Constraint Logic Programming and Concurrent Constraint Programming

Notes by Catuscia Palamidessi

1 Constraint Logic Programming

Some years ago Jaffar and Lassez proposed the paradigm of Constraint Logic Programming (CLP, [1]), an extension to logic programming which gained immediate popularity in the community. The extension consists of considering an arbitrary domain of data instead of the standard Herbrand universe, and by replacing the concept of unification with the concept of constraint over such domain. The unifiability test becomes a test of consistency of the new constraint with respect to the constraints previously established.

More formally, CLP clauses and goals have the form

$$p(x_1, \ldots, x_k) \leftarrow c_1, \ldots, c_m \diamondsuit B_1, \ldots, B_n$$

and

$$\leftarrow c_1, \ldots, c_m \diamondsuit A_1, \ldots, A_n$$

respectively, where the c_i's are a sequence of constraints, and the A_i's and B_i's are sequences of atoms. The computation step is defined as follows: let $A_i = p(t_1, \ldots, t_k)$ be the selected atom in the goal

$$\leftarrow c_1, \ldots, c_m \diamondsuit A_1, \ldots, A_n$$

and let

$$p(x_1, \ldots, x_k) \leftarrow c'_1, \ldots, c'_r \diamondsuit B_1, \ldots, B_s$$

be a renamed clause of the program, such that

$$c_1 \wedge \ldots \wedge c_m \wedge c'_1 \wedge \ldots \wedge c'_r \wedge x_1 \doteq t_1 \wedge \ldots \wedge x_k \doteq t_k$$

is satisfiable (consistent) with respect to a given theory, where '$\wedge$' is the logical conjunction. Then the new goal becomes

$$\leftarrow c_1 \wedge \ldots \wedge c_m \wedge c'_1 \wedge \ldots \wedge c'_r \wedge x_1 \doteq t_1 \wedge \ldots \wedge x_k \doteq t_k \diamondsuit A_1, \ldots, A_{i-1}, B_1, \ldots, B_s, A_{i+1}, \ldots, A_n.$$

The computation *succeeds* when the atom part of the goal is empty, and in that case the constraint part is called the *computed constraint*. It *fails* when the atom part is not empty, and there are no clauses in the program for which a computation step can be made, with the selected atom.

Of course, the semantics is parametric with respect to the theory in which constraints are evaluated.

2 Concurrent constraint programming

Concurrent constraint programming was proposed in [2, 3, 4].

2.1 The language

Processes are described by the following grammar

$$\begin{array}{rl} \textit{Programs} & D ::= \epsilon \mid p(\bar{x}) \text{ :- } A \mid D, D \\ \textit{Agents} & A ::= \mathbf{Stop} \mid \mathbf{tell}(c) \mid \sum_{i=1}^{n} \mathbf{ask}(c_i) \rightarrow A_i \mid A \parallel A \mid \exists_x A \mid p(\bar{x}) \end{array}$$

The agent **Stop** represents successful termination. The basic actions are given by the $\mathbf{ask}(c)$ and $\mathbf{tell}(c)$ constructs, where c is a constraint. These actions work on a common *store* which represents the constraints extablished until that moment. The action $\mathbf{ask}(c)$ is a test (guard) on the current store and its execution does not modify the store. We say that $\mathbf{ask}(c)$ is *enabled* in d iff $d \Rightarrow c$. The action $\mathbf{tell}(c)$ sets the store to $c \wedge d$ (note that in this model the store

rR1	$\langle \mathbf{tell}(c), d\rangle \longrightarrow \langle \mathbf{Stop}, c \wedge d\rangle$	
rR2	$\langle \sum_{i=1}^{n} \mathbf{ask}(c_i) \rightarrow A_i, d\rangle \longrightarrow \langle A_j, d\rangle$	$j \in [1, n]$ *and* $d \Rightarrow c_j$
rR3	$\dfrac{\langle A, c\rangle \longrightarrow \langle A', c'\rangle}{\begin{array}{c}\langle A \parallel B, c\rangle \longrightarrow \langle A' \parallel B, c'\rangle \\ \langle B \parallel A, c\rangle \longrightarrow \langle B \parallel A', c'\rangle\end{array}}$	
rR4	$\dfrac{\langle A, d \wedge \exists_x c\rangle \longrightarrow \langle B, d' \wedge \exists_x c\rangle}{\langle \exists_x^d A, c\rangle \longrightarrow \langle \exists_x^{d'} B, c \wedge \exists_x d'\rangle}$	
rR5	$\langle p(\bar{x}), c\rangle \longrightarrow \langle A[\bar{y}/\bar{x}], c\rangle$	$p(\bar{y})$:- A is the renaming of a declaration in D

Table 1: The Transition System, w.r.t. a given set of declarations D

can become inconsistent, i.e. *false*. We could require that the computation ends when this happens). The *guarded choice* agent $\sum_{i=1}^{n} \mathbf{ask}(c_i) \rightarrow A_i$ selects nondeterministically one $\mathbf{ask}(c_j)$ which is enabled, and then behaves like A_j. If no guards are enabled, then it *suspends*, waiting for other (parallel) agents to add information to the store. Parallel composition is represented by $\parallel$. The situation in which all components of a system of parallel agents suspend is called *global suspension* or *deadlock*. The agent $\exists_x A$ represents an agent A with x considered *local* to A. Finally, the agent $p(\bar{x})$ is a procedure call, where p is the name of the procedure and $\bar{x}$ is the list of the actual parameters. The meaning of $p(\bar{x})$ is given by a procedure declaration of the form $p(\bar{y})$:- A, where $\bar{y}$ is the list of the formal parameters. We assume that there is one and only one procedure declaration for every procedure name; this is not a restriction with respect to concurrent logic languages, as the presence of alternative clauses can be simulated by the choice construct.

2.2 Operational semantics

The operational model of ccp, informally introduced above, is described by a transition system $T = (Conf, \longrightarrow)$. The configurations (in) *Conf* are pairs consisting of a process and a constraint representing the current store, and the transition relation is given in Table 1. Note that in order to describe the operational semantics we had to add to the sintax an agent of the form $\exists_x^d A$. This represents an agent A with local variable x and local store d. We assume that $\exists_x A$ stands for $\exists_x^{true} A$.

References

[1] Jaffar, J. and Lassez, J.-L. Constraint logic programming. In *Proc,. of ACM Symposium on Principles of Programming Languages*, pages 111–119, 1987.

[2] Saraswat, V. *Concurrent Constraint Programming.* PhD thesis, Carnegie-Mellon University. In ACM distinguished dissertation series. The MIT Press, 1993.

[3] Saraswat, V. and Rinard, M. Concurrent constraint programming. In *Proc. of the seventeenth ACM Symposium on Principles of Programming Languages*, pages 232–245, 1990.

[4] Saraswat, V., Rinard, M., and Panangaden, P. Semantics foundations of Concurrent Constraint Programming. In *Proc. of the eighteenth ACM Symposium on Principles of Programming Languages*, 1991.