The GNU Prolog finite domain constraints solver

The Sudoku 4×4 in Prolog

	2		
1			
			4
		1	

$\text{sudoku4}(L) \leftarrow \text{generate}(L) \land \text{test}(L).$

$\text{generate}(L) \leftarrow \text{all\_member}(L,[1,2,3,4]).$

(3) $\text{test}(X_{11}, X_{12}, X_{13}, X_{14}, X_{21}, X_{22}, \ldots, X_{42}, X_{43}, X_{44}) \leftarrow$
$\land\ \text{all\_diff}([X_{11}, X_{12}, X_{13}, X_{14}]) \land \text{all\_diff}([X_{21}, X_{22}, X_{23}, X_{24}])$
$\land\ \text{all\_diff}([X_{31}, X_{32}, X_{33}, X_{34}]) \land \text{all\_diff}([X_{41}, X_{42}, X_{43}, X_{44}])$
$\land\ \text{all\_diff}([X_{11}, X_{21}, X_{31}, X_{41}]) \land \text{all\_diff}([X_{14}, X_{24}, X_{34}, X_{44}])$
$\land\ \text{all\_diff}([X_{12}, X_{22}, X_{32}, X_{42}]) \land \text{all\_diff}([X_{13}, X_{23}, X_{33}, X_{43}])$
$\land\ \text{all\_diff}([X_{11}, X_{12}, X_{21}, X_{22}]) \land \text{all\_diff}([X_{13}, X_{14}, X_{23}, X_{24}])$
$\land\ \text{all\_diff}([X_{31}, X_{32}, X_{41}, X_{42}]) \land \text{all\_diff}([X_{33}, X_{34}, X_{43}, X_{44}]).$

(4) $\text{all\_member}(L, D) \leftarrow$
$L = [\,] \lor L = [V|R] \land \text{member}(V, D) \land \text{all\_member}(R, D).$

(5) $\text{all\_diff}(L) \leftarrow L = [\,] \lor L = [V|R] \land \neg\text{member}(V, R) \land \text{all\_diff}(R).$

Query:

$\text{sudoku4}([X_{11},2,X_{13},X_{14},1,X_{22},X_{23},X_{24},X_{31},X_{32},X_{33},4,X_{41},X_{42},1,X_{44}]).$

The Sudoku 4×4 in Prolog : The basic "Generate & test" solution

The search tree has $4^{12} \approx 17$ million leaves !!

The Sudoku 4×4 in Prolog : A more efficient version

Generate only valide lines
(somehow, the “generate” and “test” parts are interleaved).

$\text{sudoku4}(L) \leftarrow \text{generate}(L) \wedge \text{test}(L).$

$\text{generate}(X_{11},X_{12},X_{13},\ldots,X_{42},X_{43},X_{44}) \leftarrow$

$\quad \text{all\_member\_diff}([X_{11},X_{12},X_{13},X_{14}]) \wedge \text{all\_member\_diff}([X_{21},X_{22},X_{23},X_{2}$

$\quad \wedge \text{all\_member\_diff}([X_{31},X_{32},X_{33},X_{34}]) \wedge \text{all\_member\_diff}([X_{41},X_{42},X_{43},X_{44}]$

$\text{test}(X_{11},X_{12},X_{13},\ldots,X_{42},X_{43},X_{44}) \leftarrow$

$\quad \text{all\_diff}([X_{11},X_{21},X_{31},X_{41}]) \wedge \text{all\_diff}([X_{14},X_{24},X_{34},X_{44}])$

$\quad \wedge\ \text{all\_diff}([X_{12},X_{22},X_{32},X_{42}]) \wedge \text{all\_diff}([X_{13},X_{23},X_{33},X_{43}])$

$\quad \wedge\ \text{all\_diff}([X_{11},X_{12},X_{21},X_{22}]) \wedge \text{all\_diff}([X_{13},X_{14},X_{23},X_{24}])$

$\quad \wedge\ \text{all\_diff}([X_{31},X_{32},X_{41},X_{42}]) \wedge \text{all\_diff}([X_{33},X_{34},X_{43},X_{44}]).$

$\text{all\_member\_diff}(L,D) \leftarrow L = [\,] \vee$

$L = [V|R] \wedge \text{select}(V,D,S) \wedge \text{all\_member\_diff}(R,S).$

$\text{all\_diff}(L) \leftarrow L = [\,] \vee L = [V|R] \wedge \neg\text{member}(V,R) \wedge \text{all\_diff}(R).$

Remark. $\text{select}(V, D, RD)$ is true if $V \in D$ and $RD = D \backslash V$.

