MPRI 2015-2016
Constraint Programming (2.35.1) exam
Reified constraints

Sylvain Soliman

November 24, 2015

1 Lexicographic order constraint

Suppose given a CLP($\mathcal{FD}$) language, like SWI-Prolog, with finite domain constraints: `X #= Y + Z`, `X #\= Y + Z`, `X #=< Y + Z`, `X #>= Y + Z` and their counterparts without `Z`.

Question 1.1

Write a non-deterministic CLP program `lexleq(L1, L2)` *enforcing the lexical order constraint* ***between two tuples*** *represented as lists with the same number of $\mathcal{FD}$ variables, using only constraints given above.*

Question 1.2

Write the first two iterations of the T_P operator on the above program.

Suppose now that reified constraints are available for each of the $\mathcal{FD}$ constraints given above, and their combination using `#\/, #/\, #<==>`, etc.

Question 1.3

Write a deterministic CLP program enforcing the lexical order constraint between two lists of $\mathcal{FD}$ variables.

Question 1.4

Give an example where the second program (the deterministic one) propagates strictly more than the first one (assuming arc-consistency on the elementary constraints).

2 CC definition and semantics

Suppose now again that only $\mathcal{FD}$ constraints are given, but this time in a CC($\mathcal{FD}$) concurrent constraint programming language, i.e., *tells* and *asks* are given for `X #= Y + Z`, `X #\= Y + Z`, `X #=< Y + Z`, `X #>= Y + Z` and their counterparts without `Z`.

When there is no ambiguity, *tells* can be omitted, but with careful parenthesizing.

For the reasons illustrated in the previous section, we want to add to our language reified constraints of the form:

```
B #<==> c
```

where `B` is a boolean variable with domain $\{0, 1\}$, and `c` is one of the above $\mathcal{FD}$ constraints. The reified constraint enforces that the boolean `B` equals 1 if and only if the constraint `c` is satisfied.

Question 2.1

Write a deterministic (without the choice operator +) CC($\mathcal{FD}$) program P defining the constraint `B #<==>X #>=Y` *as a three-argument predicate, for the $\mathcal{O}_{ts}$ observable.*

Question 2.2

Give a denotational semantics $[\![P,$ `B #<==>X #>=Y` $]\!]$ of the constraint defined above, related to the $\mathcal{O}_{ts}$ observable. Recall precisely how these two semantics are linked.

Note that it will not be necessary to decompose the sets $\uparrow c$ and $\downarrow c$ for the $\mathcal{FD}$ constraints considered, but that possible simplifications should be done.

Question 2.3

Write a non-deterministic version of program P, using +.

What can you say about its observables "of interest"?

3 LCC definition and semantics

Let us now consider how to implement the same reified constraints `B #<==>c` as in the previous section, but in LCC($\mathcal{H}$). We suppose that $\mathcal{FD}$ variables are represented, as one of you suggested during class, through synchronization constraints corresponding to some upper (`ub`) and lower bounds (`lb`) and not only the least upper bound (`max`) and greatest lower bound (`min`) as in class.

$$domain(X, Min, Max) = tell(lb(X, Min)) \parallel tell(ub(X, Max))$$

Note that now, you can have several domain constraints on an $\mathcal{FD}$ variable.

Question 3.1

Write the predicate `geq_x(X, Y)` corresponding to the action on `X` of the tell *for `X #>=Y`.*

Question 3.2

Supposing that the above tell *is given, write the $geq(B, X, Y)$ program corresponding to `B #<==>X #>=Y` in LCC($\mathcal{H}$).*

The logical semantics of our program might use, contrary to what we did in class, Intuitionistic Logic (LJ, not the linear one, ILL).

Question 3.3

Give a sound logical semantics in LJ of LCC agents, and prove its soundness.

Give an example of non-completeness of that semantics with respect to an observable of your choice.

4 Internalizing search as reified constraints

We have seen briefly in class that any search strategy can be internalized as constraint propagation and labelling over boolean variables and reified constraints.

Here is an SWI-Prolog program that corresponds to the CLPZinc encoding of a dichotomic search of a variable between two bounds.

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

dichotomic_search(X, Min, Max) :-
   X in Min..Max,
   MaxIter is ceiling(log(Max - Min + 1)/log(2)),
   dichotomic_search_rec(X, L, MaxIter),
   label_and_indexicals(X, L).

dichotomic_search_rec(_, [], 0).
dichotomic_search_rec(X, [MinX, MaxX, B | L], N) :-
   N > 0,
   % in FD expressions / is the Euclidean division
   B #<==> X #> (MinX + MaxX)/2,
   M is N - 1,
   dichotomic_search_rec(X, L, M).

label_and_indexicals(_, []).
label_and_indexicals(X, [Min, Max, B | L]) :-
   % fd_inf and fd_sup are indexicals on the current
   % min and max values in the domain of an FD variable
   fd_inf(X, Min),
   fd_sup(X, Max),
   (
      var(B)   % B not instantiated yet
   ->
      format("~w =< X =< ~w~n", [Min, Max]),
      label([B]),
      label_and_indexicals(X, L)
   ;
      true
   ).
```

Question 4.1

Draw the search trees corresponding to the execution of the goals:

- *`dichotomic_search(X, 0, 4).`*
- *`X #\=2, dichotomic_search(X, 0, 4).`*

On top of the obvious gain to target any CP solver with the same CLPZinc source, one of the benefits of this approach is that there can be propagation from the current search state to the strategy.

Question 4.2

Give an example where the use of equivalence constraints `B #<==>c` allows for a better propagation than the usual strategy-directed search that corresponds to `B #==>c`.