Compositional Analysis for Concurrent Constraint Programming*

Moreno Falaschi† Maurizio Gabbrielli‡ Kim Marriott§ Catuscia Palamidessi¶

Abstract

We propose a framework for the analysis of concurrent constraint programming (ccp). Our approach is based on simple denotational semantics which approximate the usual semantics in the sense that they give a superset of the input-output relation of a ccp program. Analyses based on these semantics can be easily and efficiently implemented using standard techniques from the analysis of logic programs.

1 Introduction

Concurrent constraint programming (ccp) [12, 13, 14] is a new programming paradigm which elegantly combines logical concepts and concurrency mechanisms. The computational model of ccp is based on the notion of *constraint system*, which consists of a set of constraints and an *entailment* (implication) relation. Processes interact through a common *store*. Communication is achieved by *telling* (adding) a given constraint to the store, and by *asking* (checking whether the store entails) a given constraint.

There is a growing interest in both the theoretical aspects and the practical applications of ccp languages. However, before their full potential can be realized there is a need for a framework in which to express and develop sophisticated dataflow analysis which can be used in compilers to produce more efficient target code and to help programmers to write error free code. The importance of dataflow analysis is amplified because of concurrency – both programmers and compilers find concurrent programs notoriously difficult to understand and reason about. However it is also hard for dataflow analyzers to understand concurrent programs and so it has proven difficult to develop simple, efficient and precise dataflow analyses for concurrent languages. The main contribution of this paper is to give a simple denotational semantics for ccp languages which we believe is a good basis for efficient yet precise analysis.

Existing denotational semantics for ccp are not a very suitable basis for analysis as they deal with complicated structures such as reactive sequences ([6]) or trace operators ([14]). These structures are known to be necessary in order to model the observables exactly, but their complexity makes it difficult to formalize analyses and prove their correctness. As a matter of fact, for the purpose of analysis, this complexity is often unnecessary. In fact analyses necessarily *approximate* the observables, in the sense that they lose information. This suggests that we can use simpler semantics which need not be correct in the classical sense, but only in the sense that

*This work has been partially supported by ESPRIT BRA 6707, *ParForce*.

†Dipartimento di Elettronica e Informatica, Università di Padova, Via Gradenigo 6/A, Padova, Italy.
e.mail: falaschi@di.unipi.it.

‡CWI, P.O. Box 4079, 1009 AB Amsterdam, The Netherlands. e.mail: gabbri@cwi.nl.

§Department of Computer Science, Monash University, Clayton 3168, Victoria, Australia.
e.mail: marriott@bruce.cs.monash.oz.au

¶Dipartimento di Informatica e Scienze dell'Informazione, Via Benedetto XV, 3, Genova, Italy.
e.mail: catuscia@di.unipi.it.

they are *correct approximations*.

The underlying idea is to build the denotation of a ccp process as a combination of the input-output constraint relation of the component processes. The resulting denotation is a set of input-output pairs. Such a semantics is a good basis for efficient dataflow analysis because it is similar to the standard structures used in the analysis of logic programming languages ([11]). Thus, the same implementation techniques such as memoization tables [7] and analysis domains and functions can be used with only slight modification.

Our denotational model approximates the standard operational semantics in the sense that it contains the input-output information of every possible computation. This means that it is a suitable basis for *universal* analysis in which we wish to verify that a certain property is satisfied by all computations. The converse does not hold, i.e. there may be pairs in the denotation of a process which do not correspond to any computation. This imprecision is not surprising as it is well known that pairs of constraints do not contain enough information to define a semantics which is both compositional and correct (in the classical sense).

From this denotational semantics we develop a generic dataflow analysis framework consisting of semantic equations which are parametric in the choice of constraints descriptions and the operations on these descriptions. A particular analysis is obtained by simply choosing a description domain and defining operations on the domain. Correctness of the resulting analysis is guaranteed by our construction and by results from abstract interpretation [5].

Previous related research includes the works of Codognet *et al.* [4] and Codish *et al.* [2, 3], who have investigated the analysis of concurrent logic languages, a particular subclass of the ccp languages. Our work primarily differs from these in the semantic basis of the abstract interpretation. Codognet *et al.* base their analysis on a complex and operational AND-OR tree semantics, while Codish *et al.* base their analysis on a transition system operational semantics. In [3] Codish *et al.* define a new operational semantics, which approximates the standard one, and which is confluent in the sense that different scheduling strategies give isomorphic results, thus simplifying the analysis. The loss of precision introduced by confluence is orthogonal to the loss of precision in our approach. Another denotational semantics for the analysis of a particular concurrent logic language was developed in [9]. This semantics is based on sequences-like structures and is therefore more complex than our semantics. Furthermore, our approach has the advantage that it can be easily implemented using standard techniques from logic programming.

The rest of this paper is organized as follows. In the next two sections we recall the definitions of a constraint system and of the ccp paradigm. In Section 4 we give the denotational semantics for the input-output relation associated to the standard operational model. In Section 5 we consider a variation of the notion of observables and the corresponding semantics. Finally in Section 6 we give the equations for the abstract denotational semantics. Section 7 concludes.

2 Constraint systems

In [13] constraint systems are defined following Scott's treatment of information systems [15]. The starting point is a set of simple constraints on which a compact entailment relation $\vdash$ is defined. Then a constraint system is constructed by considering sets of simple constraints and by extending the entailment relation on it. This construction is made in such a way that the resulting structure is a complete *algebraic* lattice, which ensures the effectiveness of the extended entailment relation. Here we abstract from this construction, and only consider the resulting structure.

Definition 2.1 A *constraint system* is a complete (algebraic) lattice $\langle \mathcal{C}, \leq, \wedge, \mathit{true}, \mathit{false} \rangle$ where $\wedge$ is the lub operation, and *true*, *false* are the least and the greatest elements of $\mathcal{C}$, respectively[1].

In order to treat the hiding operator of the language it will be helpful to introduce a sort of existential quantifier. In this framework it is convenient to formalize this notion by means of the theory of cylindric algebras, due to Henkin, Monk and Tarski [8]. This leads to the concept of *cylindric constraint system*. In the following, we assume given a (denumerable) set of variables *Var* with typical elements $x, y, z, \ldots$.