The Sudoku 4×4 in Prolog : With the constraint solver

$$
\begin{aligned}
&\text{sudoku4\_fd}(L) \leftarrow L = [X_{11},X_{12},X_{13},\ldots,X_{42},X_{43},X_{44}] \\
&\quad \wedge \text{fd\_domain}(L,[1,2,3,4]) \\
&\quad \wedge \text{fd\_all\_diff}([X_{11},X_{12},X_{13},X_{14}]) \wedge \text{fd\_all\_diff}([X_{21},X_{22},X_{23},X_{24}]) \\
&\quad \wedge \text{fd\_all\_diff}([X_{31},X_{32},X_{33},X_{34}]) \wedge \text{fd\_all\_diff}([X_{41},X_{42},X_{43},X_{44}]) \\
&\quad \wedge \text{fd\_all\_diff}([X_{11},X_{21},X_{31},X_{41}]) \wedge \text{fd\_all\_diff}([X_{14},X_{24},X_{34},X_{44}]) \\
&\quad \wedge \text{fd\_all\_diff}([X_{12},X_{22},X_{32},X_{42}]) \wedge \text{fd\_all\_diff}([X_{13},X_{23},X_{33},X_{43}]) \\
&\quad \wedge \text{fd\_all\_diff}([X_{11},X_{12},X_{21},X_{22}]) \wedge \text{fd\_all\_diff}([X_{13},X_{14},X_{23},X_{24}]) \\
&\quad \wedge \text{fd\_all\_diff}([X_{31},X_{32},X_{41},X_{42}]) \wedge \text{fd\_all\_diff}([X_{33},X_{34},X_{43},X_{44}]) \\
&\quad \wedge \text{fd\_labeling}(L).
\end{aligned}
$$

The Sudoku 4×4 in Prolog : Comparison of the 3 versions

- first version: implemented without thinking about how prolog evaluates the queries, too slow.
- second version: faster, but the programmer had to think more about prolog's evaluation mechanism – this is not the aim with logic programming.
- third version: even faster, and written without thinking about how constraints are solved.
  It uses an external constraint solver.

A quick overview of the constraint solver

1. each variable receives an initial ***domain***;
   (above: the call fd_domain($L, [1, 2, 3, 4]$) associates the domain $\{1, 2, 3, 4\}$ to all variables in L)

A quick overview of the constraint solver : The basic mechanism

1. each variable receives an initial ***domain***;
   (above: the call fd_domain($L, [1, 2, 3, 4]$) associates the domain $\{1, 2, 3, 4\}$ to all variables in L)
2. every encountered ***constraint*** is stored

A quick overview of the constraint solver : The basic mechanism

1. each variable receives an initial ***domain***;
   (above: the call fd_domain($L, [1, 2, 3, 4]$) associates the domain $\{1, 2, 3, 4\}$ to all variables in L)
2. every encountered ***constraint*** is stored
   with domain reduction based on local consistency conditions if possible; above:
   - X_{12} instanciated to 2
   - constraint fd_all_diff($[X_{11}, X_{12}, X_{13}, X_{14}]$)

   $\Rightarrow$ the value 2 is removed from the domains of X_{11}, X_{13}, X_{14}
3. the call fd_labeling triggers the external constraint solver.

A quick overview of the constraint solver : Domains

A domain D_X is associated with each variable X that appears in a constraint.

Initially: $D_X = [0, \ldots, \text{fd\_max\_integer}] \subseteq \mathbf{N}^+$

```
| ?- X = Y.
Y = X
yes
```

```
| ?- X #= Y.
X = _#0(0..268435455)
Y = _#0(0..268435455)
yes
```

A quick overview of the constraint solver : Domains

A domain D_X is associated with each variable X that appears in a constraint.
Initially: $D_X = [0, \ldots, \text{fd\_max\_integer}] \subseteq \mathbf{N}^+$

```
| ?- X = Y.
Y = X
yes
```

```
| ?- X #= Y.
X = _#0(0..268435455)
Y = _#0(0..268435455)
yes
```

```
| ?- X\=Y.
no
```

```
| ?- X\==Y.
yes
```

```
| ?- X #\= Y.
X = _#2(0..268435455)
Y = _#20(0..268435455)
yes
```

A quick overview of the constraint solver : Domains

The first effect of a constraint is to reduce the domain of the variables:

```
| ?- X + Y #= 5.
X = _#21(0..5)
Y = _#39(0..5)
yes
```

A quick overview of the constraint solver : Domains

The first effect of a constraint is to reduce the domain of the variables:

```
| ?- X + Y #= 5.
X = _#21(0..5)
Y = _#39(0..5)
yes
```

```
| ?- X #< 3.
X = _#2(0..2)
yes
```

