Constraint Programming within CLIPS

Jean-Marc Labat and Michel Futtersack

LIP6 Lab, Paris VI University, France

Jean-Marc.Labat, Michel.Futtersack@poleia.lip6.fr

Abstract

CCP (CLIPS Constraint Programming) is a generic knowledge base which enables CLIPS to solve CSPs (Constraint Satisfaction Problems). CCP is composed of three modules. The MAIN module contains the top level loop including the I/O rules. The SEARCH module implements a classical Chronological Backtrack algorithm. The PROPAGATION module implements the Forward Checking algorithm. CCP has been extensively tested with well known puzzles as N-queens, graph colouring, crosswords, etc. To implement any particular problem, the users only need to declare each variable with its domain and they have to define specific domain description and rules of propagation. So that all specific knowledge is encapsulated in the PROPAGATION module. CCP is a generic tool that can be embedded in a knowledge based system like a work horse for local CSPs solving. CCP is compact, since it contains only a dozen rules. Moreover, CCP can be seen as a pedagogical tool illustrating many advanced features of CLIPS (logical dependencies, forall, exists conditional elements, the built function and a control strategy based on the focus stack)

Keywords

CCP, CSP, Constraint Programming

1. Introduction

A wide variety of problems can be described as Constraint Satisfaction Problems (CSPs) : time table, cross-words, graph colouring, job shop scheduling, SAT problems. Combinatorial optimization such as the assignment problem, the vehicle routing problem or the knapsack problem can be also considered as CSP problems by transforming Max f(x) (respectively Min) into $f(x) > B$ (resp $f(x) < B$) where B is a given value, high (resp low) enough to produce a good solution. The CSP framework also offers a sound basis for representing and solving decision problems without uncertainty [Schiex et al, 95]. All of these problems are at least NP-complete (see [Reeves, 93]) and huge instances require a large amount of computational resources. This is the reason why all these problems are very important in the field of Artificial Intelligence, with much research centered on the discovery of new algorithms for CSPs .

But, these algorithms, generally speaking, are black boxes. They do not enable you to solve problems needing Human Computer Interaction. The most typical example is time-table for schools. At least in France, there is no commercial software available to satisfy teachers: in most cases, time tables are still hand-made. It appears that, for some CSPs, it seems better to use knowledge based systems. But these problems need a mechanism of hypothesis generation with backtracking when an impasse occurs (to be described later in details). Our project stems from a knowledge base proposed by [Laurière, 86] using an inference engine called SNARK in which a function "REVENIR-SUR" directly implements the backtrack mecanism. We do not have such a feature within CLIPS (which is very costly since all the firings must be registered) but we used instead the logical conditional element.

Of course, one can object that an inference engine using variables realizes this mecanism before firing a rule. Such is the case of the RETE algorithm [Forgy, 82] which implements the instantiation of variables with backtrack when an impasse occurs. For example, consider the well known puzzle in which there are five houses, each of a different color, inhabited by men of different nationalities, with different pets, drinks, and cigarettes. Given the initial set of conditions, it must be determined which attributes are assigned to each man. It can be solved with only one rule in which each variable is represented by a conditional element (see the Zebra knowledge base proposed within CLIPS 6.0). However, this approach is very costly in terms of memory and solving N-queens with twenty or more queens is impossible on a PC with 32 Mo using the same way (see the rule in Appendix 1) So, the aim of this paper is to present the knowledge base we have designed using CLIPS to go beyond this limit.

In the next section, we present the classical Chronological Backtrack algorithm with a heuristic method to choose the next variable to instantiate [Laurière, 78] and with Forward Checking algorithm since [Haralick et al, 80] has proved that it is the most efficient algorithm, at least for binary CSP. Then, in Section 3, we present CCP (CCP stand for CLIPS Constraint Programming) our generic knowledge base and the PROPAG module which is specific to each problem. In Section 4, we conclude by introducing future work using CCP.

2. Basic algorithms for solving CSPs

2.1 Chronological Backtracking

The basic algorithm used to find a solution to CSPs is called Chronological Backtracking (CB). CB is a constructive algorithm [Minton et al, 92], since it builds up a solution variable by variable. The iterative process is decomposed into two steps. First CB chooses a variable (called the current-hypothesis) and instantiates it to the first possible value from its domain. Then CB checks the constraints whose all variables are instantiated. If one constraint fails, the algorithm backtracks. The previous instantiation of the current-hypothesis is undone and, if it is possible, replaced with another value (backtrack on the value). If all the values of the current-hypothesis have been tested, the algorithm backtracks to the previous instantiated variable (backtrack on the variable). If the set of possible values is empty for this variable, the algorithm backtracks again until an instantiated variable with a non empty set of possible values is found. This process continues until all the variables have been successfully instantiated, which means a solution has been found, or until all possible combinations of instantiations have been tried without success.

2.2 Improving backtracking with heuristics and with Forward Checking algorithm

The drawbacks of this approach are obvious and well-known and many improvements are possible. In our knowledge-based approach, we have introduced two classical improvements. First, we use a standard heuristic to choose the next variable to instantiate. The heuristic consists to choose the variable with the smallest remaining domain at each iteration of the algorithm. This is known as "first fail principle". The second improvement adds to the chronological backtrack a look-ahead scheme which attempts to detect impasses as early as possible in the search. The aim is to eliminate, in the domains of variables yet to be instantiated, values that are not consistent with the current instantiation. This is called the propagation phase, since the new instantiation gives new information on the current solution being constructed. At least, three algorithms exist doing this kind of job: Forward Checking, Partial Look-Ahead, Full Look-Ahead. We have retained Forward Checking, known to be the best compromise between the time spended on filtering the domains and the speed-up achieved by avoiding useless instantiations. Forward checking filters the domains of not instantiated variables which are directly linked to the current-hypothesis by a constraint. Of course, when a backtrack occurs, the inferences made during the propagation must be retracted.

2.3 Example: 4-queens problem.

Consider the classical 4-queens problem which consists to set 4 queens on a 4x4 in such a way that no one queen can capture another. It is obvious that you have to set only one queen per row (and per column). So, it is possible defining the problem as setting a queen on each row where the place is defined by the number of the column.

We obtain the following formulation:

Four variables Q1, Q2, Q3, Q4 where the number indicates the number of the row

The starting domain of each variable is D = {1, 2, 3, 4}

The constraints between Qi and Qj are [if v(Di) is the value of the variable Di]:

$i-j \neq v(Di) - v(Dj)$ (no two queens on the same positive diagonal)

$i-j \neq v(Dj) - v(Di)$ (no two queens on the same negative diagonal)

$v(Di) \neq v(Dj)$ (no two queens on the same column).

The search tree (see Figure 1) shows the process until the first solution is found. At the beginning (state 0), the board is empty. At state 1, Q1 is the first current-hypothesis with 1 as value. The Forward-Checking algorithm eliminates for Q2, Q3, Q4 the marked squares. At state 3, it is worth noticing two points: As there is no choice for Q3, since its domain only contains one value, Q3 is not marked as a hypothesis. After the propagation of Q3 value, the domain of Q4 becomes empty. This denotes an impasse and the search algorithm backtracks to the current hypothesis which is Q2. Since there exists another value for Q2, it is a backtrack on the value. At state 5, there is another impasse. But this time, there is no another value possible for Q2, so it is a backtrack on the previous instantiated variable, Q1. And so on, until the first solution is found. One can notice that on the hypothesis made at the state 6, there is no more hypothesis until the solution is found.

In this example, the number of generated states is only 8. A simple enumerative algorithm would have generated 77 states and Chronological Backtracking without Forward-Checking, 31 states. More generally, the number of nodes is dramatically reduced using Chronological Backtracking with the "first fail" heuristic and Forward Checking.

In Figure 1, a cross indicates a value retracted by the Forward Checking algorithm. A gray square indicates a value which has previously produced an impasse. Note that in the state 4, Q4 is instantiated but it could have been Q3. The agenda contains two activations of the rule `forced-instantiation` (see Section 3.2) one with Q3 and one with Q4. No strategy exists for deciding which one must fire first, so CLIPS chooses.

3. Presentation of CCP

CCP is composed of three modules: the MAIN module, the SEARCH module and the PROPAGATION module. We will see that the MAIN module and the SEARCH module are completely independent of the particular problem. So, we put them into CCP.clp, a file that the users should not change. They should just load it before the file containing the PROPAG module which contains templates, facts and rules specific to the users problem.

3.1 The MAIN module

Three deftemplates are defined into this module. The first one, called `pb`, describes the current state of the problem and has three slots. The slot `state` describes the state of the problem with the six following possible values : `start`, `in-progress`, `new-depth`, `choice-var`, `solved` and `blocked`. The slot `level` represents the number of recursive calls made by the chronological backtrack during the search. At each new hypothesis, a fact issued from `pb` template is duplicated with the slot `level` incremented by one. This slot `level` allows us to backtrack on the variable when it is necessary. The last slot is `current-hypothesis` whose value is the variable chosen as an hypothesis at the current level of the search.

The second template, called `var`, describes every variable of the problem. The users must generate with this template all the variables describing their problem. Since these facts are the only ones which depend on the particular problem, we have decided that it is better for the users to define their variables in the PROPAG module, although the template is defined in the MAIN module. The advantage we see is the users have nothing to introduce in the MAIN module nor in the SEARCH module. The template contains four slots which are `name` for the name of the variable, `level` for the level in the stack (so, it has the same value that `level` in pb), `possible-values` for the list of possible values remaining in the domain at each level and `value` giving the value for an instantiated variable. For example, you can see in Figure 2 the facts corresponding to the templates `pb` and `var` at level 0 and 1 corresponding to the state 0 and state 1 in Figure 1.

The third template `solution` is useful to count the number of solutions and to register each solution into a mulislot `var-val` which is a list containing each variable and its value (we tried initially to construct a list of lists into the multislot `var-val` but it does not seem possible with CLIPS).

The MAIN module contains five rules which are all independent of the particular problem. In the first, the users are asked if they want all the solutions or if they prefer be asked after each found solution. We use the `build` function to add the rule corresponding to the users choice. Of course, using the `build` function is not necessary but it spares one rule and it is a very simple but not trivial illustration of the `build` function.

The second one is a metarule which constructs the focus stack when the problem is in progress. This value means that all the propagations are made and that the problem is not solved, nor blocked. So, it is necessary to make a new hypothesis. The rule changes the value from `in-progress` to `new-depth`.

The third one registers a new solution when it occurs.