Definition 2.2 Let $\langle \mathcal{C}, \leq, \wedge, \mathit{true}, \mathit{false} \rangle$ be a constraint system. Assume that for each $x \in \mathit{Var}$ a function $\exists_x : \mathcal{C} \rightarrow \mathcal{C}$ is defined such that for any $c, d \in \mathcal{C}$:

(i) $\exists_x(c) \leq c$,

(ii) if $d \leq c$ then $\exists_x(d) \leq \exists_x(c)$,

(iii) $\exists_x(c \wedge \exists_x(d)) = \exists_x(c) \wedge \exists_x(d)$,

(iv) $\exists_x(\exists_y(c)) = \exists_y(\exists_x(c))$.

Then $\mathbf{C} = \langle \mathcal{C}, \leq, \wedge, \mathit{true}, \mathit{false}, \mathit{Var}, \exists \rangle$ is a *cylindric constraint system*.

In the following $\exists_x(c)$ is denoted by $\exists_x c$ with the convention that, in case of ambiguity, the scope of $\exists_x$ is limited to the first constraint subexpression. (So, for instance, $\exists_x c \wedge d$ stands for $\exists_x(c) \wedge d$.) Furthermore, for $\vec{x} = x_1, \ldots, x_n$ the notation $\exists_{\vec{x}}(A)$ stands for $\exists_{x_1}(\ldots \exists_{x_n}(A) \ldots)$.

In order to model parameter passing, it will be useful to consider the so-called *diagonal formulas* [8]. We assume that, for x, y ranging in *Var*, $\mathcal{A}$ contains the *diagonal elements* d_{xy} which satisfy the following properties.

(i) $d_{xx} = \mathit{true}$,

(ii) if $z \neq x, y$ then $d_{xy} = \exists_z(d_{xz} \wedge d_{zy})$,

(iii) if $x \neq y$ then $c \leq d_{xy} \wedge \exists_x(c \wedge d_{xy})$.

Note that if $\mathbf{C}$ models the equality theory, then the elements d_{xy} can be thought of as the formulas $x = y$. In the following, given $\vec{x} = x_1, \ldots, x_n$ and $\vec{y} = y_1, \ldots, y_n$, we use the notation $d_{\vec{x}\vec{y}}$ to represent $d_{x_1 y_1} \wedge \ldots \wedge d_{x_n y_n}$.

3 The language ccp

In this section we give the definition of ccp, following [14]. We refer to that paper for more details. We assume given a cylindric constraint system $\mathbf{C}$ on a set of variables *Var* with typical elements $x, y, \ldots$. The description of the language is parametric with respect to it, and so is the semantical construction we develop in this paper. In the following, the notation $\vec{\chi}$ indicates a sequence of the form $\chi_1, \ldots, \chi_n$. The processes are described by the following grammar

$$\begin{array}{rl} \mathit{Processes} & P ::= D.A \\ \mathit{Declarations} & D ::= \epsilon \mid p(\vec{x}) \text{ :- } A \mid D, D \\ \mathit{Agents} & A ::= \ \mathbf{Stop} \mid \mathbf{tell}(c) \mid \\ & \quad\quad \sum_{i=1}^{n} \mathbf{ask}(c_i) \rightarrow A_i \mid \\ & \quad\quad A \parallel A \mid \exists_x A \mid p(\vec{x}) \end{array}$$

The agent **Stop** represents successful termination. The basic actions are given by $\mathbf{ask}(c)$ and $\mathbf{tell}(c)$ constructs, where c is a *finite constraint*, i.e. an algebraic element of $\mathcal{C}$. These actions work on a common *store* which ranges over $\mathcal{C}$. $\mathbf{ask}(c)$ is a test on the current store and its execution does not modify the store. We say that $\mathbf{ask}(c)$ is a *guard* and that is *enabled* in d iff $c \leq d$. If d is the current store, then the execution of $\mathbf{tell}(c)$ sets the store to

[1] The entailment relation $\vdash$, which is commonly used in the literature, is the reverse of $\leq$. Formally: $\forall c, d \in C.\ c \vdash d \Leftrightarrow d \leq c$.

$c \wedge d$. The *guarded choice* agent $\sum_{i=1}^{n} g_i \rightarrow A_i$ selects nondeterministically one g_i which is enabled, and then behaves like A_i. If no guards are enabled, then it *suspends*, waiting for other (parallel) agents to add information to the store. Parallel composition is represented by $\|$. The situation in which all components of a system of parallel agents suspend is called *global suspension* or *deadlock*. The agent $\exists_x A$ behaves like A, with x considered *local* to A. Finally, the agent $p(\vec{x})$ is a procedure call, where p is the name of the procedure and $\vec{x}$ is the list of the actual parameters. The meaning of $p(\vec{x})$ is given by a procedure declaration of the form $p(\vec{y})$:- A, where $\vec{y}$ is the list of the formal parameters. We assume that in a process $D.A$ there is one and only one procedure declaration for every procedure name occurring in A; this is not a restriction with respect to concurrent logic languages, as the presence of alternative clauses can be simulated by the choice construct. In the following, we omit the declaration part when it is empty (ϵ).

3.1 The operational model and the observables $\mathcal{O}$

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 store. Table 1 describes the rules of T with respect to a given set of declarations D.

The guarded choice operator models global nondeterminism (**R2**), in the sense that it depends on the current store whether or not a guard is enabled, and the current store is subject to modifications by the external environment (**R1**). **R3** describes parallelism as interleaving. To describe locality (**R4**) the syntax has been extended by an agent $\exists_x^d A$ in which x is local to A and d is the store that has been produced locally on x. Initially the local store is empty, i.e. $\exists_x A = \exists_x^{true} A$. The execution of a procedure call is modeled by **R5**. $\Delta_{\vec{y}}^{\vec{x}}$ stands for $\exists_{\vec{\alpha}}^{d_{\vec{x}\vec{\alpha}}} \exists_{\vec{y}}^{d_{\vec{\alpha}\vec{y}}}$ and it is used to establish the link between the formal parameters $\vec{y}$ and the actual parameters $\vec{x}$. The variables $\vec{\alpha}$ are introduced in order to avoid problems related to names clash between $\vec{x}$ and $\vec{y}$. They are assumed to occur neither in the procedure declaration nor in the procedure call.