A quick overview of the constraint solver : Domains

The first effect of a constraint is to reduce the domain of the variables:

```
| ?- X + Y #= 5.
X = _#21(0..5)
Y = _#39(0..5)
yes
```

```
| ?- X #< 3.
X = _#2(0..2)
yes
```

```
| ?- X #< 3 , X+Y #= 6.
X = _#2(0..2)       Y = _#41(4..6)
yes
```

A quick overview of the constraint solver : Domains

```
| ?- X #< 3 , write(X) , nl , write(Y)
, X + Y #= 6.
_#2(0..2)
_22
X = _#2(0..2)       Y = _#41(4..6)
yes
```

A quick overview of the constraint solver : Domains

```
| ?- X #< 3 , write(X) , nl , write(Y)
, X + Y #= 6.
_#2(0..2)
_22
X = _#2(0..2)     Y = _#41(4..6)
yes
```

```
| ?- X #< 2 , Y #< 2 , Z #< 2
, X #\= Y , X #\= Z , Y #\= Z.
X = _#2(0..1)  Y = _#22(0..1)  Z = _#42(0..1)
yes
```

A quick overview of the constraint solver : Domains

Remarks:

- the predicates #=, #>, ... do not completely solve the constraints.
- the evaluation of each constraint C only eliminates from the domains of the variables values that do not appear in any solution of C:
  it ensure ***local consistency***
  (it is local to ***one*** constraint)

A quick overview of the constraint solver : Invoking the solver

The predicate fd_labeling solves all the constraints that have been ***posted*** :

```
| ?- X #< 3 , X + Y #= 6 , fd_labeling([X,Y]).
X = 0    Y = 6 ? ;
X = 1    Y = 5 ? ;
X = 2    Y = 4
yes
```

A quick overview of the constraint solver : Invoking the solver

```
| ?- X #< 2 , Y #< 2 , Z #< 2 , X #\= Y , X #\= Z , Y #\= Z
, fd_labeling([X,Y,Z]).
no
```

The actual constraint solving algorithm will not be studied here...

Remark: all constraints are simultaneously solved, not only the ones that involve the variables that appear in the parameter of fd_labeling:

```
| ?- X #< 2 , Y #< 2 , Z #< 2 , X #\= Y , X #\= Z , Y #\= Z
, fd_labeling([X,Y]).
no
```

A quick overview of the constraint solver : Other predicates

fd_domain(X, L): removes from the domain of X values that are not in L.

fd_domain_bool(L): removes from the domain of each variable in L values that are not in $\{0, 1\}$.

fd_all_different(L): constraints all variables in the list L to have different values.

`fd_all_different([X,Y,Z])` is equivalent to:

```
X #\= Y , X #\= Z , Y #\= Z
```

A quick overview of the constraint solver : Other predicates

```
| ?- fd_domain_bool([X,Y,Z]) , fd_all_different([X,Y,Z]).
X = _#0(0..1)   Y = _#18(0..1)   Z = _#36(0..1)
yes

| ?- fd_domain_bool([X,Y,Z]) , fd_all_different([X,Y,Z])
, fd_labeling([X,Y,Z]).
no
```

fd_atmost(N, L, V): imposes that at most N variables from the list L have value V.
There is also fd_atleast and fd_exactly.

A quick overview of the constraint solver

Constraint solving and backtracking The following program and query show that constraints posted in different branches of the evaluation tree are solved independently from one another:

```
p(X,Y,Z) :- fd_domain([X,Y,Z],1,2) , Y #\= Z
, q(X,Y,Z) , fd_labeling([X,Y,Z]).
q(X,Y,Z) :- member([U,V],[[X,Y],[X,Z]])
, U #\= V , fail.
q(_,_,_).
```

Then:

```
| ?- p(X,Y,Z).
X = 1   Y = 1   Z = 2 ? ;
X = 1   Y = 2   Z = 1 ? ...
yes
```

A quick overview of the constraint solver : Optimisation

The predicate fd_minimize returns the minimum value allowed for a variable X among those possible when constraints are solved with fd_labeling:

```
| ?- X + Y #= 10 , Y #< 3 , fd_minimize(fd_labeling([X,Y]),X)
X = 8    Y = 2
yes
```

How it works:

- each time fd_labeling($[X, Y]$) gives a solution $X = n$, the search is started again with a new constraint $X \#< n$;
- when a failure occurs (either because there are no remaining choice-points for Goal or because the added constraint is inconsistent with the rest of the store) the last solution is recomputed since it is optimal.