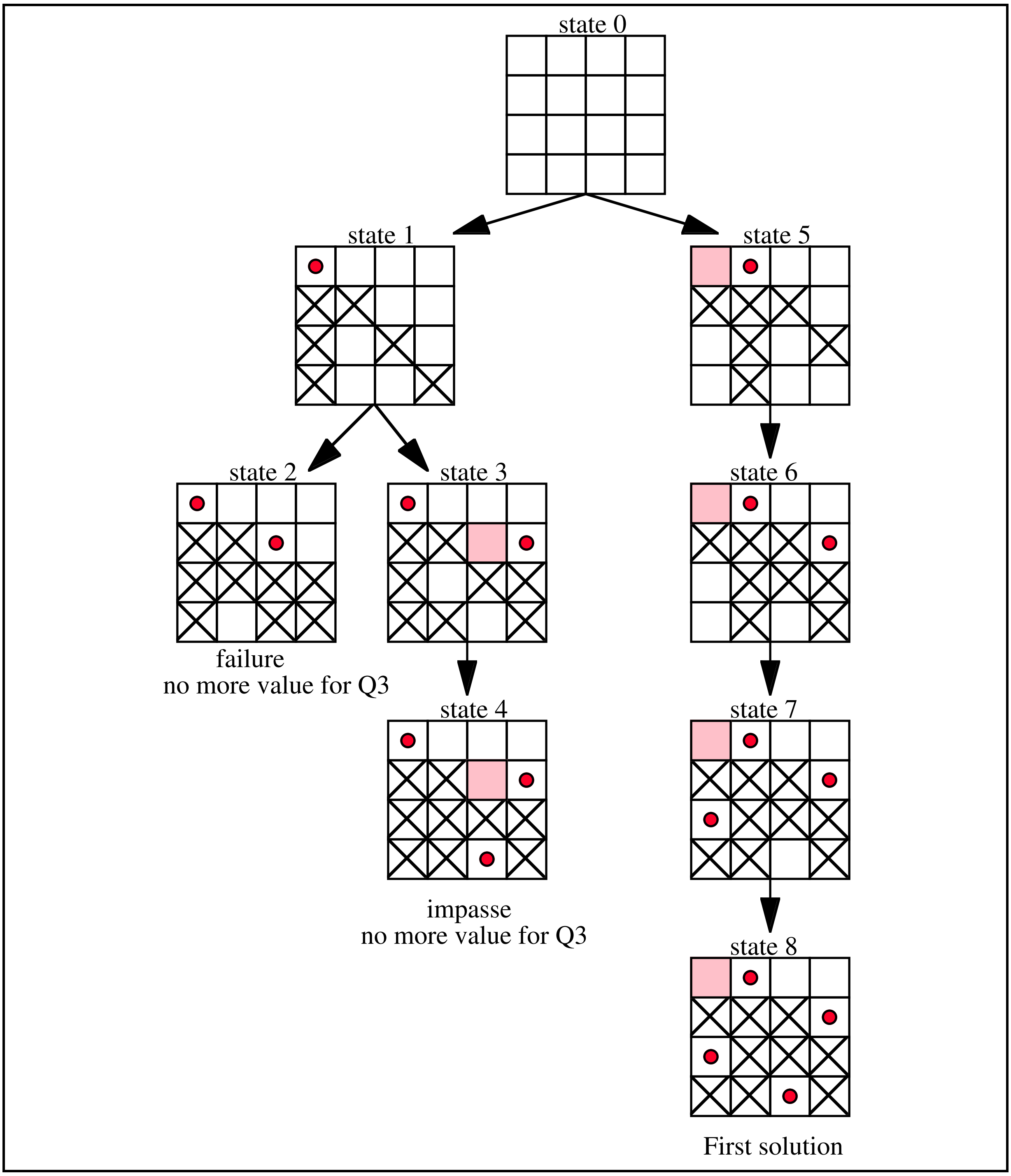

Figure 1: The tree developped until the first solution is found

(using Chronological Backtracking with Forward-Checking)

The fourth rule stops the search (no another solution or even no solution at all). This is the case when the value of the slot `state` is `blocked` and the value of the slot `level` is zero into the fact `pb` because no more backtrack is possible (there is no another value in the domain of the first variable chosen as current-hypothesis). The final rule of this module prints the solution (default printing method). If the users desire a customized printing we propose them to define it in the PROPAG module.

```
(level-search 0)
 (pb (state start) (level 0) (current-hypothesis no))
 (var (name x4) (level 0) (possible-values 1 2 3 4) (value nil))
 (var (name x3) (level 0) (possible-values 1 2 3 4) (value nil))
 (var (name x2) (level 0) (possible-values 1 2 3 4) (value nil))
 (var (name x1) (level 0) (possible-values 1 2 3 4) (value nil))

 (level-search 1)
 (pb (state in-progress) (level 1) (current-hypothesis x1))
 (var (name x1) (level 1) (possible-values 2 3 4) (value 1))
 (var (name x4) (level 1) (possible-values 2 3) (value nil))
 (var (name x3) (level 1) (possible-values 2 4) (value nil))
 (var (name x2) (level 1) (possible-values 3 4) (value nil))
```

Figure 2 : Facts corresponding to the templates `pb` and `var`

3.2 The SEARCH module

This module contains only rules but it imports templates defined in MAIN. These six rules implement the Chronological Backtrack with the "first fail" heuristic. When the slot `state` of the fact `pb` at the current level is `new-depth`, a new hypothesis must be made.

The first rule creates a new level of search. This means two new facts: if the current level is `?n`, a new ordered fact `(level-search (+ ?n 1))` is asserted and a fact `pb` is duplicated with the value `(+ ?n 1)` for the slot `level` and the value `choice-var` for the slot `state`.

This value activates the rule named `choice-var-val` which chooses as current-hypothesis the most constrained variable (i.e. one among those whose domain is the most reduced) and chooses as value the first available in the domain. We will improve this choice in the future, mainly by interacting with the users who can indicate a good choice, using human properties as experience, social knowledge or physical perception. After the instantiation of the current-hypothesis, the propagation of this new information must be made immediately. So the rule ends by focusing on the PROPAG module, which will be described after the SEARCH module.

The three other rules of the SEARCH module analyse the result of the PROPAG module according to the number of values remaining in the domain of not yet instantiated variables.

If the domain of a not yet instantiated variable contains only one element, there is no choice and the rule named `forced-instantiation` fires. Notice that a forced instantiation does not generate a new level in the stack but it is necessary to return immediately to the PROPAG module. Even this rule must not be fired twice before returning into the PROPAG module, since it can produce an erroneous instantiation. For example, one can see in the state 2 of the Figure 1 that the rule `forced-instantiation` can fire twice, once with Q3, once with Q4. Of course, this does not produce a solution.

When the domain of at least one variable is empty, backtracking is necessary, since the partial instantiation cannot produce a solution. Two cases are possible. If there is another possible value into the domain of the current-hypothesis, the current instantiation is undone and the following value is chosen by the rule named `backtrack-on-value`. To be consistent with this change of value, it is necessary to retract all the inferences made by the PROPAG module. This is made by means of the CLIPS's logical conditional element feature which provides a truth maintenance capability. We obtain the deletion of the facts `var` at the current level by retracting the ordered fact `(level-search ?n)` (and reasserting it in the same rule). In addition, the value of the slot `state` is changed from `blocked` to `in-progress` and, as there is a new value for the current hypothesis, it is necessary to return immediately in the PROPAG module.

The last case arrives when there is no further possible value into the domain of the current-hypothesis. In such a case, the rule named `backtrack-on-variable` fires. All the facts of the current level must be deleted and the current level becomes the previous one. One can notice that the value of the slot `state` at the previous level must be blocked because the value of the variable instantiated at this level must be changed. Moreover, it is possible that several backtracks on variable must be made consecutively. And, in all cases, after the rule `backtrack-on-variable` fires, the rule `backtrack-on-value` must fire. For example, it is the case in Figure 1 after the state 5. The current hypothesis is Q2, (Q4 has a forced instantiation), the problem is blocked and there is no more value for Q2. So the rule `backtrack-on-variable` fires and just after the rule `backtrack-on-value` fires with Q1 (see Figure 3).

```
f-18    (level-search 1)
f-20    (variable (name x1) (level 1) (possible-values 2 3 4) (value 1))
f-27    (pb (state new-depth) (level 1) (current-hypothesis x1))
f-36    (level-search 2)
f-37    (variable (name x2) (level 2) (possible-values) (value 4))
f-41    (variable (name x4) (level 2) (possible-values) (value 3))
; x4 has a forced instantiation
f-42    (variable (name x3) (level 2) (possible-values) (value nil))
f-43    (pb (state blocked) (level 2) (current-hypothesis x2))
; the rule "backtrack-on-variable" fires
f-18    (level-search 1)
f-20    (variable (name x1) (level 1) (possible-values 2 3 4) (value 1))
f-24    (variable (name x4) (level 1) (possible-values 2 3) (value nil))
f-25    (variable (name x3) (level 1) (possible-values 2 4) (value nil))
f-26    (variable (name x2) (level 1) (possible-values 3 4) (value nil))
f-44    (pb (state blocked) (level 1) (current-hypothesis x1))
; the rule "backtrack-on-value" fires
f-45    (level-search 1)
f-46    (variable (name x1) (level 1) (possible-values 3 4) (value 2))
f-47    (pb (state in-progress) (level 1) (current-hypothesis x1))
```

Figure 3 : Some facts showing the backtracking mecanism

3.3 The PROPAG module

As we remarked earlier this module contains all the specific templates, facts and rules necessary for a particular problem. First the users must create within the template `var` the facts list containing all the variables describing the problem. Two slots must be fill in: the slot `name` and the multislot `possible-values`. After this necessary declaration, the users have to define all the constraints they want to deal with. We have noted that in all the problems we have treated, we have to define some new templates in order to describe how the variables interact. For example, in the N-queens problem we define a template `queen` with the slot `name` (in order to make the link with the facts issued from `var`) and the slot `row` because the number which suffixes the value of the slot `name` indicates the number of the row (see Figure 4)

```
(deffacts chessboard
  (queen (name x1) (row 1))
  (queen (name x2) (row 2))
  (queen (name x3) (row 3))
  (queen (name x4) (row 4)))
```

Figure 4 : Facts issued from the template queen

```
(defrule PROPAG::propagation-2
; if the queen ?x is placed in the ?n1 row and the ?v column then remove
; from the list of possible values of the queen ?y all the values
; corresponding to the same diagonal
     (logical (level-search  ?n))
     (not (level-search   ?n1&:(> ?n1 ?n)))
     (variable (name ?x) (value ?v&~nil) (level ?n))
     (queen (name ?x) (row ?n1))
     (queen (name ?y) (row ?n2))
     ?f <- (variable (name ?y) (value nil) (level ?m)
     (possible-values $?liste&:(member$ (- ?v (- ?n1 ?n2)) ?liste)))
     (not (variable (name ?y) (level ?m1&:(> ?m1 ?m))))
=>
     (bind ?var (member$ (- ?v (- ?n1 ?n2)) ?liste))
     (if (= ?m ?n) then (modify ?f  (possible-values (delete$ ?liste ?var
?var)))
          else (duplicate ?f (level ?n) (possible-values (delete$ ?liste
?var ?var)))))
```

Figure 5 : A rule of the PROPAG module for the N-queens

Of course, the propagation rules are very particular. But only the LHS are specific, because the conditional elements describes the constraints. The RHS of these rules are all the same, since there is always one single action which deletes a value into the domain of a not yet instantiated variable. In order to limit the size of the fact-list, one can note that, when a new level is created in the stack of hypothesis, we do not duplicate the variables. We duplicate them only when a change occurs, i.e. the variable is the current-hypothesis or the variable is concerned by the propagation. This appears in our rules in the PROPAG module since in the RHS we need an "if then else" function. If it is the first reduction of its domain at the current level, then we need to duplicate the variable, otherwise we have just to modify it. (see Figure 5).