We describe now what we intend to *observe* about a process. Intuitively, for every possible initial store (input) we want to collect the results (outputs) of all possible computations: in the finite case the final store and the termination mode (success, failure or deadlock); in the infinite case the limit of the intermediate stores. Note that we do not have explicit termination modes in the transition system since we can extract this information from the final configuration. In fact, the computation is successful iff the final configuration is of the form $\langle \mathbf{Stop} \parallel \cdots \parallel \mathbf{Stop}, c\rangle$ with $c \neq false$, it fails iff the final store is $false$ and it deadlocks otherwise. Actually, in ccp successful termination can be detected and represented in the final store (see [16], the short-circuit algorithm). In conclusion, for finite computations we can restrict ourselves to observe the final store without loss of generality. Since **ask** does not modify the store, and **tell** increases it, the evolution of the store during the computation is monotonic. Hence we can restrict to consider as possible input-output pairs the set $P = \{\langle c, d\rangle \mid c, d \in \mathcal{C} \text{ and } c \leq d\}$. In the following we assume the set of declarations to be fixed. Given a set X, $\mathcal{P}(X)$ denotes the set of all the subsets of X.

Definition 3.1 The mapping $\mathcal{O} : Agents \rightarrow \mathcal{P}(P)$ which gives the observables of an agent, is defined by $\mathcal{O}(A) = \mathcal{O}_{\mathrm{Fin}}(A) \cup \mathcal{O}_{\mathrm{Inf}}(A)$, with

$$\mathcal{O}_{\mathrm{Fin}}(A) = \{\langle c, d\rangle \mid \text{ there exists } B \text{ s.t. } \langle A, c\rangle \longrightarrow^* \langle B, d\rangle \not\longrightarrow \}$$

R1	$\langle \mathbf{tell}(c), d\rangle \longrightarrow \langle \mathbf{Stop}, c \wedge d\rangle$	
R2	$\langle \sum_{i=1}^{n} g_i \rightarrow A_i, d\rangle \longrightarrow \langle A_j, d\rangle$	$j \in [1, n]$ *and* $g_j = \mathbf{ask}(c)$ *and* $c \leq d$
R3	$\dfrac{\langle A, c\rangle \longrightarrow \langle A', c'\rangle}{\begin{array}{c}\langle A \parallel B, c\rangle \longrightarrow \langle A' \parallel B, c'\rangle \\ \langle B \parallel A, c\rangle \longrightarrow \langle B \parallel A', c'\rangle\end{array}}$	
R4	$\dfrac{\langle A, d \wedge \exists_x c\rangle \longrightarrow \langle B, d'\rangle}{\langle \exists_x^d A, c\rangle \longrightarrow \langle \exists_x^{d'} B, c \wedge \exists_x d'\rangle}$	
R5	$\langle p(\vec{x}), c\rangle \longrightarrow \langle \Delta_{\vec{y}}^{\vec{x}} A, c\rangle$	$p(\vec{y}) : -A$ is the declaration for $p(\vec{x})$ in D

Table 1: The transition system T.

$\mathcal{O}_{\text{Inf}}(A) =$
$\{ \ \langle c, lub_{n\in\omega}(c_n)\rangle \mid c = c_0$ and
there exist $B_1, \ldots, B_n, \ldots$ s.t.
$\langle A, c_0\rangle \longrightarrow \langle B_1, c_1\rangle \longrightarrow \ldots \longrightarrow \langle B_n, c_n\rangle \longrightarrow \ldots\}$

where $\not\longrightarrow$ denotes the absence of outgoing transitions and $\longrightarrow^*$ denotes the reflexive and transitive closure of $\longrightarrow$.

4 An approximating Denotational Semantics

In this section we discuss a compositional semantics which approximates the observables and is based on input-output pairs, hence it is simple enough to provide a suitable basis for efficient compositional analysis. The semantics will be a proper approximation, because in general modeling exactly the observables of a concurrent language (compositionally) requires structures more complicate than input-output pairs. It is easy to show that the semantics $\mathcal{O}$ previously described is not compositional. The following is a counter-example (we use '+' as a syntactic abbreviation for Σ).

Example 4.1 Let $a, b, c \in \mathcal{C}$ be constraints ordered by the relation $a \leq b \leq c$ and let us consider the agents A_1 and A_2

$$\begin{array}{rl} A_1 = & (\mathbf{ask}(true) \rightarrow \mathbf{tell}(a)) \\ & + \\ & (\mathbf{ask}(b) \rightarrow \mathbf{tell}(c)) \end{array}$$

$$\begin{array}{rl} A_2 = & (\mathbf{ask}(true) \rightarrow \mathbf{tell}(a)) \\ & + \\ & (\mathbf{ask}(b) \rightarrow \mathbf{tell}(c)) \\ & + \\ & (\mathbf{ask}(true) \rightarrow (\mathbf{tell}(a) \parallel \mathbf{ask}(b) \rightarrow \mathbf{tell}(c))) \end{array}$$

It is easy to check that $\mathcal{O}(A_1) = \mathcal{O}(A_2)$. However, given the agent

$$B = \mathbf{ask}(a) \rightarrow \mathbf{tell}(b)$$

we have $\mathcal{O}(A_1 \parallel B) \neq \mathcal{O}(A_2 \parallel B)$ since $\langle true, c\rangle \in \mathcal{O}(A_2 \parallel B) \setminus \mathcal{O}(A_1 \parallel B)$.

The problem here is that, when considering the abstract input-output behaviour, the (agent corresponding to the) third branch of the process A_2 is equivalent to the union of the first two branches. Indeed, the control structure disappears and the third branch is represented by the two pairs $\langle true, a\rangle$ and $\langle b, c\rangle$.

In the following we present two denotational semantics $\mathcal{D}$ and $\mathcal{D}_{\rm Fin}$ which approximate $\mathcal{O}$ and $\mathcal{O}_{\rm Fin}$ respectively. We introduce $\mathcal{D}_{\rm Fin}$ because it is more accurate than $\mathcal{D}$, hence it is preferable when one is not interested in the analysis of infinite computations.

We require both semantics to satisfy the equations of Table 2, which reflect the transition system in Table 1. In the equation for $||$, the symbol $\circ$ represents the composition of relations, which is defined as $R_1 \circ R_2 = \{\langle r_1, r_2\rangle \mid$ there exists r s.t. $\langle r_1, r\rangle \in R_1$ and $\langle r, r_2\rangle \in R_2\}$. The transitive closure is extended to the infinite case, i.e. if the pairs $\langle r_0, r_1\rangle, \langle r_1, r_2\rangle, \ldots, \langle r_n, r_{n+1}\rangle, \ldots$ are in R, then the pair $\langle r_0, lub_{n\in\omega} r_n\rangle$ is in the transitive closure of R.

