participate in any solution
- One reduction can trigger other reductions such that reduction can be further improved.
- Implied constraints: do not alter the set of solutions, but can lead to a smaller search tree as they can be used during propagation
- Complete seach

Branch and bound search for COPs

- Wanted: solution with a minimal value of the cost function
- During backtracking search keep the currently best value of the cost function
- Prune the seach tree by identifying nodes under which no solution with a smaller cost can be present

Towards Constraint Programming

- Programming language with support for generating and solving CSPs and COPs.
- Support for CSP variables, unknowns as in mathematics, and their domains by built-in facilities
- Support for general methods such as backtracking and branch and bound search, various variable and value choice heuristics.
- Specialised, domain specific methods in the form of constraint solvers integrated with the general methods