4 Concluding remarks

Of course, the performance of this knowledge base cannot be compared with the specialized CSPs algorithms in terms of computing time. However, we had two ideas in mind while developping this knowledge base.

First, from a pedagogical point of view, one can use CCP to explain the mecanism of Chronological Backtracking with Forward Checking and the "first fail" heuristic. Another pedagogical point of view is the use of many advanced features of CLIPS: the `build` function, a control strategy based on the management of the focus stack, the `forall`, `exists` and overall the `logical` conditional element.

Secondly, and this is the most important point, we would embed CCP into a larger knowledge base, since we have noted that, in some cases, solving a problem needs exploring hypotheses locally. Moreover, even with pure CSPs, the use of a knowledge based system must be very efficient, particulary in order to introduce interaction between the human and the computer during the solving process.

References

[Forgy, 82] C. L. Forgy: "Rete: a Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem", AI 19-1, 17-37, 1982

[Haralick et al, 80] R. M. Haralick, G. L. Elliott : "Increasing Tree Search Efficiency for Constraint Satisfaction Problems", Artificial Intelligence (14) 263-313, 1980

[Laurière, 78] J.-L. Laurière: "A Language and a Program for Stating and Solving Combinatorial Problems", Artificial Intelligence (10) 29-117, 1978

[Laurière, 86] J.-L. Laurière: "Un langage déclaratif: SNARK", Revue Technique et Science informatiques, vol 5, n°3, 141-171, 1986

[Minton et al, 92] S. Minton, M. Johnston, A. Philips and P. Laird: "Minimizing Conflicts: a Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems", Artificial Intelligence (58) 161-205, 1992

[Reeves, 93] C. Reeves: *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell, 1993

[Schiex et al, 95] T. Shiex, H. Fargier, G. Verfaillie: "Valued Constraint Satisfaction Problems: Hard and Easy Problems", Proceedings of IJCAI 95, 631-637, 1995

Annexe 1 : Solving 5-queens with only one rule

```
(deftemplate queen (slot row)(slot column))
(defrule init
=>
   (loop-for-count (?i 1 5) do
      (loop-for-count (?j 1 5) do
         (assert (queen (row ?i)(column ?j))))))
(defrule 5-queens
   (queen (row 1)(column ?y1))
   (queen (row 2)(column ?y2&~?y1&:(and
              (<> (+ 1 ?y1)(+ 2 ?y2)) (<> (- 1 ?y1)(- 2 ?y2)))))
   (queen (row 3)(column ?y3&~?y1&~?y2&:(and
              (<> (+ 1 ?y1)(+ 3 ?y3)) (<> (- 1 ?y1)(- 3 ?y3))
              (<> (+ 2 ?y2)(+ 3 ?y3)) (<> (- 2 ?y2)(- 3 ?y3)))))
   (queen (row 4)(column ?y4&~?y1&~?y2&~?y3&:(and
              (<> (+ 1 ?y1)(+ 4 ?y4)) (<> (- 1 ?y1)(- 4 ?y4))
              (<> (+ 2 ?y2)(+ 4 ?y4)) (<> (- 2 ?y2)(- 4 ?y4))
              (<> (+ 3 ?y3)(+ 4 ?y4)) (<> (- 3 ?y3)(- 4 ?y4)))))
   (queen (row 5)(column ?y5&~?y1&~?y2&~?y3&~?y4&:(and
              (<> (+ 1 ?y1)(+ 5 ?y5)) (<> (- 1 ?y1)(- 5 ?y5))
              (<> (+ 2 ?y2)(+ 5 ?y5)) (<> (- 2 ?y2)(- 5 ?y5))
              (<> (+ 3 ?y3)(+ 5 ?y5)) (<> (- 3 ?y3)(- 5 ?y5))
              (<> (+ 4 ?y4)(+ 5 ?y5)) (<> (- 4 ?y4)(- 5 ?y5)))))
=>
  (loop-for-count (?i 1 (- ?y1 1)) do (printout t "* "))(printout t "X
")(loop-for-count (?i (+ ?y1 1) 5) do (printout t "* "))(printout t crlf)
   (loop-for-count (?i 1 (- ?y2 1)) do (printout t "* "))(printout t "X
")(loop-for-count (?i (+ ?y2 1) 5) do (printout t "* "))(printout t crlf)
   (loop-for-count (?i 1 (- ?y3 1)) do (printout t "* "))(printout t "X
")(loop-for-count (?i (+ ?y3 1) 5) do (printout t "* "))(printout t crlf)
   (loop-for-count (?i 1 (- ?y4 1)) do (printout t "* "))(printout t "X
")(loop-for-count (?i (+ ?y4 1) 5) do (printout t "* "))(printout t crlf)
   (loop-for-count (?i 1 (- ?y5 1)) do (printout t "* "))(printout t "X
")(loop-for-count (?i (+ ?y5 1) 5) do (printout t "* "))(printout t crlf)
(printout t " Type RETURN for the next solution : ")
(readline)
(printout t crlf crlf)
)
```

Annexe 2 : CCP code

```
;***********************************************************************
; File:       CCP.clp  (C.C.P. stand for Clips Constraints Programming)
; Content:    Generic Clips Module for Constraints Programming
; Authors:    Jean-Marc Labat and Michel Futtersack
; Version:    1.0  (September 1997)
; Contact:    Jean-Marc.Labat,Michel.Futtersack@poleia.lip6.fr
;***********************************************************************
;########################  MODULE  MAIN  ###########################

(defmodule MAIN
   (export deftemplate ?ALL))

(deftemplate MAIN::pb
   (slot  state (default start))
   (slot  level  (default 0))
   (slot  current-hypothesis (default no))
)

(deftemplate MAIN::var
   (slot  name)
   (slot  level (default 0))
   (multislot  possible-values)
   (slot  value)
)

(deftemplate MAIN::solution
   (slot number (default 1))
   (multislot var-val)
)

(deffacts MAIN::initial-pb
   (pb)
   (solution)
   (level_search 0)
)

(deffunction MAIN::yes-or-no-p (?question)
  (bind ?x bogus)
  (while (and (neq ?x yes) (neq ?x y) (neq ?x no) (neq ?x n))
     (format t "%s(Yes or No) " ?question)
     (bind ?x (lowcase (sym-cat (read)))))
  (if (or (eq ?x yes) (eq ?x y)) then TRUE else FALSE))

 (defrule MAIN::all-solutions?
  ?pb<- (pb (state start))
=>
  (if (yes-or-no-p "Do you want all the solutions ? ")
     then  (build
            "(defrule  MAIN::pb-solved
                  (pb  (state solved))
                  ?sol <- (solution (number ?nb))
                =>
                  (printout t \"the problem is solved\" crlf)
                  (duplicate ?sol (number (+ ?nb 1))(var-val))
                  (focus SEARCH))
            ")
      else  (build
            "(defrule  MAIN::pb-solved
                  ?f <- (pb (state solved))
                  ?sol <- (solution (number ?nb))
```

```
                  =>
                    (printout t \"the problem is solved\" crlf)
                    (duplicate ?sol (number (+ ?nb 1))(var-val))
                    (if (yes-or-no-p \"Do you want another solution ? \")
                      then (focus SEARCH)
                      else (modify ?f (level 0) (state blocked)) ))
               ")
   )
   (modify ?pb (state in-progress))
)

(defrule MAIN::analysis
   ?f <- (pb (state in-progress))
=>
   (modify ?f (state new-depth))
   (focus SEARCH PROPAG SEARCH MAIN)
)

(defrule MAIN::save-solution
; if all the variables have got a value at the deapest level of the search
; then save the solution in a list of bindings (<variable> <value>)
          (declare (salience 10))
          (level_search ?n)
          (not (level_search  ?n1&:(> ?n1 ?n)))
          (forall (var (name ?x))
                  (var (name ?x)(value ?v&~nil)))

          (var (name ?x) (value ?val&~nil))
    ?s <- (solution (var-val $?list&:(not (member$ ?x $?list))))
    ?f <- (pb (level ?n))
=>
    (modify ?s (var-val (insert$ $?list 1  (create$ ?x ?val))))
    (modify ?f (state solved))
)

(defrule MAIN::no-more-solution
   (declare (salience 10))
   (level_search  0)
   ?f <- (solution (number ?nb) (var-val))
   (pb (state blocked) (level 0))
=>
   (if (= ?nb 1) then (printout t "I found no solution" crlf)
                 else (printout t "No more solution" crlf crlf)
                      (retract ?f)
                      (printout t "I found " (- ?nb 1)  " solution(s)" crlf
crlf)
   )
)

(defrule MAIN::print-solution
; the default printing method
   (pb (state blocked) (level 0))
   (solution (number ?n) (var-val $?list&:(> (length$ $?list) 0)))
=>
   (printout t crlf "Solution number " ?n " : "  crlf)
   (bind ?length (length$ $?list))
   (bind ?i 1)
   (while (< ?i ?length) (printout t (nth$ ?i $?list) " : " (nth$ (+ ?i 1)
$?list) crlf) (bind ?i (+ ?i 2)))
)

; add the following rule if you have specific printing rules in the PROPAG
; module
```

```
;(defrule MAIN::to_customized_printing
;; branch to rules for specific printing in the PROPAG module
;   (declare (salience -10))
;   (pb (state blocked)(level 0))
;=>
;   (printout t crlf crlf)
;   (focus PROPAG)
;)


;#################  MODULE  SEARCH   ####################################


(defmodule SEARCH
   (import MAIN deftemplate ?ALL))


(defrule  SEARCH::generate-new-depth
         (level_search ?n)
         (not (level_search  ?n1&:(> ?n1 ?n)))
   ?f <- (pb (level ?n) (state new-depth))
=>
   (duplicate ?f (level  (+  ?n  1))  (state  choice-var)(current-hypothesis
nil))
   (assert (level_search (+ ?n 1)))
)


(defrule SEARCH::choice-var-val
; make an hypothesis at a new level of the search. Choose the most
constrained
; variable (i.e. the ones which has the less possible values) and choose a
value among
; its possible values (the first available in the list...we should improve
this choice !!)
         (logical (level_search ?n))
   ?g <- (pb (level ?n)(state choice-var)(current-hypothesis nil))
   ?f <- (var (name ?x) (level ?x1) (value nil) (possible-values $?nb1))
         (not (var (name ?x) (level ?x2&:(> ?x2 ?x1))) )
         (forall (and (var (name ?y&~?x) (value nil)(level ?m))
                      (not (var (name ?y) (level ?m1&:(> ?m1 ?m)))))
                 (var   (name   ?y)(level   ?m)(possible-values   $?nb2&:(>=
(length$ ?nb2) (length$ $?nb1)))))

=>
   (modify ?g (state in-progress) (current-hypothesis ?x))
   (duplicate ?f (level  ?n)  (value  (nth$  1  $?nb1))(possible-values  (rest$
$?nb1)))
   (focus PROPAG)
)



;.Examine the current state : forced instanciation or impasse or backtrack

(defrule SEARCH::instanciation
; if a not yet instanciated variable has only one possible value then
provide it with this value
; then set the focus to PROPAG for evaluating the consequences of this
forced instanciation
; then return to SEARCH
         (logical (level_search  ?n))
```

```
   ?f <- (var (name ?x) (level ?n) (value nil) (possible-values $?list&:(=
(length$ $?list) 1)))
         (pb (state in-progress) (level ?n))
=>
   (modify ?f (value (nth$ 1 $?list))(possible-values))
   (focus PROPAG SEARCH)
)

(defrule SEARCH::impasse
; if there exists a not yet instanciated variable which has no more
possible values then you got an impasse
          (declare (salience 10))
          (logical (level_search ?n))
   ?pb <- (pb (level ?n) (state in-progress))
          (exists (var (level ?n)(value nil) (possible-values)))
=>
   (modify ?pb (state blocked))
)

(defrule SEARCH::backtrack-on-value
; if the problem solving process is blocked with the current hypothesis ?x
; or if we are looking for another solution, we have to choose another
value for
; the hypothesis.
; We stay at the same level of search but BE CAREFUL !!! we have to delete
all the facts
; depending on the current value of the hypothesis ?x
; thanks to the logical dependencies it suffices to retract the fact ?s
; then to restore it (with a new time-tag)
   ?s   <- (level_search  ?n)
   ?pb  <- (pb (level ?n) (state blocked|solved) (current-hypothesis ?x))
   ?var <- (var (name ?x) (level ?n) (value ?v&~nil) (possible-values
$?list&:(<> (length$ $?list) 0)))
 =>
   (retract ?s)
   (assert (level_search ?n))
   (modify ?var  (value (nth$ 1 $?list))(possible-values (rest$ $?list)))
   (modify ?pb (state in-progress))
   (focus PROPAG)
)

(defrule SEARCH::backtrack-on-variable
; if the problem solving process is blocked and no other value is available
for the
; current hypothesis then we have to return to a previous level of the
search
; where the problem solving process was not blocked. The next step will be
the choice of another
; value for the current hypothesis of this previous level
   ?s     <- (level_search ?n)
   ?pb    <- (pb (level ?n) (state blocked|solved)(current-hypothesis ?x))
   ?pbbis <- (pb (level ?m&:(= ?m (- ?n 1))) (state ~blocked))
   ?var   <- (var (name ?x) (level ?n) (possible-values $?list&:(= (length$
$?list) 0)))
=>
   (retract ?s)
   (retract ?pb)
   (retract ?var)
   (modify ?pbbis  (state blocked))
)
```