In order to describe infinite computations, we follow the TCSP approach. In TCSP a process which diverges is regarded as a source of nondeterminism. The extreme case is represented by a process which loops without performing any visible action. The denotation of such a process will be the maximal set of pairs $P = \{\langle c, d\rangle \mid c, d \in \mathcal{C}$ and $c \leq d\}$, representing a totally unpredictable behavior. This leads to a *least fixpoint approach*, w.r.t. the ordering $\mathcal{D}_1 \leq_D \mathcal{D}_2$ iff, for all agents A, $\mathcal{D}_1[\![A]\!] \supseteq \mathcal{D}_2[\![A]\!]$.

Definition 4.2 The *denotational semantics* $\mathcal{D}$: *Agents* $\rightarrow \mathcal{P}(P)$ is the least (wrt $\leq_D$) function which satisfies the equations in Table 2.

Proposition 4.3 *For any agent A, $\mathcal{O}(A) \subseteq \mathcal{D}[\![A]\!]$.*

The crucial point in this proposition is the denotation of $||$. The proof that $\mathcal{O}(A \parallel B) \subseteq \mathcal{D}[\![A \parallel B]\!]$ is based on showing that every interleaving of A and B can be mimicked by an alternation of *maximal* transition sequences starting from A and from B. This involves also the proof of commutativity and associativity of the denotation of $||$.

In general, $\mathcal{O}$ is *properly* included in $\mathcal{D}$ for three reasons. One reason is the treatment of $||$ in $\mathcal{D}$, which allows the parallel components to restart their computation from the beginning, at each step. For instance, consider again the agents A_1, A_2 and B of Example 4.1. In the denotational semantics of $A_1 \parallel B$ it is possible that A_1 chooses the first branch and produce a; then waits for b, starts again from the choice point, chooses the second branch and produces c. Thus $A_1 \parallel B$ and $A_2 \parallel B$ are equated by the denotational model. The second reason is the treatment of Σ, which is modeled as local choice. Formally this is expressed by the fact that the suspension of one guard in the store c is sufficient to generate the pair $\langle c, c\rangle$. Representing global choice would require the information about the interleaving points. In a semantics based on pairs, this would imply to associate a pair to every transition step, which would cause a worse loss of accuracy. The third reason is inherent to the way infinite computations are modeled. Consider the definition

$$p :- p.$$

The denotation of p is the maximal set of input-output pairs P: $\mathcal{D}[\![p]\!] = P$. On the other hand, the operational semantics of p is $\mathcal{O}(p) = \{\langle d, d\rangle \mid d \in \mathcal{C}\}$.

This 'lack of precision' in modeling the infinite computations seems to be unavoidable in the input-output semantics. An exact characterization of infinite processes would require more complicate structures, such as metric spaces and sequences of constraints. The third problem does not occur if we restrict to finite computations. This can be done by considering the greatest fixpoint instead of the least one.

Definition 4.4 $\mathcal{D}_{\rm Fin}$: *Agents* $\longrightarrow \mathcal{P}(P)$ is the greatest function which satisfies the equations in

D1	$\mathcal{D}[\![\mathbf{Stop}]\!] = \{\langle c, c\rangle \mid c \in \mathcal{C}\}$
D2	$\mathcal{D}[\![\mathbf{tell}(c)]\!] = \{\langle d, d \wedge c\rangle \mid d \in \mathcal{C}\}$
D3	$\mathcal{D}[\![\sum_{i=1}^{n} \mathbf{ask}(c_i) \rightarrow A_i]\!] = \bigcup_{i=i}^{n}\{\langle c, d\rangle \mid c_i \leq c \text{ and } \langle c, d\rangle \in \mathcal{D}[\![A_i]\!]\} \cup \bigcup_{i=i}^{n}\{\langle c, c\rangle \mid c \in \mathcal{C} \text{ and } c_i \not\leq c\}$
D4	$\mathcal{D}[\![A \parallel B]\!] = \{\langle c, d\rangle \mid \langle c, d\rangle \text{ is in the transitive closure of } \mathcal{D}[\![A]\!] \circ \mathcal{D}[\![B]\!] \text{ and } \langle d, d\rangle \in \mathcal{D}[\![A]\!] \cap \mathcal{D}[\![B]\!]\}$
D5	$\mathcal{D}[\![\exists_x A]\!] = \{\langle c, d\rangle \mid \text{there exists } d' \in \mathcal{C} \text{ s.t. } \langle \exists_x c, d'\rangle \in \mathcal{D}[\![A]\!] \text{ and } d = c \wedge \exists_x d'\}$
D6	$\mathcal{D}[\![p(\vec{x})]\!] = \{\langle c, d\rangle \mid \langle c, d\rangle \in \mathcal{D}[\![\Delta^{\vec{x}}_{\vec{y}} A]\!], \text{ where } p(\vec{y}) :- A \text{ is the declaration for } p(\vec{x}) \text{ in } D\}$

Table 2: The denotational semantics

Table 2 (with the difference that here we can restrict to the *finite* transitive closure).

Note that for the previous definitions of p and q we have $\mathcal{D}_{\text{Fin}}[\![p]\!] = \mathcal{O}(p) = \emptyset$.

Proposition 4.5 *For any agent A, $\mathcal{O}_{\text{Fin}}(A) \subseteq \mathcal{D}_{\text{Fin}}[\![A]\!]$.*

5 Upward closed semantics

If we look at the parallel operator as conjunction, at the choice operator as disjunction, and at the guarded statement as implication, then we can regard a program as a logical theory. This is the so-called *declarative interpretation* of cc programming, and concurrent logic programming. From this point of view, it makes sense to define a notion of observables which can be interpreted as the set of "logical consequences" of the program. Formally, we can obtain this set by collecting all the logical consequences of the final results for a given initial constraint. This is equivalent to *close* $\mathcal{O}$ upward w.r.t. the second component, and we denote these observables by $\mathcal{O}^{\text{U}}$. Given a program D and an agent A, $\langle c, d\rangle \in \mathcal{O}^{\text{U}}(A)$ will be read as "if c and D and A then d".