There is also fd_maximize.

A quick overview of the constraint solver : Optimisation

Example : graph coloring

```
| ?- X #\= Y , X #\= Z , X #< Max , Y #< Max , Z #< Max
, fd_minimize(fd_labeling([X,Y,Z]), Max).
```

Exercises

Exercise 1. A factory has four workers w1,w2,w3,w4 and four products p1,p2,p3,p4. The problem is to assign workers to products so that each worker is assigned to one product, each product is assigned to one worker, and the profit maximized. The profit made by each worker working on each product is given in the matrix:

	p1	p2	p3	p4
w1	7	1	3	4
w2	8	2	5	1
w3	4	3	7	2
w4	3	1	6	3

Exercise 2. Four roommates are subscribing to four newspapers. The table gives the amounts of time each person spends on each newspaper. Akiko gets up at 7:00, Bobby gets up at 7:15, Chloé gets up at 7:15, and Dola gets up at 8:00.

	The Guardian	Le Monde	El Pais	Die Taz
Albert	60	30	2	5
Bobby	75	3	15	10
Chloé	5	15	10	30
Dola	90	1	1	1

Nobody can read more than one newspaper at a time and at any time a newspaper can be read by only one person. The goal is to schedule the newspapers such that the four persons finish the newspapers at an earliest possible time.

Question 2.1. Describe a model of the problem using constraints. How many variables and how many constraints are they? What is the objective?

(Hint: use a variable that represents the latest finishing time.)

Question 2.2. Write a program to solve it using the constraint solver.

Exercises

Exercise 3. On an $8x8$ chessboard, we want to put eight queens so that no two queens threaten each other. A queen threatens every other piece that is on the same column, row, or diagonal.

Question 3.1. A solution can be defined as a permutation of the list $[1, \ldots, 8]$. How?

Question 3.2. Write a small program using the constraint solver to solve the problem when there are 4 queens to place on a 4×4 chessboard.

Question 3.3. Assuming an encoding as suggested with the previous question, how do you write, in the language of the constraint solver, the constraints between two queens represented by variables Q_i and $Q_{i+\delta}$, at positions i and $i + \delta$ in the list above?

Question 3.4. Write a predicate constraints/2 that recursively posts all constraints between a queen Q_i and the list of all queens with indices $i+1, \ldots, n$: constraints$(Q_i, [Q_{i+1}, \ldots, Q_n])$ is always true, and its execution must post all necessary constraints.

Exercises

Question 3.5. Finish the program with a recursive predicate that makes the necessary calls to constraints/2.

Exercises

Solution: *A program that solves the N-queen problem:*

(1) $consraints(Q, L) \leftarrow consraints(Q, L, 1).$

(2) $consraints(Q_1, L, \delta) \leftarrow L = []$

$\vee\ L = [Q_2|R] \wedge Q_1\ \#\backslash{=}\#\ Q_2 \wedge \delta\ \#\backslash{=}\#\ Q_1 - Q_2 \wedge \delta\ \#\backslash{=}\#\ Q_2 - Q_1$

$\wedge\ \delta_1\ is\ \delta + 1 \wedge consraints(Q_1, R, \delta_1).$

(3) $safe(L) \leftarrow L = [] \vee L = [Q|R] \wedge consraints(Q, R) \wedge safe(R).$

(4) $eightqueens(N, L) \leftarrow length(L, N) \wedge fd\_domain(L, 1, N)$

$\wedge\ safe(L) \wedge fd\_labeling(L).$

Exercises

Exercise 4. Graph coloring is an important combinatorial optimization problem. In this exercise, we assume that a graph is described by a predicate edge/2. We want to write a program, using the constraint solver, that computes a coloring of the graph with a minimum number of colors.

You can test your predicates on a small graph that you define yourself. Two bigger graphs are described in the files flat20_3_0.pland jean.pl, that you can download at the address www.irit.fr/~Jerome.Mengin/prolog/

Question 4.1. Write two predicates that create the lists of edges and of vertices of the graph. For instance, listVertices(L) must be called with an uninstantiated variable, and must then instantiate this variable with the list of all vertices of the graph described by edge/2.

Question 4.2. Write a predicate that generates an ***association list*** of pairs $[V, X_V]$, one for each vertex X, where X_V is a newly created variable that will be eventually instanciated with the color assigned to X.

Question 4.3. Extend the preceding predicate so that it also returns the list of the X_V's (without the associated vertices).

Question 4.4. Write a predicate that posts all constraints - one for each edge in the graph.

Question 4.5. Finish your program with a predicate that finds the minimum number of colors necessary to color the graph.