Annexe 3 : Examples of problems solved by CCP

```
;*****************************************************************
; File:       N-QUEENS.clp
; Content:    Specific Module for Solving the N-queens Problem.
;             This module must be used with the MAIN and SEARCH
;             modules contained in the CCP.clp file
; Authors:    Jean-Marc Labat and Michel Futtersack
; Version:    1.0  (September 1997)
; Contact:    Jean-Marc.Labat,Michel.Futtersack@poleia.lip6.fr
;*****************************************************************


;#################### the N-Queens Problem #########################
; Place N queens on a NxN chessboard in such a way that no one queen
; could take another queen. We have chosen N = 12. Greater N is feasible
; but the time of computation grows exponentially !
;##################################################################



;####################  Modeling  ##########################################
; A variable represents a queen in a row. For example, x1 represents
; the queen placed on the first row. We have to define a bijection of
; the set of rows {x1, x2,...x12} onto the set of columns {1, 2, ...12}
; In this modeling some constraints are implicit : only one variable
; is called x1 implies that only one queen will be placed on the first row
;#########################################################################

(defmodule PROPAG
   (import MAIN deftemplate ?ALL))

;################## possible values for the variables ################

(deffacts PROPAG::queens
   (var (name x1)  (possible-values (create$ 1 2 3 4 5 6 7 8 9 10 11 12)))
   (var (name x2)  (possible-values (create$ 1 2 3 4 5 6 7 8 9 10 11 12)))
   (var (name x3)  (possible-values (create$ 1 2 3 4 5 6 7 8 9 10 11 12)))
   (var (name x4)  (possible-values (create$ 1 2 3 4 5 6 7 8 9 10 11 12)))
   (var (name x5)  (possible-values (create$ 1 2 3 4 5 6 7 8 9 10 11 12)))
   (var (name x6)  (possible-values (create$ 1 2 3 4 5 6 7 8 9 10 11 12)))
   (var (name x7)  (possible-values (create$ 1 2 3 4 5 6 7 8 9 10 11 12)))
   (var (name x8)  (possible-values (create$ 1 2 3 4 5 6 7 8 9 10 11 12)))
   (var (name x9)  (possible-values (create$ 1 2 3 4 5 6 7 8 9 10 11 12)))
   (var (name x10) (possible-values (create$ 1 2 3 4 5 6 7 8 9 10 11 12)))
   (var (name x11) (possible-values (create$ 1 2 3 4 5 6 7 8 9 10 11 12)))
   (var (name x12) (possible-values (create$ 1 2 3 4 5 6 7 8 9 10 11 12)))
)

;##################  description of the chessboard  #################

(deftemplate PROPAG::queen
   (slot name)
   (slot row)
)


(deffacts chessboard
   (queen (name x1) (row 1))
   (queen (name x2) (row 2))
   (queen (name x3) (row 3))
   (queen (name x4) (row 4))
   (queen (name x5) (row 5))
```

```
   (queen (name x6) (row 6))
   (queen (name x7) (row 7))
   (queen (name x8) (row 8))
   (queen (name x9) (row 9))
   (queen (name x10) (row 10))
   (queen (name x11) (row 11))
   (queen (name x12) (row 12))
   (nb_variables 12)
   (rank_of_printing 1)
   (solution_to_be_printed 1)
)

;##########################  Functions  ###############################

(deffunction PROPAG::cassoc$ (?key ?Alist)
; useful for printing the solution
; cassoc$ returns the value associated with the key ?key in the association
list ?Alist
   (nth$ (+ 1 (member$ ?key ?Alist)) ?Alist)
)

;########################## Constraints propagation  ##################

(defrule PROPAG::propagation-1
; if the column ?v has been chosen for the queen ?x then remove ?v
;  from the list of possible values of the other queens not yet placed
   (declare (salience 2))
   (logical (level_search  ?n))
   (not (level_search  ?n1&:(> ?n1 ?n)))
   (var (name ?x) (value ?v&~nil)(level ?n))
   ?f <- (var (name ?y) (value nil) (level ?m)
                   (possible-values $?liste&:(member$ ?v ?liste)))
   (not (var (name ?y) (level ?m1&:(> ?m1 ?m))))
=>
   (bind ?var (member$ ?v ?liste))
   (if (= ?m ?n)
   then (modify ?f  (possible-values (delete$ ?liste ?var ?var )))
   else (duplicate ?f (level ?n) (possible-values (delete$ ?liste ?var ?var
))))
)


(defrule PROPAG::propagation-2
; if the queen ?x is placed in the ?n1 row and the ?v column then
; remove from the list of possible values of the queen ?y all the values
; correponding to the same diagonal
   (logical (level_search  ?n))
   (not (level_search   ?n1&:(> ?n1 ?n)))
   (var (name ?x) (value ?v&~nil) (level ?n))
   (queen (name ?x) (row ?n1))
   (queen (name ?y) (row ?n2))
   ?f <- (var (name ?y) (value nil) (level ?m)
                   (possible-values  $?liste&:(member$  (-  ?v  (-  ?n1  ?n2))
?liste)))
   (not (var (name ?y) (level ?m1&:(> ?m1 ?m))))
=>
   (bind ?var (member$ (- ?v (- ?n1 ?n2)) ?liste))
   (if (= ?m ?n)
   then (modify ?f  (possible-values (delete$ ?liste ?var ?var)))
   else  (duplicate  ?f  (level  ?n)  (possible-values  (delete$  ?liste  ?var
?var))))
)
```

```
 (defrule PROPAG::propagation-3
; like propagation-2, for the other diagonal
   (logical (level_search  ?n))
   (not (level_search  ?n1&:(> ?n1 ?n)))
   (var (name ?x) (value ?v&~nil) (level ?n))
   (queen (name ?x) (row ?n1))
   (queen (name ?y) (row ?n2))
   ?f <- (var (name ?y) (value nil) (level ?m)
                  (possible-values  $?liste&:(member$  (-  ?v  (-  ?n2  ?n1))
?liste)))
   (not (var (name ?y) (level ?m1&:(> ?m1 ?m))))
=>
   (bind ?var (member$ (- ?v (- ?n2 ?n1)) ?liste))
    (if (= ?m ?n)
    then (modify ?f  (possible-values (delete$ ?liste ?var ?var)))
    else  (duplicate  ?f  (level  ?n)  (possible-values  (delete$  ?liste  ?var
?var))))
)

;############################  rules for printing the chessboard  #######

(defrule PROPAG::customized_printing
; print the chessboard corresponding to a solution
      (pb (state blocked)(level 0))
      (solution (number ?n) (var-val $?list&:(> (length$ $?list) 0)))
      (solution_to_be_printed ?n)
      (queen (name ?queen) (row ?r))
      (nb_variables ?nb)
?p <- (rank_of_printing ?r)
=>
   (bind ?v (cassoc$ ?queen $?list))
   (loop-for-count  (?i  1  (-  ?v  1))  do  (printout  t  "*  "))(printout  t  "X
")(loop-for-count (?i (+ ?v 1) ?nb) do (printout t "* "))(printout t crlf)
   (assert (rank_of_printing (+ ?r 1)))
   (retract ?p)
)

(defrule PROPAG::next_printing
; for printing the next solution if any
   (declare (salience -10))
   (pb (state blocked)(level 0))
   (solution (number ?n))
?p<-(rank_of_printing ?r)
?q<-(solution_to_be_printed ?n)
=>
   (printout t crlf crlf)
   (retract ?p)
   (assert (rank_of_printing 1))
   (retract ?q)
   (assert (solution_to_be_printed (+ ?n 1)))
)
```

```
;**************************************************************************
;* File:      GRAPHCOL.clp                                                *
;* Content:   Specific Module for Solving the Graph Coloring Problem.*
;*            This module must be used with the MAIN and SEARCH     *
;*            modules contained in the CCP.clp file                 *
;* Authors:   Michel Futtersack and Jean-Marc Labat                 *
;* Version:   1.0  (September 1997)                                 *
;* Contact:   Michel.Futtersack,Jean-Marc.Labat@poleia.lip6.fr      *
;**************************************************************************


;###################  The Graph Coloring Problem  #####
;
; Given an undirected graph, try to attribute a color to each node
; in such a way that no neighbours have the same color
;
;#########################################################################



;###################  Modeling  ###########################################
;
; A variable represents a node. We have to define a map of the
; set of nodes {x1,     xn} onto a set of colors {Cyan, Magenta, Yellow}.
; It might happen that 3 colors no suffice, but we know (the theorem has
; been proved recently by a computer) that 4 colors suffice for coloring
; any graph
;##########################################################################


(defmodule PROPAG
   (import MAIN deftemplate ?ALL))


;##################  possible values for the variables  ################

(deffacts PROPAG::variables
   (var (name x1) (possible-values (create$ Cyan Magenta Yellow)));
   (var (name x2) (possible-values (create$ Cyan Magenta Yellow)));
   (var (name x3) (possible-values (create$ Cyan Magenta Yellow)));
   (var (name x4) (possible-values (create$ Cyan Magenta Yellow)));