It will turn out that $\mathcal{O}^{\text{U}}$ can be approximated more precisely than $\mathcal{O}$ (note however that $\mathcal{O}^{\text{U}}$ is less informative than $\mathcal{O}$). In fact, one of the reasons of difference between $\mathcal{D}$ and $\mathcal{O}$ is that $\mathcal{D}[\![A]\!]$ may contain a pair $\langle c, d\rangle$ which is not in $\mathcal{O}(A)$ because it is obtained by "restarting a parallel component of A from the beginning". In this case, however, $\mathcal{O}(A)$ will contain a pair $\langle c, d'\rangle$ with $d' \leq d$. Now, the point is that, in the upward closed semantics, this pair induces also the presence of $\langle c, d\rangle$. Therefore this difference disappears.

For a poset $\langle X, \leq\rangle$, we denote by $\mathcal{P}^{\text{U}}(X)$ the set of the upward-closed subsets of X, i.e. $Y \in \mathcal{P}^{\text{U}}(X)$

iff $Y \subseteq X$ and for each $x, y \in X$, if $y \in Y$ and $y \leq x$ then $x \in Y$.

Definition 5.1 The mapping $\mathcal{O}^{\mathrm{U}} : Agents \rightarrow (\mathcal{P}^{\mathrm{U}}(\mathcal{C}) \rightarrow \mathcal{P}^{\mathrm{U}}(\mathcal{C}))$ is defined by $\mathcal{O}^{\mathrm{U}}(A)C = \mathcal{O}^{\mathrm{U}}_{\mathrm{Fin}}(A)C \cup \mathcal{O}^{\mathrm{U}}_{\mathrm{Inf}}(A)C$, where

$$\mathcal{O}^{\mathrm{U}}_{\mathrm{Fin}}(A)C = \{\ d \mid \text{ there exist } c, d', B. \text{ s.t. } \langle A, c\rangle \longrightarrow^* \langle B, d'\rangle \not\longrightarrow \text{ and } d' \leq d\}$$

$$\mathcal{O}^{\mathrm{U}}_{\mathrm{Inf}}(A)C = \{\ d \mid \text{ there exist } c_0, B_1, c_1, \ldots, B_n, c_n, \ldots \text{ s.t. } \langle A, c_0\rangle \longrightarrow \langle B_1, c_1\rangle \longrightarrow \ldots \longrightarrow \langle B_n, c_n\rangle \longrightarrow \ldots \text{ and for each } n \in \omega, c_n \leq d\}$$

The denotational model corresponding to previous operational semantics is defined as follows.

Definition 5.2 $\mathcal{D}^{\mathrm{U}}, \mathcal{D}^{\mathrm{U}}_{\mathrm{Fin}} : Agents \rightarrow (\mathcal{P}^{\mathrm{U}}(\mathcal{C}) \rightarrow \mathcal{P}^{\mathrm{U}}(\mathcal{C}))$ are the least and the greatest functions which satisfy the equations in Table 3. The ordering is the standard one: $\mathcal{D}_1 \leq \mathcal{D}_2$ iff for each $A \in Agents$, for each $C \in \mathcal{P}^{\mathrm{U}}(\mathcal{C})$, $\mathcal{D}_2[\![A]\!]C \subseteq \mathcal{D}_1[\![A]\!]C$ holds.

The following theorem states the adequacy of the upward closed semantics for analysis.

Theorem 5.3 *For any agent A, $\mathcal{O}^{\mathrm{U}}(A) \subseteq \mathcal{D}^{\mathrm{U}}[\![A]\!]$ and $\mathcal{O}^{\mathrm{U}}_{\mathrm{Fin}}(A) \subseteq \mathcal{D}^{\mathrm{U}}_{\mathrm{Fin}}[\![A]\!]$.*

5.1 Denotations as closure operators

In this section we show that the upward closed denotational semantics of an agent is a *closure operator*[2]. This allows us to follow the approach of [14], where a process is represented by a set of constraints, namely the fixpoints of the associated closure operator. The advantage is that the semantic operators can be defined in a simple way; in particular, $\|$ is given by set intersection. In our case, the resulting construction looks very similar to the semantics of Angelic ccp as defined in [10].

[2] Given a poset $(X, \leq)$, a function $f : X \rightarrow X$ is a *closure operator* iff f is extensive ($\forall x \in X.\ x \leq f(x)$) monotonic ($\forall x, y \in X.\ x \leq y \Rightarrow f(x) \leq f(y)$) and idempotent ($\forall x \in X.\ f(f(x)) = f(x)$).

Proposition 5.4 *For every agent A, $\mathcal{D}^{\mathrm{U}}[\![A]\!]$ is a continuous closure operator on $\langle \mathcal{P}^{\mathrm{U}}(\mathcal{C}), \supseteq\rangle$.*

Since $\mathcal{D}^{\mathrm{U}}[\![A]\!]$ is a closure operator, we can redefine the equations of Definition 5.2 using fixpoints. In fact it is well-known that a closure operator on a complete lattice can be represented by the set of its fixpoints: for $F = \{x \mid f(x) = x\}$ we have $f(x) = glb(\check{x} \cap F) = min(\check{x} \cap F)$, where $\check{x} = \{y \in X \mid x \leq y\}$. Furthermore, we show that we can restrict to consider only the *singleton fixpoints*, namely objects of the form $\check{c}$. Such a reformulation is simpler, hence more convenient for abstract interpretation.

Let $\mathcal{P}^{\mathrm{UC}}(\mathcal{C})$ be the set of upward closed subsets of $\mathcal{C}$ which have a finite set of minimal elements, namely the Scott-compact elements of $\mathcal{P}^{\mathrm{U}}(\mathcal{C})$. The lattice $\langle \mathcal{P}^{\mathrm{UC}}(\mathcal{C}), \supseteq\rangle$ is a sub-lattice of $\langle \mathcal{P}^{\mathrm{U}}(\mathcal{C}), \supseteq\rangle$ with lub operator given by set intersection and glb operator given by set union. Moreover $\langle \mathcal{P}^{\mathrm{UC}}(\mathcal{C}), \supseteq\rangle$ is a CPO isomorphic to the Smyth power-domain over $\langle \mathcal{C}, \leq\rangle$. Note that $\langle \mathcal{P}^{\mathrm{UC}}(\mathcal{C}), \supseteq\rangle$ is not a *complete* sublattice because union and intersection of infinite compact sets might not be compact.

