An on-line course on constraint programming

TeachLP 2004

Christine Solnon

LIRIS, CNRS FRE 2672 / University Lyon 1

Context: e-MIAGE

- e-MIAGE = on-line version of MIAGE
- MIAGE = a popular French training
  - on « Information Systems for Company Management »
  - delivered in 20 French universities
    $\leadsto$ national pedagogical programme
  - duration = 3 years
    $\leadsto$ Master's degree
- National decision to create e-MIAGE in 2001
  $\leadsto$ First e-trainees in September 2003

Context: e-MIAGE

- For whom ?
  - trainees who cannot physically attend courses (handicapped persons, foreigners, ...)
  - professionals that are continuing education
  - ... and also “traditional” MIAGE trainees (free access to online courses)
- How ?
  - via Internet
  - no imposed rhythm... but a course unit must be trained during a semester
  - tutoring by e-mail + periodical meetings

Context: e-MIAGE

National pedagogical programme of the MIAGE training

- 50 course units
  - 1 course unit $\leadsto$ 40 hours and 3 ECTS
- 1 of these 50 course units is « Artificial Intelligence »
- Decomposition of the AI course unit in 36 sessions
  - 2 introductory sessions,
  - 8 sessions about logic,
  - 5 sessions about Prolog,
  - 7 sessions about constraint programming,
  - 2 sessions about planning,
  - 4 sessions about ontologies,
  - 3 sessions about machine learning,
  - 5 sessions about expert systems.

7 sessions about constraint programming

- The goal is not to train experts...
  ... but to initiate students to using CP to solve problems
  - know what is a CSP
  - be able to model a problem as a CSP
  - know the basic principles of constraint solving
  - be able to use a CP language to solve a CSP
- Programming language = Gnu-Prolog
  - Integrates a constraint solver over finite domains
  - Free and easy to install on all computers/OS
  - On-line user's manual

  $\leadsto$ go deeper into the practice of Prolog

Organization of the 7 course sessions

- Session 1: Course session on « Constraints and CSPs »
  - What is a constraint ?
  - What is a CSP ?
  - First example: the n-queens problem

    $\leadsto$ Different models
  - Second example: the stable marriage problem (Gent & Prosser 2002)
    - n men and n women
    - each man/woman gives a preference list of women/men
    - preference lists may be uncomplete and may contain ties
    - marry men and women so that they are "stable"

Organization of the 7 course sessions

- Session 1: Course session on « Constraints and CSPs »
- Session 2: Training session on « CSPs modeling »
  - 5 exercises:
    1. Computing coins returned back by a slot machine
       $\rightsquigarrow$ integer variables and linear integer constraints
       $\rightsquigarrow$ add an optimization criterion
    2. Map coloring problem
       $\rightsquigarrow$ add an optimization criterion
    3. Salt and Mustard Puzzle from Lewis Caroll
       $\rightsquigarrow$ 2 different modelings
    4. Crypt-arithmetic puzzle « SEND + MORE = MONEY »
       $\rightsquigarrow$ 2 different modelings
    5. Zebra puzzle
  - For each exercise:
    - the problem is described in natural language,
    - the student is asked to model the problem by means of (X, D, C),
    - the student may ask for some help by clicking on a link.

Organization of the 7 course sessions

- Session 1: Course session on « Constraints and CSPs »
- Session 2: Training session on « CSPs modeling »
- Session 3: Course session on « CSPs solving »

  - Restricted to complete algorithms / finite domains
    - « Generate and Test » algorithm
      $\rightsquigarrow$ Search space of a CSP
    - « Simple Backtrack » algorithm
    - « Look-ahead » algorithm
      $\rightsquigarrow$ Local consistencies and domain filtering
    - Integration of heuristics
      $\rightsquigarrow$ Variable and value orderings
  - For each algorithm
    - informal description
    - imperative (recursive) pseudo-code
    - illustration on the 4-queens problem

Organization of the 7 course sessions

- Session 1: Course session on « Constraints and CSPs »
- Session 2: Training session on « CSPs modeling »
- Session 3: Course session on « CSPs solving »
- Session 4: Training session on « Writing CSP solvers »