   (var (name x5) (possible-values (create$ Cyan Magenta Yellow)));
   (var (name x6) (possible-values (create$ Cyan Magenta Yellow)));
   (var (name x7) (possible-values (create$ Cyan Magenta Yellow)));
   (var (name x8) (possible-values (create$ Cyan Magenta Yellow)));
   (var (name x9) (possible-values (create$ Cyan Magenta Yellow)));
   (var (name x10) (possible-values (create$ Cyan Magenta Yellow)));
   (var (name x11) (possible-values (create$ Cyan Magenta Yellow)));
   (var (name x12) (possible-values (create$ Cyan Magenta Yellow)));
   (var (name x13) (possible-values (create$ Cyan Magenta Yellow)));
   (var (name x14) (possible-values (create$ Cyan Magenta Yellow)));
   (var (name x15) (possible-values (create$ Cyan Magenta Yellow)));
   (var (name x16) (possible-values (create$ Cyan Magenta Yellow)));
   (var (name x17) (possible-values (create$ Cyan Magenta Yellow)));
   (var (name x18) (possible-values (create$ Cyan Magenta Yellow)));
   (var (name x19) (possible-values (create$ Cyan Magenta Yellow)));
   (var (name x20) (possible-values (create$ Cyan Magenta Yellow)));
)
```

```
;####################  Description of the graph  ######################

(deftemplate node
   (slot name)
   (multislot neighbours)
)

(deffacts PROPAG::graph
   (node (name x1)(neighbours (create$ x3 x4 x6 x13 x17)))
   (node (name x2)(neighbours (create$ x9 x14 x15 x17)))
   (node (name x3)(neighbours (create$ x1 x10 x12 x17)))
   (node (name x4)(neighbours (create$ x1 x18)))
   (node (name x5)(neighbours (create$ x6 x18)))
   (node (name x6)(neighbours (create$ x1 x5 x7)))
   (node (name x7)(neighbours (create$ x6 x13 x14 x19)))
   (node (name x8)(neighbours (create$ x10 x20 x17)))
   (node (name x9)(neighbours (create$ x2 x16 x20)))
   (node (name x10)(neighbours (create$ x3 x8 x11 x20)))
   (node (name x11)(neighbours (create$ x10 x12)))
   (node (name x12)(neighbours (create$ x3 x11 x18)))
   (node (name x13)(neighbours (create$ x1 x7 x17)))
   (node (name x14)(neighbours (create$ x2 x7)))
   (node (name x15)(neighbours (create$ x2 x16)))
   (node (name x16)(neighbours (create$ x2 x9 x15 x20)))
   (node (name x17)(neighbours (create$ x1 x2 x3 x8 x13)))
   (node (name x18)(neighbours (create$ x4 x5 x12)))
   (node (name x19)(neighbours (create$ x7)))
   (node (name x20)(neighbours (create$ x8 x9 x10 x16)))
)

;#####################  Constraints propagation  #####################

(defrule PROPAG::no-same-color-for-neighbours
; if the node ?x is colored with the color ?v then remove ?v from the
possible values
; of any node ?y connected to ?x
   (declare (salience 2))
   (logical (level_search ?n))
   (not (level_search  ?n1&:(> ?n1 ?n)))
   (var (name ?x) (value ?v&~nil)(level ?n))
   ?f <- (var (name ?y) (value nil) (level ?m)
                    (possible-values $?liste&:(member$ ?v ?liste)))
   (not (var (name ?y) (level ?m1&:(> ?m1 ?m))))
   (node (name ?x))
   (node (name ?y)(neighbours $?neigh&:(member$ ?x $?neigh)))
=>
   (bind ?x (member$ ?v ?liste))
   (if (= ?m ?n)
       then (modify ?f  (possible-values (delete$ ?liste ?x ?x)))
       else (duplicate ?f (level ?n) (possible-values (delete$ ?liste ?x
?x)))
   )
)
```

```
;***********************************************************************
;* File:          CROSSWORDS.clp                                       *
;* Content:       Specific Module for Solving a CROSSWORDS Problem.    *
;*                This module must be used with the MAIN and SEARCH    *
;*                modules contained in the CCP.clp file                *
;* Authors:       Jean-Marc Labat and Michel Futtersack                *
;* Version:       1.0  (September 1997)                                *
;* Contact:       Jean-Marc.Labat, Michel.Futtersack@poleia.lip6.fr    *
;***********************************************************************


;######################## A Crosswords Problem ########################
; Fill in the following grid :
;   ________________________________________________
;   |  |  |  |  |  |  |  |  |**|  |  |  |  |  |  |
;   ________________________________________________
;   |  |  |  |  |  |  |  |**|  |  |  |  |  |  |  |
;   ________________________________________________
;   |  |  |  |  |**|  |  |  |  |  |  |  |**|  |  |
;   ________________________________________________
;   |  |  |  |  |  |**|  |  |  |  |**|  |  |  |  |
;   ________________________________________________
;   |  |**|  |**|  |  |  |  |  |  |  |**|  |  |  |
;   ________________________________________________
;   |  |  |  |  |  |  |**|  |  |**|  |  |  |  |  |
;   ________________________________________________
;   |  |  |**|  |  |  |  |  |**|  |  |  |  |  |**|
;   ________________________________________________
;   |**|  |  |  |  |  |  |  |  |**|  |  |  |**|  |
;   ________________________________________________
;   |  |  |  |**|  |  |  |**|  |  |**|  |  |  |  |
;   ________________________________________________
;   |  |  |  |  |**|  |  |  |  |  |  |  |**|  |  |
;   ________________________________________________
;   |  |  |  |  |  |**|  |  |  |  |  |**|  |  |  |
;   ________________________________________________
;   |  |**|  |  |  |  |  |  |**|  |  |  |  |  |  |
;   ________________________________________________
;   |  |  |  |  |  |  |  |**|  |  |  |  |  |  |  |
;   ________________________________________________
;   |  |  |  |**|  |  |**|  |  |  |  |  |**|  |  |
;   ________________________________________________
;   |  |  |**|  |  |  |  |  |  |  |  |**|  |  |  |
;   ________________________________________________
;
; placing horizontally or vertically the following (french) words :
;
; 2 letters : "as" "au" "ca" "en" "le" "no" "on" "re" "si" "te" "ut"
; 3 letters : "are" "emu" "eta" "eus" "nes" "ost" "pas"  "pre" "ree" "rue"
;"soi" "sur" "ton" "tri" "ulm"
; 4 letters : "anet" "arts" "crin" "emet" "erre"  "lino" "niee" "nuit"
;"reel" "sees" "unie" "user"
; 5 letters : "delta" "egale" "emeus" "etire" "gener" "gnous" "maree"
;"ornas" "relue" "satin" "verre"
; 6 letters : "apotre" "enemas" "enigme" "eperon" "essore" "froler"
;"genois" "isatis" "limais" "murage" "pietre" "ruelle"
; 7 letters : "cendres" "ecrites" "ecrouer" "enferme" "enumere" "esperes"
;"granule" "mailles" "origine"  "perores" "pimente" "reussie" "uretere"
;"valeurs"
; 8 letters : "assenees" "braisent" "curetage" "emergera"
;######################################################################
```

```
;########################  Modeling  ####################################
; A variable represents a place (i.e. a set of contiguous blanks on the
;same row or the same column) in the grid.
; h1, h2,...h41 denote the horizontal places and v1, v2,...v39 denote the
;vertical places.
; The names of the horizontal variables have been attributed by reading the
;grid
; from the left to the right row by row  The names of the vertical
;variables have been
; attributed by reading the grid from the top to the bottom column by
;column
; We have to define a bijection of the set of places onto the set of words
;#########################################################################


 (defmodule PROPAG
   (import MAIN deftemplate ?ALL))


;##################  possible values for the variables  #################

(deffacts PROPAG::pb-initial
   (var (name h1)     (possible-values (create$ "assenees" "braisent"
"curetage" "emergera" )))
   (var (name h2)   (possible-values (create$ "apotre" "enemas" "enigme"
"eperon" "essore" "froler" "genois" "isatis" "limais" "murage" "pietre"
"ruelle")))
   (var (name h3)   (possible-values (create$ "cendres" "ecrites" "ecrouer"
"enferme" "enumere" "esperes" "granule"
                                "mailles" "origine"  "perores" "pimente"
"reussie" "uretere" "valeurs")))
   (var (name h4)   (possible-values (create$ "cendres" "ecrites" "ecrouer"
"enferme" "enumere" "esperes" "granule"
                                "mailles" "origine"  "perores" "pimente"
"reussie" "uretere" "valeurs")))
   (var (name h5)   (possible-values (create$ "anet" "arts" "crin" "emet"
"erre"  "lino" "niee" "nuit" "reel" "sees" "unie" "user")))
   (var (name h6)   (possible-values (create$ "cendres" "ecrites" "ecrouer"
"enferme" "enumere" "esperes" "granule"
                                "mailles" "origine"  "perores" "pimente"
"reussie" "uretere" "valeurs")))
   (var (name h7)   (possible-values (create$ "as" "au" "ca" "en" "le" "no"