Proposition 5.5 *$\mathcal{P}^{\mathrm{UC}}(\mathcal{C})$ is closed wrt the denotations of agents, i.e. for any agent A, for each $C \in \mathcal{P}^{\mathrm{UC}}(\mathcal{C})$, $\mathcal{D}^{\mathrm{U}}[\![A]\!]C \in \mathcal{P}^{\mathrm{UC}}(\mathcal{C})$.*

As a consequence, non-compact sets are never introduced during the computation and we can reduce to the domain $\mathcal{P}^{\mathrm{UC}}(\mathcal{C})$, i.e. we can define $\mathcal{D}^{\mathrm{U}} : Agents \rightarrow (\mathcal{P}^{\mathrm{UC}}(\mathcal{C}) \rightarrow \mathcal{P}^{\mathrm{UC}}(\mathcal{C}))$.

Proposition 5.6 *For any agent A, $\mathcal{D}^{\mathrm{U}}[\![A]\!]$ is linear, i.e. $\forall C, C' \in \mathcal{P}^{\mathrm{UC}}(\mathcal{C})$, $\mathcal{D}^{\mathrm{U}}[\![A]\!](C \cup C') = \mathcal{D}^{\mathrm{U}}[\![A]\!]C \cup \mathcal{D}^{\mathrm{U}}[\![A]\!]C'$.*

As a consequence, for all agents A, $\mathcal{D}^{\mathrm{U}}[\![A]\!]$ can be represented by the subset of its fixpoints which are of the form $\check{c}$ with $c \in \mathcal{C}$.

DU1	$\mathcal{D}^{\mathrm{U}}[\![\mathbf{Stop}]\!]C = C$
DU2	$\mathcal{D}^{\mathrm{U}}[\![\mathbf{tell}(c)]\!]C = \{d \mid d \in C \text{ and } c \leq d\}$
DU3	$\mathcal{D}^{\mathrm{U}}[\![\sum_{i=1}^{n} \mathbf{ask}(c_i) \rightarrow A_i]\!]C = \bigcup_{i=1}^{n}\{d \mid \text{ there exists } c \in C \text{ s.t. } c_i \leq c \text{ and } d \in \mathcal{D}^{\mathrm{U}}[\![A_i]\!]\{c\}\} \cup \bigcup_{i=1}^{n}\{d \mid \text{ there exists } c \in C \text{ s.t. } c_i \not\leq c \text{ and } c \leq d\}$
DU4	$\mathcal{D}^{\mathrm{U}}[\![\exists_x A]\!]C = \{d \mid \text{ there exists } c \in C \text{ s.t. } d' \in \mathcal{D}^{\mathrm{U}}[\![A]\!]\{\exists_x c\} \text{ and } c \wedge \exists_x d' \leq d\}$
DU5	$\mathcal{D}^{\mathrm{U}}[\![A \parallel B]\!]C = lub_{n\in\omega}(\mathcal{D}^{\mathrm{U}}[\![A]\!] \circ \mathcal{D}^{\mathrm{U}}[\![B]\!])^n C$
DU6	$\mathcal{D}^{\mathrm{U}}[\![p(\vec{x})]\!]C = \{d \mid d \in \mathcal{D}^{\mathrm{U}}[\![\Delta^{\vec{x}}_{\vec{y}} A]\!]C \text{ where } p(\vec{y}) : -A \text{ is the declaration for } p(\vec{x}) \text{ in } D\}$

Table 3: The upward closed denotational semantics $\mathcal{D}^{\mathrm{U}}$.

Definition 5.7 $\mathcal{S}, \mathcal{S}_{\mathrm{Fin}} : Agents \longrightarrow \mathcal{P}(\mathcal{C})$ are the least and the greatest function which satisfies the equations in Table 4, respectively.

Proposition 5.8 *For each agents A and for each $C \in \mathcal{P}^{\mathrm{UC}}(\mathcal{C})$ we have*

$$\mathcal{D}^{\mathrm{U}}[\![A]\!]C = \{d \mid \text{ there exists } c \in C, \text{ there exists } d' \in \mathcal{S}[\![A]\!] \cap \check{c} \text{ s.t. } d' \leq d\}$$

6 Analyses

In this section we show how the semantics presented in the previous sections can be used for program analyses. Abstract interpretation [5] formalizes the idea of "approximate computation", where descriptions of data replace the data itself. The idea is that an analysis is a computation in which the program is evaluated using a *non-standard interpretation* of data and operators in the program. According to this view the semantics which we have presented are mimicked by the abstract semantic equations. Constraints are replaced by descriptions of constraints and the operators are replaced by operators which approximate the concrete ones.

Definition 6.1 A *description* $\langle \mathbf{A}, \propto, \mathbf{C}\rangle$ consists of an *abstract domain* $\mathbf{A}$, a *concrete domain* $\mathbf{C}$, and an *approximation relation* $\propto \subseteq \mathbf{A} \times \mathbf{C}$.

The approximation relation is lifted to functions and relations as follows:

- Let $\langle \mathbf{A}_1, \propto_1, \mathbf{C}_1\rangle$ and $\langle \mathbf{A}_2, \propto_2, \mathbf{C}_2\rangle$ be descriptions, $F : \mathcal{A}_1 \rightarrow \mathcal{A}_2$ and $F' : \mathcal{C}_1 \rightarrow \mathcal{C}_2$ be functions. Then $F \propto F'$ iff $\forall a \in \mathcal{A}_1.\ \forall c \in \mathcal{C}_1.\ a \propto_1 c \Rightarrow F(a) \propto_2 F'(c)$.
- Let $\langle \mathbf{A}_1, \propto_1, \mathbf{C}_1\rangle$ and $\langle \mathbf{A}_2, \propto_2, \mathbf{C}_2\rangle$ be descriptions, $R \subseteq \mathcal{A}_1 \times \mathcal{A}_2$ and $R' \subseteq \mathcal{C}_1 \times \mathcal{C}_2$ be relations. Then $R \propto R'$ iff $\forall a \in \mathcal{A}_1.\ \forall c \in \mathcal{C}_1.\ a \propto_1 c$ and $\langle c, c'\rangle \in R' \Rightarrow \exists \langle a, a'\rangle \in R$ and $a' \propto_2 c'$.