  - Implement the algorithms of session 3 in Prolog

    $\leadsto$ better understanding of enumeration and propagation

    ... and improved practice of Prolog
  - How to guide trainees?
  - Restriction to binary CSPs

    $\leadsto$ description of binary CSPs by 2 predicates

Cours de PPC - session 4 - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

file:///home/csolnon/transparents/conferences/teachLP04/session4.htm#grd1

1 - Description of binary CSPs in Prolog

To write a constraint solver, one first have to define data structures for describing the CSP to solve. Here, we shall restrict our attention to binary CSPs, so that every constraint involves two variables, and we shall describe a binary CSP by means of the two following predicates :

- *variables/1* describes the variables of the CSP, with their associated domains :
  - Template :*variables(?list)*
  - Description :

    variables(L) unifies *L* with a list of variable names, each variable name being followed by the ':' symbol and a domain, that is the list of the values the variable can take. Hence, if the CSP contains *n* variables *{x1, x2, ..., xn}* , then *L* unifies with the list *[x1:D1, x2:D2, ..., xn:Dn]* such that *D1* is the list of values contained in *D(x1), ..., Dn* is the list of values contained in *D(xn)*.

    Be careful: the names of the variables of the CSP (*x1, x2, ..., xn*) should not be Prolog variables, but constant terms... starting by lower case letters.

  - Example on the 4-queens CSP:

```
|?- variables(L).
L = [x(1):[1,2,3,4], x(2):[1,2,3,4], x(3):[1,2,3,4], x(4):[1,2,3,4]]
yes
```

Done

Cours de PPC - session 4 - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

file:///home/csolnon/transparents/conferences/teachLP04/session4.htm#grd1

- *consistent/2* describes the binary constraints of the CSP:
  - Template: *consistent((+term,+term),(+term,+term))*
  - Description :

    consistent((Xi,Vi),(Xj,Vj)) succeeds if the partial assignment *{(Xi,Vi),(Xj,Vj)}* is consistent, that is, if the binary constraint holding between variables *Xi* and *Xj* is ensured for values *Vi* and *Vj*.
  - Example on the 4-queens CSP:

```
|?- consistent((x(1),2),(x(4),2)).
no

| ?- consistent((x(1),1),(x(2),3)).
yes
```

You can download Prolog code describing

- the 4-queens CSP,
- the map coloring CSP,
- the stable marriage CSP.

You will use them to test your constraint solvers.

Done

Cours de PPC - session 4 - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

file:///home/csolnon/transparents/conferences/teachLP04/session4.htm#grd2

2 - Programming "Generate and Test" in Prolog

2.1 - Generating complete assignments

To implement the "generate and test" algorithm studied during point 1 of course session 3, one first has to define a predicate that, given a list of variables and domains of the CSP, generates a complete assignment. To this aim, you have to write the *generate/2* predicate described below:

- Template: *generate(+list_of_terms, ?list_of_terms)*

- Description: *generate(V,A)* succeeds if
  - *V* is a list of variable names, each variable name being associated with a domain, and
  - *A* unifies with a complete assignment of the variables of *V*, that is a list of couples *(variable,value)*.

- Example: Let us consider a CSP with 3 variables, *a*, *b* and *c*, such that *D(a)={0,1}*, *D(b)={2,4,6}* and *D(c)={0}*. All complete assignments for this CSP may be generated as follows:

```
| ?- generate([a:[0,1], b:[2,4,6], c:[0]],A).
A = [(a,0),(b,2),(c,0)] ? ;
A = [(a,0),(b,4),(c,0)] ? ;
A = [(a,0),(b,6),(c,0)] ? ;
A = [(a,1),(b,2),(c,0)] ? ;
A = [(a,1),(b,4),(c,0)] ? ;
A = [(a,1),(b,6),(c,0)] ? ;
no
```