"on" "re" "si" "te" "ut")))
   (var (name h8)    (possible-values (create$ "delta" "egale" "emeus"
"etire" "gener" "gnous" "maree" "ornas" "relue" "satin" "verre")))
   (var (name h9)   (possible-values (create$ "anet" "arts" "crin" "emet"
"erre"  "lino" "niee" "nuit" "reel" "sees" "unie" "user")))
   (var (name h10)  (possible-values (create$ "anet" "arts" "crin" "emet"
"erre"  "lino" "niee" "nuit" "reel" "sees" "unie" "user")))
   (var (name h11)  (possible-values (create$ "cendres" "ecrites" "ecrouer"
"enferme" "enumere" "esperes" "granule"
                                "mailles" "origine"  "perores" "pimente"
"reussie" "uretere" "valeurs")))
   (var (name h12)  (possible-values (create$ "are" "emu" "eta" "eus" "nes"
"ost" "pas"  "pre" "ree" "rue" "soi" "sur" "ton" "tri" "ulm")))
   (var (name h13)  (possible-values (create$ "apotre" "enemas" "enigme"
"eperon" "essore" "froler" "genois" "isatis" "limais" "murage" "pietre"
"ruelle")))
   (var (name h14)  (possible-values (create$ "as" "au" "ca" "en" "le" "no"
"on" "re" "si" "te" "ut")))
   (var (name h15)   (possible-values (create$ "delta" "egale" "emeus"
"etire" "gener" "gnous" "maree" "ornas" "relue" "satin" "verre")))
```

```
   (var (name h16)  (possible-values (create$ "as" "au" "ca" "en" "le" "no"
"on" "re" "si" "te" "ut")))
   (var  (name  h17)   (possible-values  (create$  "delta"  "egale"  "emeus"
"etire" "gener" "gnous" "maree" "ornas" "relue" "satin" "verre")))
   (var  (name  h18)   (possible-values  (create$  "delta"  "egale"  "emeus"
"etire" "gener" "gnous" "maree" "ornas" "relue" "satin" "verre" "tente")))
   (var  (name  h19)   (possible-values  (create$  "assenees"  "braisent"
"curetage" "emergera")))
   (var (name h20)  (possible-values (create$ "are" "emu" "eta" "eus" "nes"
"ost" "pas"  "pre" "ree" "rue" "soi" "sur" "ton" "tri" "ulm")))
   (var (name h21)  (possible-values (create$ "are" "emu" "eta" "eus" "nes"
"ost" "pas"  "pre" "ree" "rue" "soi" "sur" "ton" "tri" "ulm")))
   (var (name h22)  (possible-values (create$ "are" "emu" "eta" "eus" "nes"
"ost" "pas"  "pre" "ree" "rue" "soi" "sur" "ton" "tri" "ulm")))
   (var (name h23)  (possible-values (create$ "as" "au" "ca" "en" "le" "no"
"on" "re" "si" "te" "ut")))
   (var (name h24)  (possible-values (create$ "anet" "arts" "crin" "emet"
"erre"  "lino" "niee" "nuit" "reel" "sees" "unie" "user")))
   (var (name h25)  (possible-values (create$ "anet" "arts" "crin" "emet"
"erre"  "lino" "niee" "nuit" "reel" "sees" "unie" "user")))
   (var (name h26)  (possible-values (create$ "cendres" "ecrites" "ecrouer"
"enferme" "enumere" "esperes" "granule"
                                 "mailles"  "origine"   "perores"  "pimente"
"reussie" "uretere" "valeurs")))
   (var (name h27)  (possible-values (create$ "as" "au" "ca" "en" "le" "no"
"on" "re" "si" "te" "ut")))
   (var  (name  h28)   (possible-values  (create$  "delta"  "egale"  "emeus"
"etire" "gener" "gnous" "maree" "ornas" "relue" "satin" "verre")))
   (var  (name  h29)   (possible-values  (create$  "delta"  "egale"  "emeus"
"etire" "gener" "gnous" "maree" "ornas" "relue" "satin" "verre")))
   (var (name h30)  (possible-values (create$ "are" "emu" "eta" "eus" "nes"
"ost" "pas"  "pre" "ree" "rue" "soi" "sur" "ton" "tri" "ulm")))
   (var  (name  h31)  (possible-values  (create$  "apotre"  "enemas"  "enigme"
"eperon"  "essore"  "froler"  "genois"  "isatis"  "limais"  "murage"  "pietre"
"ruelle")))
   (var  (name  h32)  (possible-values  (create$  "apotre"  "enemas"  "enigme"
"eperon"  "essore"  "froler"  "genois"  "isatis"  "limais"  "murage"  "pietre"
"ruelle")))
   (var (name h33)  (possible-values (create$ "cendres" "ecrites" "ecrouer"
"enferme" "enumere" "esperes" "granule"
                                 "mailles"  "origine"   "perores"  "pimente"
"reussie" "uretere" "valeurs")))
   (var (name h34)  (possible-values (create$ "cendres" "ecrites" "ecrouer"
"enferme" "enumere" "esperes" "granule"
                                 "mailles"  "origine"   "perores"  "pimente"
"reussie" "uretere" "valeurs")))
   (var (name h35)  (possible-values (create$ "are" "emu" "eta" "eus" "nes"
"ost" "pas"  "pre" "ree" "rue" "soi" "sur" "ton" "tri" "ulm")))
   (var (name h36)  (possible-values (create$ "as" "au" "ca" "en" "le" "no"
"on" "re" "si" "te" "ut")))
   (var  (name  h37)   (possible-values  (create$  "delta"  "egale"  "emeus"
"etire" "gener" "gnous" "maree" "ornas" "relue" "satin" "verre")))
   (var (name h38)  (possible-values (create$ "as" "au" "ca" "en" "le" "no"
"on" "re" "si" "te" "ut")))
   (var (name h39)  (possible-values (create$ "as" "au" "ca" "en" "le" "no"
"on" "re" "si" "te" "ut")))
   (var  (name  h40)   (possible-values  (create$  "assenees"  "braisent"
"curetage" "emergera")))
   (var (name h41)  (possible-values (create$ "are" "emu" "eta" "eus" "nes"
"ost" "pas"  "pre" "ree" "rue" "soi" "sur" "ton" "tri" "ulm")))

   (var (name v1)   (possible-values (create$ "cendres" "ecrites" "ecrouer"
"enferme" "enumere" "esperes" "granule"
```

```
                                  "mailles"  "origine"   "perores"  "pimente"
"reussie" "uretere" "valeurs")))
   (var (name v2)   (possible-values (create$ "cendres" "ecrites" "ecrouer"
"enferme" "enumere" "esperes" "granule"
                                  "mailles"  "origine"   "perores"  "pimente"
"reussie" "uretere" "valeurs")))
   (var (name v3)   (possible-values (create$ "anet" "arts" "crin" "emet"
"erre"  "lino" "niee" "nuit" "reel" "sees" "unie" "user")))
   (var (name v4)    (possible-values (create$ "apotre" "enemas" "enigme"
"eperon" "essore" "froler" "genois" "isatis" "limais" "murage" "pietre"
"ruelle")))
   (var (name v5)   (possible-values (create$ "are" "emu" "eta" "eus" "nes"
"ost" "pas"  "pre" "ree" "rue" "soi" "sur" "ton" "tri" "ulm")))
   (var (name v6)    (possible-values (create$ "apotre" "enemas" "enigme"
"eperon" "essore" "froler" "genois" "isatis" "limais" "murage" "pietre"
"ruelle")))
   (var (name v7)   (possible-values (create$ "cendres" "ecrites" "ecrouer"
"enferme" "enumere" "esperes" "granule"
                                  "mailles"  "origine"   "perores"  "pimente"
"reussie" "uretere" "valeurs")))
   (var (name v8)   (possible-values (create$ "anet" "arts" "crin" "emet"
"erre"  "lino" "niee" "nuit" "reel" "sees" "unie" "user")))
   (var (name v9)   (possible-values (create$ "are" "emu" "eta" "eus" "nes"
"ost" "pas"  "pre" "ree" "rue" "soi" "sur" "ton" "tri" "ulm")))
   (var (name v10)  (possible-values (create$ "anet" "arts" "crin" "emet"
"erre"  "lino" "niee" "nuit" "reel" "sees" "unie" "user")))
   (var (name v11)  (possible-values (create$ "as" "au" "ca" "en" "le" "no"
"on" "re" "si" "te" "ut")))
   (var (name v12)  (possible-values (create$ "apotre" "enemas" "enigme"
"eperon" "essore" "froler" "genois" "isatis" "limais" "murage" "pietre"
"ruelle")))
   (var (name v13)   (possible-values (create$ "delta" "egale" "emeus"
"etire" "gener" "gnous" "maree" "ornas" "relue" "satin" "verre")))
   (var (name v14)  (possible-values (create$ "are" "emu" "eta" "eus" "nes"
"ost" "pas"  "pre" "ree" "rue" "soi" "sur" "ton" "tri" "ulm")))
   (var (name v15)  (possible-values (create$ "apotre" "enemas" "enigme"
"eperon" "essore" "froler" "genois" "isatis" "limais" "murage" "pietre"
"ruelle")))
   (var (name v16)  (possible-values (create$ "anet" "arts" "crin" "emet"
"erre"  "lino" "niee" "nuit" "reel" "sees" "unie" "user")))
   (var (name v17)   (possible-values (create$ "delta" "egale" "emeus"
"etire" "gener" "gnous" "maree" "ornas" "relue" "satin" "verre")))
   (var (name v18)  (possible-values (create$ "cendres" "ecrites" "ecrouer"
"enferme" "enumere" "esperes" "granule"
                                  "mailles"  "origine"   "perores"  "pimente"
"reussie" "uretere" "valeurs")))
   (var (name v19)  (possible-values (create$ "apotre" "enemas" "enigme"
"eperon" "essore" "froler" "genois" "isatis" "limais" "murage" "pietre"
"ruelle")))
   (var (name v20)  (possible-values (create$ "are" "emu" "eta" "eus" "nes"
"ost" "pas"  "pre" "ree" "rue" "soi" "sur" "ton" "tri" "ulm")))
   (var (name v21)  (possible-values (create$ "as" "au" "ca" "en" "le" "no"
"on" "re" "si" "te" "ut")))
   (var (name v22)   (possible-values (create$ "delta" "egale" "emeus"
"etire" "gener" "gnous" "maree" "ornas" "relue" "satin" "verre")))