For cc languages, we are interested in descriptions of constraint systems. We give the following definition, which allows us to develop a compositional analysis based on $\mathcal{D}$.

S1	$\mathcal{S}[\![\mathbf{Stop}]\!] = \mathcal{C}$
S2	$\mathcal{S}[\![\mathbf{tell(c)}]\!] = \check{\mathbf{c}}$
S3	$\mathcal{S}[\![\sum_{i=1}^{n} \mathbf{ask}(c_i) \rightarrow A_i]\!] = \bigcup_{i=1}^{n} \mathcal{S}[\![A_i]\!] \cap \check{c}_i \; \cup \; \bigcup_{i=1}^{n} \{c \mid c \in \mathcal{C} \text{ and } c_i \not\leq c\}$
S4	$\mathcal{S}[\![A \parallel B]\!] = \mathcal{S}[\![A]\!] \cap \mathcal{S}[\![B]\!]$
S5	$\mathcal{S}[\![\exists_x A]\!] = \{c \mid c \in \mathcal{C} \text{ and there exists } d \in \mathcal{S}[\![A]\!] \text{ s.t. } \exists_x c = \exists_x d\}$
S6	$\mathcal{S}[\![p(\vec{x})]\!] = \mathcal{S}[\![\Delta^{\vec{x}}_{\vec{y}} A]\!]$, where $p(\vec{y}) : -A$ is the declaration of $p(\vec{x})$ in A

Table 4: The singleton semantics $\mathcal{S}$.

Definition 6.2 A constraint system description for a cylindric constraint system $\mathbf{C} = \langle \mathcal{C}, \leq, \wedge, \mathit{true}, \mathit{false}, \mathit{Var}, \exists \rangle$ is a tuple $\langle \mathcal{A}, \propto, \wedge^{\mathcal{A}}, \triangle^{\mathcal{A}}, \downarrow^{\mathcal{A}}, \uparrow^{\mathcal{A}}, \exists^{\mathcal{A}} \rangle$ such that

1. $\langle \mathcal{A}, \propto, \mathcal{C} \rangle$ is a description,
2. $\mathcal{A}$ is a complete lattice with ordering $\leq^{\mathcal{A}}$,
3. $\wedge^{\mathcal{A}} : \mathcal{A} \times \mathcal{A} \rightarrow \mathcal{A}$ is extensive in both the arguments and approximates $\wedge$,
4. $\triangle^{\mathcal{A}} : \mathcal{C} \rightarrow \mathcal{A} \rightarrow \mathcal{A}$ and for all constraints c, $\lambda a'.c \triangle^{\mathcal{A}} a'$ is extensive and approximates $\lambda c'.c \wedge c'$,
5. $\uparrow^{\mathcal{A}}$: $\mathcal{C} \rightarrow \mathcal{A} \rightarrow \mathcal{A}$ and for all constraints c, $\lambda a'.c \uparrow^{\mathcal{A}} a'$ is extensive and $c \uparrow^{\mathcal{A}} a'$ approximates all constraints c' such that $c \leq c'$ and $a' \propto c'$,
6. $\downarrow^{\mathcal{A}}$: $\mathcal{C} \rightarrow \mathcal{A} \rightarrow \mathcal{A}$ and for all constraints c, $\lambda a'.c \downarrow^{\mathcal{A}} a'$ is extensive and $c \downarrow^{\mathcal{A}} a'$ approximates all constraints c' such that $c \not\leq c'$ and $a' \propto c'$,
7. for each $x \in \mathit{Var}$, $\exists^{\mathcal{A}}_x$ approximates $\exists_x$.

In Table 5 we show the semantic equations which abstract the denotational semantics defined by Table 2. This definition also makes use of the *possible entailment* relation $\vdash^{\mathcal{A}}_{pos} \subseteq \mathcal{A} \times \mathcal{C}$ defined by

$$a \vdash^{\mathcal{A}}_{pos} c \Leftrightarrow \text{ there exists } c' \text{ s.t. } a \propto c' \text{ and } c \leq c',$$

and the *definite entailment* relation $\vdash^{\mathcal{A}}_{def} \subseteq \mathcal{A} \times \mathcal{C}$ defined by

$$a \vdash^{\mathcal{A}}_{def} c \Leftrightarrow \text{ for each } c', \text{ if } a \propto c' \text{ then } c \leq c'.$$

Analogously to the concrete case, abstract denotations of programs are sets of input-output pairs and we can obtain from the equations two semantics corresponding to the greatest and the least fixpoint approach.

Definition 6.3 Let $P^{\mathcal{A}} = \{\langle a, b \rangle \mid a, b \in \mathcal{A} \text{ and } a \leq^{\mathcal{A}} b\}$. The semantics $\mathcal{D}^{\mathcal{A}}, \mathcal{D}^{\mathcal{A}}_{\mathrm{Fin}} : \mathit{Agents} \rightarrow \mathcal{P}(P^{\mathcal{A}})$ are the greatest and the least functions which satisfy the equations in Table 5, respectively.

