6.2 CLP in Prolog

Dienstag, 11. Juli 2017 09:00

Now we explain how CLP is implemented in Prolog.
The pred. symbols $=$, true, fail are pre-defined in Prolog. But we have to tell Prolog which constraint theories we want to use.
Prolog has a module-system of pre-defined libraries.

use_module : predicate to import all predicates defined in a module

To import the library with the predicate symbols from the constraint theory CT_{FD}, one can add the following directive to the Prolog-program:

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

Afterwards, Σ_{FD} and Δ_{FD} are available and one can use a constraint solver for CT_{FD}.
However, entailment is undecidable for CT_{FD}.
Thus: one can't decide automatically whether

$$CT_{FD} \cup \{\forall X\; X = X, \mathit{true}\} \models \exists\, CO \qquad (*)$$

Even for theories where this is decidable, it could be very inefficient.

Therefore, Prolog only <u>approximates</u> ($\ast$).

Instead of checking satisfiability of CO, it only checks <u>path consistency</u> of CO.

<u>Def 6.2.1</u> (Path Consistency)

Let $CO = \varphi_1 \wedge \ldots \wedge \varphi_m$ be a conjunction of constraints with $\varphi_i \in At(\Sigma_{FD}, \Delta_{FD}, \mathcal{V})$. Let $X_1, \ldots, X_n$ be the variables of CO. Let $D_1, \ldots, D_n$ be subsets of $\mathbb{Z}$. We say that $D_1, \ldots, D_n$ are <u>admissible domains</u> for $X_1, \ldots, X_n$ w.r.t. CO iff for all φ_i with $1 \le i \le m$ and all X_j with $1 \le j \le n$ we have: For all $a_j \in D_j$ there exist $a_1 \in D_1, \ldots, a_{j-1} \in D_{j-1}, a_{j+1} \in D_{j+1}, \ldots, a_n \in D_n$ such that $CT_{FD} \models \varphi_i[X_1/a_1, \ldots, X_n/a_n]$.

If there exist admissible domains $D_1, \ldots, D_n$ which are all non-empty, then CO is <u>path-consistent</u>.

Main difference between path-consistency and satisfiability is that for path-consistency one regards all constraints separately.

<u>Ex. 6.2.2</u>

Typically, one first sets all $D_j = \mathbb{Z}$. Afterwards one iterates through all constraints and all variables and reduces the domains D_j. This is repeated until the domains D_j do not change anymore.

$$X_1 \ \#> 5 \ \wedge \ X_1 \ \#< X_2 \ \wedge \ X_2 \ \#< 9$$

- We start with $D_1 = \mathbb{Z}$ and $D_2 = \mathbb{Z}$.
- Now we regard the constraint $X_1 \ \#> 5$ and variable X_1:
  We have to remove all elements from D_1 where the constraint does not hold.
  $D_1 = \{6, 7, \dots\}$, $D_2 = \mathbb{Z}$
- Now we regard constraint $X1 \ \#< X_2$ and var. $X1$:
  Is it true that for every $a_1 \in D_1$ there is an $a_2 \in D_2$ with $a_1 \ \#< a_2$? Yes.
- Now constr. $X_1 \ \#< X2$ and var. X_2:
  Is it true that for every $a_2 \in D_2$ there is an $a_1 \in D_1$ with $a_1 \ \#< a_2$? No.
  $D_1 = \{6, 7, \dots\}$ $D_2 = \{7, 8, \dots\}$
- Now we regard $X2 \ \#< 9$ and var. $X2$:
  $D_1 = \{6, 7, \dots\}$ $D_2 = \{7, 8\}$
- Now we again regard $X_1 \ \#> 5$. D_1, D_2 do not change.

- Then we regard X1 #< X2 and var. X1:

  $D_1 = \{6, 7\}$ $\quad\quad$ $D_2 = \{7, 8\}$

Afterwards, D_1 and D_2 do not change anymore ⟹ admissible domains.

Since $D_1 \neq \emptyset$ and $D_2 \neq \emptyset$, our conjunction of constraints is path-consistent (i.e., "true" in Prolog).

X1 in 6..7 ← pre-defined predicate "in" which abbreviates

X1 #>= 6, X1 #=< 7.

The predicate "in" can also be used by the programmer.

?- X1 in 6 .. sup, X1 #< X2, X2 in inf .. 8.

sup: ∞ $\quad\quad$ inf: $-\infty$

There exists a pre-defined pred. label in clpfd which forces Prolog to enumerate solutions.

?- X1 #> 5, X1 #< X2, X2 #< 9, label([X1, X2]).

X1 = 6, X2 = 7 ;

$X1 = 6, \; X2 = 8$;

$X1 = 7, \; X2 = 8$

↖ In this way, one can print out answer substitutions instead of answer constraints.

However, this only works if the admissible domains are finite.

`?- X1 #> 5, X1 #< X2, label([X1, X2]).`

error

<u>Ex. 6.2.3</u> There are examples where CO are path-consistent, but unsatisfiable.

`?- X1 #> X2, X1 #=< X2.`

Prolog starts with $D_1 = \mathbb{Z}$, $D_2 = \mathbb{Z}$.

- Now we start with $X1 \ \#> X2$.

  For every $a_1 \in D_1$ there is an $a_2 \in D_2$ with $a_1 \ \#> a_2$.

  — " — $a_2 \in D_2$ —— " — $a_1 \in D_1$ with $a_1 \ \#> a_2$.

- Then we regard $X1 \ \#=< X2$.

  Again, D_1 and D_2 remain unchanged.

Typical programs that are easy to compute with CLP:

<u>Ex. 6.2.4.</u> n-queens problem

1 2 3 4

Ex. 6.2.4. n-queens problem

chess board of size $n \times n$

Goal: place n queens on the board such that the queens cannot take each other

	1	2	3	4
1			X	
2	X			
3				X
4		X		

$[2,4,1,3]$

There must not be more than one queen in any row, any column, and any diagonal.

We represent the positions of the queens by a list $[x_1, \dots, x_n]$ which means that the queen in column i is in row x_i.

?- queens(4, L)
L = [2,4,1,3]

Here, L should be the possible positions of queens on a 4x4 chess board

CLP program on Slide 47

- L must be a permutation of the numbers from 1..N.
  - pre-def. predicate "ins":
    $[X_1, \dots, X_k]$ ins 1..N iff
    X_j in 1..N for all $1 \le j \le k$.
  - pre-def. predicate all_different
- safe(L) is needed to ensure that no two queens are on the same diagonal

safe(L) ensures that no queen in L can take a queen on the right of her

- safe_between (X, L, M) is true iff the queen in row X cannot take any queen in the list L provided that M is the number of columns between the queen X and the first queen in L.

In the example we would first have the query

safe_between (2, [4, 1, 3], 1)

safe_between (2, [1, 3], 2)

safe_between (2, [3], 3)