   (var (name v23)  (possible-values (create$ "anet" "arts" "crin" "emet"
"erre"  "lino" "niee" "nuit" "reel" "sees" "unie" "user")))
   (var (name v24)  (possible-values (create$ "are" "emu" "eta" "eus" "nes"
"ost" "pas"  "pre" "ree" "rue" "soi" "sur" "ton" "tri" "ulm")))
   (var (name v25)   (possible-values (create$ "delta" "egale" "emeus"
"etire" "gener" "gnous" "maree" "ornas" "relue" "satin" "verre")))
   (var (name v26)  (possible-values (create$ "cendres" "ecrites" "ecrouer"
"enferme" "enumere" "esperes" "granule"
```

```
                                "mailles"  "origine"   "perores"  "pimente"
"reussie" "uretere" "valeurs")))
   (var (name v27)  (possible-values (create$ "are" "emu" "eta" "eus" "nes"
"ost" "pas"  "pre" "ree" "rue" "soi" "sur" "ton" "tri" "ulm")))
   (var (name v28)  (possible-values (create$ "anet" "arts" "crin" "emet"
"erre"  "lino" "niee" "nuit" "reel" "sees" "unie" "user")))
   (var (name v29)  (possible-values (create$ "apotre" "enemas" "enigme"
"eperon" "essore" "froler" "genois" "isatis" "limais" "murage" "pietre"
"ruelle")))
   (var (name v30)  (possible-values (create$ "anet" "arts" "crin" "emet"
"erre"  "lino" "niee" "nuit" "reel" "sees" "unie" "user")))
   (var (name v31)  (possible-values (create$ "delta" "egale" "emeus"
"etire" "gener" "gnous" "maree" "ornas" "relue" "satin" "verre")))
   (var (name v32)  (possible-values (create$ "are" "emu" "eta" "eus" "nes"
"ost" "pas"  "pre" "ree" "rue" "soi" "sur" "ton" "tri" "ulm")))
   (var (name v33)  (possible-values (create$ "as" "au" "ca" "en" "le" "no"
"on" "re" "si" "te" "ut")))
   (var (name v34)  (possible-values (create$ "apotre" "enemas" "enigme"
"eperon" "essore" "froler" "genois" "isatis" "limais" "murage" "pietre"
"ruelle")))
   (var (name v35)  (possible-values (create$ "are" "emu" "eta" "eus" "nes"
"ost" "pas"  "pre" "ree" "rue" "soi" "sur" "ton" "tri" "ulm")))
   (var (name v36)  (possible-values (create$ "cendres" "ecrites" "ecrouer"
"enferme" "enumere" "esperes" "granule"
                                "mailles"  "origine"   "perores"  "pimente"
"reussie" "uretere" "valeurs")))
   (var (name v37)  (possible-values (create$ "cendres" "ecrites" "ecrouer"
"enferme" "enumere" "esperes" "granule"
                                "mailles"  "origine"   "perores"  "pimente"
"reussie" "uretere" "valeurs")))
   (var (name v38)  (possible-values (create$ "apotre" "enemas" "enigme"
"eperon" "essore" "froler" "genois" "isatis" "limais" "murage" "pietre"
"ruelle")))
   (var  (name  v39)   (possible-values  (create$  "assenees"  "braisent"
"curetage" "emergera"))))

;##########################  description of the constraints  ##############

(deftemplate PROPAG::interaction
   (slot var1)
   (slot letter1)
   (slot var2)
   (slot letter2)
)

(deffacts PROPAG::cross-words
;   the  constraint  :  (interaction  (var1  <h>)  (letter1  <i>)  (var2  <v>)
;(letter2 <j>))
;  means that the ith letter of the variable <h> must be equal to the jth
;letter
; of the variable <v>


   (interaction (var1 h1) (letter1 1) (var2 v1) (letter2 1))
   (interaction (var1 h1) (letter1 2) (var2 v3) (letter2 1))
   (interaction (var1 h1) (letter1 3) (var2 v6) (letter2 1))
   (interaction (var1 h1) (letter1 4) (var2 v8) (letter2 1))
   (interaction (var1 h1) (letter1 5) (var2 v11) (letter2 1))
   (interaction (var1 h1) (letter1 6) (var2 v14) (letter2 1))
   (interaction (var1 h1) (letter1 7) (var2 v17) (letter2 1))

   (interaction (var1 h2) (letter1 1) (var2 v25) (letter2 1))
   (interaction (var1 h2) (letter1 2) (var2 v27) (letter2 1))
```

```
(interaction (var1 h2) (letter1 3) (var2 v30) (letter2 1))
(interaction (var1 h2) (letter1 4) (var2 v33) (letter2 1))
(interaction (var1 h2) (letter1 5) (var2 v36) (letter2 1))
(interaction (var1 h2) (letter1 6) (var2 v38) (letter2 1))

(interaction (var1 h3) (letter1 1) (var2 v1) (letter2 2))
(interaction (var1 h3) (letter1 2) (var2 v3) (letter2 2))
(interaction (var1 h3) (letter1 3) (var2 v6) (letter2 2))
(interaction (var1 h3) (letter1 4) (var2 v8) (letter2 2))
(interaction (var1 h3) (letter1 5) (var2 v11) (letter2 2))
(interaction (var1 h3) (letter1 6) (var2 v14) (letter2 2))
(interaction (var1 h3) (letter1 7) (var2 v17) (letter2 2))


(interaction (var1 h4) (letter1 1) (var2 v22) (letter2 1))
(interaction (var1 h4) (letter1 2) (var2 v25) (letter2 2))
(interaction (var1 h4) (letter1 3) (var2 v27) (letter2 2))
(interaction (var1 h4) (letter1 4) (var2 v30) (letter2 2))
(interaction (var1 h4) (letter1 5) (var2 v33) (letter2 2))
(interaction (var1 h4) (letter1 6) (var2 v36) (letter2 2))
(interaction (var1 h4) (letter1 7) (var2 v38) (letter2 2))

(interaction (var1 h5) (letter1 1) (var2 v1) (letter2 3))
(interaction (var1 h5) (letter1 2) (var2 v3) (letter2 3))
(interaction (var1 h5) (letter1 3) (var2 v6) (letter2 3))
(interaction (var1 h5) (letter1 4) (var2 v8) (letter2 3))

(interaction (var1 h6) (letter1 1) (var2 v14) (letter2 3))
(interaction (var1 h6) (letter1 2) (var2 v17) (letter2 3))
(interaction (var1 h6) (letter1 3) (var2 v19) (letter2 1))
(interaction (var1 h6) (letter1 4) (var2 v22) (letter2 2))
(interaction (var1 h6) (letter1 5) (var2 v25) (letter2 3))
(interaction (var1 h6) (letter1 6) (var2 v27) (letter2 3))
(interaction (var1 h6) (letter1 7) (var2 v30) (letter2 3))

(interaction (var1 h7) (letter1 1) (var2 v36) (letter2 3))
(interaction (var1 h7) (letter1 2) (var2 v38) (letter2 3))

(interaction (var1 h8) (letter1 1) (var2 v1) (letter2 4))
(interaction (var1 h8) (letter1 2) (var2 v3) (letter2 4))
(interaction (var1 h8) (letter1 3) (var2 v6) (letter2 4))
(interaction (var1 h8) (letter1 4) (var2 v8) (letter2 4))
(interaction (var1 h8) (letter1 5) (var2 v12) (letter2 1))

(interaction (var1 h9) (letter1 1) (var2 v17) (letter2 4))
(interaction (var1 h9) (letter1 2) (var2 v19) (letter2 2))
(interaction (var1 h9) (letter1 3) (var2 v22) (letter2 3))
(interaction (var1 h9) (letter1 4) (var2 v25) (letter2 4))

(interaction (var1 h10) (letter1 1) (var2 v30) (letter2 4))
(interaction (var1 h10) (letter1 2) (var2 v34) (letter2 1))
(interaction (var1 h10) (letter1 3) (var2 v36) (letter2 4))
(interaction (var1 h10) (letter1 4) (var2 v38) (letter2 4))

(interaction (var1 h11) (letter1 1) (var2 v12) (letter2 2))
(interaction (var1 h11) (letter1 2) (var2 v15) (letter2 1))
(interaction (var1 h11) (letter1 3) (var2 v17) (letter2 5))
(interaction (var1 h11) (letter1 4) (var2 v19) (letter2 3))
(interaction (var1 h11) (letter1 5) (var2 v22) (letter2 4))
(interaction (var1 h11) (letter1 6) (var2 v25) (letter2 5))
(interaction (var1 h11) (letter1 7) (var2 v28) (letter2 1))

(interaction (var1 h12) (letter1 1) (var2 v34) (letter2 2))
```

```
(interaction (var1 h12) (letter1 2) (var2 v36) (letter2 5))
(interaction (var1 h12) (letter1 3) (var2 v38) (letter2 5))

(interaction (var1 h13) (letter1 1) (var2 v1) (letter2 6))
(interaction (var1 h13) (letter1 2) (var2 v4) (letter2 1))
(interaction (var1 h13) (letter1 3) (var2 v6) (letter2 6))
(interaction (var1 h13) (letter1 4) (var2 v9) (letter2 1))
(interaction (var1 h13) (letter1 5) (var2 v12) (letter2 3))
(interaction (var1 h13) (letter1 6) (var2 v15) (letter2 2))

(interaction (var1 h14) (letter1 1) (var2 v19) (letter2 4))
(interaction (var1 h14) (letter1 2) (var2 v22) (letter2 5))

(interaction (var1 h15) (letter1 1) (var2 v28) (letter2 2))
(interaction (var1 h15) (letter1 2) (var2 v31) (letter2 1))
(interaction (var1 h15) (letter1 3) (var2 v34) (letter2 3))
(interaction (var1 h15) (letter1 4) (var2 v36) (letter2 6))
(interaction (var1 h15) (letter1 5) (var2 v38) (letter2 6))

(interaction (var1 h16) (letter1 1) (var2 v1) (letter2 7))
(interaction (var1 h16) (letter1 2) (var2 v4) (letter2 2))

(interaction (var1 h17) (letter1 1) (var2 v9) (letter2 2))
(interaction (var1 h17) (letter1 2) (var2 v12) (letter2 4))
(interaction (var1 h17) (letter1 3) (var2 v15) (letter2 3))
(interaction (var1 h17) (letter1 4) (var2 v18) (letter2 1))
(interaction (var1 h17) (letter1 5) (var2 v19) (letter2 5))

(interaction (var1 h19) (letter1 1) (var2 v4) (letter2 3))
(interaction (var1 h19) (letter1 2) (var2 v7) (letter2 1))
(interaction (var1 h19) (letter1 3) (var2 v9) (letter2 3))
(interaction (var1 h19) (letter1 4) (var2 v12) (letter2 5))
(interaction (var1 h19) (letter1 5) (var2 v15) (letter2 4))
(interaction (var1 h19) (letter1 6) (var2 v18) (letter2 2))
(interaction (var1 h19) (letter1 7) (var2 v19) (letter2 6))