Done

Cours de PPC - session 4 - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

file:///home/csolnon/transparents/conferences/teachLP04/session4.htm#grd2

Once you have programmed the *generate/2* predicate, validate it on the small example above. You may also test it on the 4-queens CSP. To do this, load the Prolog code describing the 4-queens CSP and ask Prolog to generate all complete assignments for this CSP as follows:

```
| ?- variables(V), generate(V,A).
```

Prolog should enumerate all complete assignments:

```
A = [(x(1),1),(x(2),1),(x(3),1),(x(4),1)] ? ;
A = [(x(1),1),(x(2),1),(x(3),1),(x(4),2)] ? ;
A = [(x(1),1),(x(2),1),(x(3),1),(x(4),3)] ? ;
A = [(x(1),1),(x(2),1),(x(3),1),(x(4),4)] ? ;
etc...
```

2.2 - Testing the consistency of an assignment

Once you have written and validated the *generate/2* predicate, you have to define a predicate that checks the consistency of a given assignment. To do this, you have to write the *test/1* predicate:

- Template: *test(+list_of_terms)*
- Description : *test(A)* succeeds if *A* is a consistent assignment. Let us recall that all constraints a binary so that an assignment *A* is consistent if for every pair of cpuples *(Xi,Vi)* and *(Xj,Vj)* that belong to A, the assignment *{(Xi,Vi),(Xj,Vj)}* is consistent.
- Example for the 4-queens CSP:

Done

Cours de PPC - session 4 - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

file:///home/csolnon/transparents/conferences/teachLP04/session4.htm#grd2

- Example for the 4-queens CSP:

```
| ?- test([(x(1),1),(x(2),3),(x(3),2)]).
no

| ?- test([(x(1),2),(x(2),4),(x(3),1),(x(4),3)]).
yes
```

Check that your predicate is well defined by testing it on the example below (don't forget to load before the Prolog code describing the 4-queens CSP).

2.3 - Generate and test

Using the *generate/2* and *test/1* predicates, you can now write the *generateAndTest/1* predicate:

- Template: *generateAndTest(?list_of_terms)*
- Description : *genereAndTest(A)* unifies *A* with a solution of the CSP described by the *variables/1* and *consistent/2* predicates.
- Example on the 4-queens CSP:

```
| ?- generateAndTest(A).
A = [(x(1),2),(x(2),4),(x(3),1),(x(4),3)] ? ;
A = [(x(1),3),(x(2),1),(x(3),4),(x(4),2)] ? ;
(10 ms) no
```

Done

Organization of the 7 course sessions

- Session 1: Course session on « Constraints and CSPs »
- Session 2: Training session on « CSPs modeling »
- Session 3: Course session on « CSPs solving »
- Session 4: Training session on « Writing CSP solvers »
- Session 5: Course session on « CP with Gnu-Prolog »
  - Brief overview of different existing CP languages
  - Description of CP predicates of Gnu-Prolog
    ... with many links to the online users' guide
  - First example: the n-queens problem
  - Second example: the stable marriage problem

Organization of the 7 course sessions

- Session 1: Course session on « Constraints and CSPs »
- Session 2: Training session on « CSPs modeling »
- Session 3: Course session on « CSPs solving »
- Session 4: Training session on « Writing CSP solvers »
- Session 5: Course session on « CP with Gnu-Prolog »
- Sessions 6 and 7: Training sessions on « CSPs solving with Gnu-Prolog »
  - Use CP predicates of Gnu-Prolog to solve CSPs modeled in session 2
  - For each problem:
    - Recall the definition of the problem in natural language.
    - Recall the CSP modeling proposed in session 2.
    - Identify the main predicates to define...
    - ... and ask the students to write them.

Conclusion

- First training on this course in 2003/2004
  - 13 trainees (professionals continuing education)
  - Very few questions, 2 errors detected (!)
  - The tutor did not check trainees solutions
    ... solutions available through a click !
  - Students seem satisfied, and most of them succeeded at the final test
- Many thanks to Roman Bartak, Edward Tsang, ... and all others
- The course (in french) is available at
  http://www710.univ-lyon1.fr/∼csolnon/Site-PPC/e-miage-ppc-som.htm
  and it should be translated in English (soon?)