A1	$\mathcal{D}^{\mathcal{A}}[\![\mathbf{Stop}]\!] = \{\langle a, a\rangle \mid a \in \mathcal{A}\}$	
A2	$\mathcal{D}^{\mathcal{A}}[\![\mathbf{tell}(c)]\!] = \{\langle a, c\triangle^{\mathcal{A}} a\rangle \mid a \in \mathcal{A}\}$	
A3	$\mathcal{D}^{\mathcal{A}}[\![\sum_{i=1}^{n} \mathbf{ask}(c_i) \rightarrow A_i]\!] =$	$\bigcup_{i=1}^{n}\{\langle a, b\rangle \mid a \vdash_{pos}^{\mathcal{A}} c_i \text{ and } \langle c_i \uparrow^{\mathcal{A}} a, b\rangle \in \mathcal{D}^{\mathcal{A}}[\![A_i]\!]\}$ $\cup$ $\bigcup_{i=1}^{n}\{\langle a, b\rangle \mid b = c_i \downarrow^{\mathcal{A}} a \text{ and not } a \vdash_{def}^{\mathcal{A}} c_i\}$
A4	$\mathcal{D}^{\mathcal{A}}[\![A \parallel B]\!] = \{\langle a, b\rangle \mid$	$\langle a, b\rangle$ is in the transitive closure of $\mathcal{D}^{\mathcal{A}}[\![A]\!] \circ \mathcal{D}^{\mathcal{A}}[\![B]\!]$ and $\langle b, b\rangle \in \mathcal{D}^{\mathcal{A}}[\![A]\!] \cap \mathcal{D}^{\mathcal{A}}[\![B]\!]$ $\}$
A5	$\mathcal{D}^{\mathcal{A}}[\![\exists_x A]\!] = \{\langle a, b\rangle \mid$ there exists $a' \in \mathcal{A}$ s.t. $\langle \exists_x^{\mathcal{A}} a, a'\rangle \in \mathcal{D}^{\mathcal{A}}[\![A]\!]$ and $b = a \wedge^{\mathcal{A}} \exists_x^{\mathcal{A}} a'\}$	
A6	$\mathcal{D}^{\mathcal{A}}[\![p(\vec{x})]\!] = \{\langle a, b\rangle \mid$	$\langle a, b\rangle \in \mathcal{D}[\![\Delta_{\vec{y}}^{\vec{x}} A]\!]$ where $p(\vec{y}) : -A$ is the declaration for $p(\vec{x})$ in D $\}$

Table 5: The abstract denotational semantics $\mathcal{D}^{\mathcal{A}}$.

Theorem 6.4 *For all agents A, $\mathcal{D}^{\mathcal{A}}[\![A]\!] \propto \mathcal{D}[\![A]\!]$ and $\mathcal{D}^{\mathcal{A}}_{\text{Fin}}[\![A]\!] \propto \mathcal{D}_{\text{Fin}}[\![A]\!]$.*

Similarly we can obtain the abstract semantics corresponding to the upward closed semantics. However because of space limitations we leave these out.

7 Conclusions and future work

We have proposed a framework for denotational semantics of concurrent constraint languages which can be used as a basis for compositional analysis.

The main advantage of our construction is the simplicity of the semantic domains: input-output relations or functions. This makes it suitable for cc program analysis, at the price of correctness in the *classical* sense.

From well known results we know that the loss of classical correctness is unavoidable, for such a semantic domain. Hence the only parameter to compare denotational input-output semantics would be the *accuracy* of the resulting analysis. In order to obtain an approximating semantics we have renounced to model faithfully global choice and continuation point. It would be interesting to investigate alternative approximations based on the same domain. One possibility is, for instance, to adopt the idea of [3], where all the alternatives in a choice are enabled as soon as one of the guards is enabled.

The framework we have proposed can be extended smoothly to ccp languages with atomic tell ([13]). The only problem is the adaptation of the finite semantics, in case we want to model the arrest of the computation when the store is inconsistent. Consider for instance two processes $p(x)$ and $q(x)$ which generate the constraint $x = a$ and $x = b$ respectively, and then loop. Their denotations would be empty, whereas their parallel composition would stop and generate a failure. One obvious solution is to introduce an "artificial" termination mode, which would lead to model partial computations. However, this extension complicates the analysis and makes it much less accurate; we are currently investigating alternative approaches.

Acknowledgements We would like to thank Enea Zaffanella for his helpful comments on a previous version of this paper.

References

[1] M. Alpuente, M. Falaschi, and N. Manzo. Analyses of Unsatisfiability for Incremental Logic Programming. In M. Bruynooghe and M. Wirsing, editors, *Proc. of PLILP'92, LNCS 631*, pages 443–457. Springer-Verlag, 1992.

[2] M. Codish, M. Falaschi, and K. Marriott. Suspension Analysis for Concurrent Logic Programs. In K. Furukawa, editor, *Proc. of the Eight Int. Conf. on Logic Programming*, pages 331– 345. The MIT Press, 1991.

[3] M. Codish, M. Falaschi, K. Marriott, and W. Winsborough. Efficient Analysis of Concurrent Constraint Logic Programs. In A. Lingas, editor, *Proc. of ICALP,LNCS* 1993.

[4] C. Codognet, P. Codognet, and M. Corsini. Abstract Interpretation for Concurrent Logic Languages. In S. Debray and M. Hermenegildo, editors, *Proc. North American Conf. on Logic Programming*, pages 215–232. The MIT Press, 1990.

[5] P. Cousot and R. Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Proc. of POPL*, pages 238–252, 1977.

[6] F.S. de Boer and C. Palamidessi. A Fully Abstract Model for Concurrent Constraint Programming. In S. Abramsky and T.S.E. Maibaum, editors, *Proc. of TAPSOFT/CAAP, LNCS 493*, pages 296–319. Springer-Verlag, 1991.

[7] S. Debray and D.S. Warren. Functional Computations in Logic Programs. *Proc. of TOPLAS*, pages 451–481, 1989.

[8] L. Henkin, J.D. Monk, and A. Tarski. *Cylindric Algebras (Part I).* North-Holland, 1971.

[9] K. Horiuchi. Less abstract semantics for abstract interpretation of FGHC programs. In *Proc. of the Int. Conf. on Fifth Generation Computer Systems*, pages 897–906, Tokyo, Japan, 1992.

[10] R. Jagadeesan, V.A. Saraswat, and V. Shanbhogue. Angelic non-determinism in concurrent constraint programming. Technical report, Xerox Park, 1991.

[11] K. Marriott and H. Søndergaard. Abstract Interpretation of Logic Programs: the Denotational Approach. In A. Bossi, editor, *Proc. of the Italian Conf. on Logic Programming*, pages 399–425, 1990.

[12] V.A. Saraswat. *Concurrent Constraint Programming Languages.* PhD thesis, Carnegie-Mellon University, January 1989. Published by The MIT Press, U.S.A., 1991.

[13] V.A. Saraswat and M. Rinard. Concurrent constraint programming. In *Proc. of POPL*, pages 232–245, 1990.

[14] V.A. Saraswat, M. Rinard, and P. Panangaden. Semantics foundations of Concurrent Constraint Programming. In *Proc. of POPL*, 1991.

[15] D. Scott. Domains for denotational semantics. In *Proc. of ICALP*, 1982.

[16] E.Y. Shapiro. The family of concurrent logic programming languages. *ACM Computing Surveys*, 21(3):412–510, 1989.