(interaction (var1 h19) (letter1 8) (var2 v23) (letter2 1))

(interaction (var1 h20) (letter1 1) (var2 v28) (letter2 4))
(interaction (var1 h20) (letter1 2) (var2 v31) (letter2 3))
(interaction (var1 h20) (letter1 3) (var2 v34) (letter2 5))

(interaction (var1 h21) (letter1 1) (var2 v2) (letter2 1))
(interaction (var1 h21) (letter1 2) (var2 v4) (letter2 4))
(interaction (var1 h21) (letter1 3) (var2 v7) (letter2 2))

(interaction (var1 h22) (letter1 1) (var2 v12) (letter2 6))
(interaction (var1 h22) (letter1 2) (var2 v15) (letter2 5))
(interaction (var1 h22) (letter1 3) (var2 v18) (letter2 3))

(interaction (var1 h23) (letter1 1) (var2 v23) (letter2 2))
(interaction (var1 h23) (letter1 2) (var2 v26) (letter2 1))

(interaction (var1 h24) (letter1 1) (var2 v31) (letter2 4))
(interaction (var1 h24) (letter1 2) (var2 v34) (letter2 6))
(interaction (var1 h24) (letter1 3) (var2 v37) (letter2 1))
(interaction (var1 h24) (letter1 4) (var2 v39) (letter2 2))

(interaction (var1 h25) (letter1 1) (var2 v2) (letter2 2))
(interaction (var1 h25) (letter1 2) (var2 v4) (letter2 5))
(interaction (var1 h25) (letter1 3) (var2 v7) (letter2 3))
(interaction (var1 h25) (letter1 4) (var2 v10) (letter2 1))

(interaction (var1 h26) (letter1 1) (var2 v15) (letter2 6))
```

```
(interaction (var1 h26) (letter1 2) (var2 v18) (letter2 4))
(interaction (var1 h26) (letter1 3) (var2 v20) (letter2 1))
(interaction (var1 h26) (letter1 4) (var2 v23) (letter2 3))
(interaction (var1 h26) (letter1 5) (var2 v26) (letter2 2))
(interaction (var1 h26) (letter1 6) (var2 v29) (letter2 1))
(interaction (var1 h26) (letter1 7) (var2 v31) (letter2 5))

(interaction (var1 h27) (letter1 1) (var2 v37) (letter2 2))
(interaction (var1 h27) (letter1 2) (var2 v39) (letter2 3))

(interaction (var1 h28) (letter1 1) (var2 v2) (letter2 3))
(interaction (var1 h28) (letter1 2) (var2 v4) (letter2 6))
(interaction (var1 h28) (letter1 3) (var2 v7) (letter2 4))
(interaction (var1 h28) (letter1 4) (var2 v10) (letter2 2))
(interaction (var1 h28) (letter1 5) (var2 v13) (letter2 1))

(interaction (var1 h29) (letter1 1) (var2 v18) (letter2 5))
(interaction (var1 h29) (letter1 2) (var2 v20) (letter2 2))
(interaction (var1 h29) (letter1 3) (var2 v23) (letter2 4))
(interaction (var1 h29) (letter1 4) (var2 v26) (letter2 3))
(interaction (var1 h29) (letter1 5) (var2 v29) (letter2 2))

(interaction (var1 h30) (letter1 1) (var2 v35) (letter2 1))
(interaction (var1 h30) (letter1 2) (var2 v37) (letter2 3))
(interaction (var1 h30) (letter1 3) (var2 v39) (letter2 4))

(interaction (var1 h31) (letter1 1) (var2 v7) (letter2 5))
(interaction (var1 h31) (letter1 2) (var2 v10) (letter2 3))
(interaction (var1 h31) (letter1 3) (var2 v13) (letter2 2))
(interaction (var1 h31) (letter1 4) (var2 v16) (letter2 1))
(interaction (var1 h31) (letter1 5) (var2 v18) (letter2 6))
(interaction (var1 h31) (letter1 6) (var2 v20) (letter2 3))

(interaction (var1 h32) (letter1 1) (var2 v26) (letter2 4))
(interaction (var1 h32) (letter1 2) (var2 v29) (letter2 3))
(interaction (var1 h32) (letter1 3) (var2 v32) (letter2 1))
(interaction (var1 h32) (letter1 4) (var2 v35) (letter2 2))
(interaction (var1 h32) (letter1 5) (var2 v37) (letter2 4))
(interaction (var1 h32) (letter1 6) (var2 v39) (letter2 5))

(interaction (var1 h33) (letter1 1) (var2 v2) (letter2 5))
(interaction (var1 h33) (letter1 2) (var2 v5) (letter2 1))
(interaction (var1 h33) (letter1 3) (var2 v7) (letter2 6))
(interaction (var1 h33) (letter1 4) (var2 v10) (letter2 4))
(interaction (var1 h33) (letter1 5) (var2 v13) (letter2 3))
(interaction (var1 h33) (letter1 6) (var2 v16) (letter2 2))
(interaction (var1 h33) (letter1 7) (var2 v18) (letter2 7))

(interaction (var1 h34) (letter1 1) (var2 v24) (letter2 1))
(interaction (var1 h34) (letter1 2) (var2 v26) (letter2 5))
(interaction (var1 h34) (letter1 3) (var2 v29) (letter2 4))
(interaction (var1 h34) (letter1 4) (var2 v32) (letter2 2))
(interaction (var1 h34) (letter1 5) (var2 v35) (letter2 3))
(interaction (var1 h34) (letter1 6) (var2 v37) (letter2 5))
(interaction (var1 h34) (letter1 7) (var2 v39) (letter2 6))

(interaction (var1 h35) (letter1 1) (var2 v2) (letter2 6))
(interaction (var1 h35) (letter1 2) (var2 v5) (letter2 2))
(interaction (var1 h35) (letter1 3) (var2 v7) (letter2 7))

(interaction (var1 h36) (letter1 1) (var2 v13) (letter2 4))
(interaction (var1 h36) (letter1 2) (var2 v16) (letter2 3))
```

```
    (interaction (var1 h37) (letter1 1) (var2 v21) (letter2 1))
    (interaction (var1 h37) (letter1 2) (var2 v24) (letter2 2))
    (interaction (var1 h37) (letter1 3) (var2 v26) (letter2 6))
    (interaction (var1 h37) (letter1 4) (var2 v29) (letter2 5))
    (interaction (var1 h37) (letter1 5) (var2 v32) (letter2 3))

    (interaction (var1 h38) (letter1 1) (var2 v37) (letter2 6))
    (interaction (var1 h38) (letter1 2) (var2 v39) (letter2 7))

    (interaction (var1 h39) (letter1 1) (var2 v2) (letter2 7))
    (interaction (var1 h39) (letter1 2) (var2 v5) (letter2 3))

    (interaction (var1 h40) (letter1 2) (var2 v13) (letter2 5))
    (interaction (var1 h40) (letter1 3) (var2 v16) (letter2 4))
    (interaction (var1 h40) (letter1 5) (var2 v21) (letter2 2))
    (interaction (var1 h40) (letter1 6) (var2 v24) (letter2 3))
    (interaction (var1 h40) (letter1 7) (var2 v26) (letter2 7))
    (interaction (var1 h40) (letter1 8) (var2 v29) (letter2 6))

    (interaction (var1 h41) (letter1 2) (var2 v37) (letter2 7))
    (interaction (var1 h41) (letter1 3) (var2 v39) (letter2 8))

    (word  "as") (word "soi") (word "gener") (word "ecrouer")
    (word  "au") (word "sur") (word "gnous") (word "enferme")
    (word  "ca") (word "ton") (word "maree") (word "enumere")
    (word  "en") (word "tri") (word "ornas") (word "esperes")
    (word  "le") (word "ulm") (word "relue") (word "granule")
    (word  "no") (word "anet") (word "satin") (word "mailles")
    (word  "on") (word "arts") (word "verre") (word "origine")
    (word  "re") (word "crin") (word "apotre") (word "perores")
    (word  "si") (word "emet") (word "enemas") (word "pimente")
    (word  "te") (word "erre") (word "enigme") (word "reussie")
    (word  "ut") (word "lino") (word "eperon") (word "uretere")
    (word  "are") (word "niee") (word "essore") (word "valeurs")
    (word  "emu") (word "nuit") (word "froler") (word "assenees")
    (word  "eta") (word "reel") (word "genois") (word "braisent")
    (word  "eus") (word "sees") (word "isatis") (word "curetage")
    (word  "nes") (word "unie") (word "limais") (word "emergera")
    (word  "ost") (word "user") (word "murage")
    (word  "pas") (word "delta") (word "pietre")
    (word  "pre") (word "egale") (word "ruelle")
    (word  "ree") (word "emeus") (word "cendres")
    (word  "rue") (word "etire") (word "ecrites")
    (word "tente")
)


;#####################  Constraints propagation  #####################

(defrule PROPAG::propagation-1
; remove from the list of possible values of ?x2 all the words ?m which
;don't cross well
; with the word ?v
    (logical (level_search ?n))
    (not (level_search   ?n1&:(> ?n1 ?n)))
    (var (name ?x1) (value ?v&~nil) (level ?n))
   ?f <-  (var (name ?x2) (level ?lev)(value nil)(possible-values $?poss))
    (not (var (name ?x2) (level ?mm&:(> ?mm ?lev))))
    (or (interaction (var1 ?x1) (letter1 ?num1) (var2 ?x2) (letter2 ?num2))
        (interaction (var2 ?x1) (letter2 ?num1) (var1 ?x2) (letter1 ?num2))
)
    (word  ?m&:(member$ ?m $?poss))
    (test (neq (sub-string ?num1 ?num1 ?v) (sub-string ?num2 ?num2 ?m)))
```

```
=>
   (bind ?num (member$ ?m $?poss))
   (if (= ?n ?lev)
   then (modify ?f (possible-values (delete$ $?poss ?num ?num)))
   else  (duplicate  ?f  (level  ?n)(possible-values  (delete$  $?poss  ?num
?num))))
)

(defrule PROPAG::propagation-2
; remove the word ?v from the list of possible values of ?x2
; ?v is the value of the latest assigned variable ?x1
   (logical (level_search  ?n))
   (not (level_search  ?n1&:(> ?n1 ?n)))
   (var (name ?x1) (value ?v&~nil) (level ?n))
   ?f <-  (var (name ?x2) (level ?lev)(value nil)(possible-values $?poss))
   (not (var (name ?x2) (level ?mm&:(> ?mm ?lev))))
   (test (member$ ?v $?poss))
=>
   (bind ?num (member$ ?v $?poss))
   (if (= ?n ?lev)
   then (modify ?f (possible-values (delete$ $?poss ?num ?num)))
   else  (duplicate  ?f  (level  ?n)(possible-values  (delete$  $?poss  ?num
?num))))
)
```