A Semantic Framework for the Analysis of Concurrent Constraint Programming

Rene Moreno
Dipartimento di Informatica, Università di Pisa
Corso Italia 40, 56125 Pisa, Italy.
Phone : +39 (+50) 887279
E-mail: moreno@di.unipi.it

Abstract

Compositional semantics allow to reason about programs in an incremental way, providing the basis for the development of modular data-flow analysis. The major drawback of these semantics is their complexity. This observation applies in particular for concurrent constraint programming (*ccp*). In this work we consider an operational semantics of *ccp* by using sequences of pairs of finite constraints to represent *ccp* computations which is equivalent to a denotational semantics, providing the basis for the development of an abstract interpretation framework for the analysis of *ccp* .

1 Introduction

Concurrent constraint programming (*ccp*) [9, 10, 11] is a 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* 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.

Like for most of concurrent languages, the presence of guarded nondeterminism causes the denotational semantics of *ccp* to be rather complicated (see [2] and [11]), and therefore programs are difficult to analyze and to reason about.

Compositionality is one of the most desirable properties of a formal semantics, since it provides a foundation of program verification and modular design. It depends upon the operators of the language, the behavior we want to describe (*observables*) and the degree of abstraction we want to reach.

Our goal is the definition of an operational semantics which is equivalent to a denotational one, providing the basis for the development of a semantic framework to reason about properties of *ccp* computations and their abstractions, following the ideas in [8].

This compositional characterization of the operational semantics of *ccp* is defined by using sequences of pairs of finite constraints, called *reactive sequences* ([1]) and is equivalent to a denotational semantics. Our model is correct w.r.t. the standard operational semantics in the sense that the input/output observables can be obtained in a simple way from the sequences representing computations.

The idea of defining a compositional operational semantics for *ccp* was investigated in [2]. In that work the compositional semantics is defined by using sequences of constraints labelled by assume/tell modes and the observation criteria adopted are the final results, together with termination modes. Our operational model associates to an agent a set of sequences that intuitively represents computations steps performed by an agent and there are not termination modes.

The paper is organized as follows. In the next section we give the syntax of the language, the standard operational model and the notion of input/output observables. In section 3 we define an operational semantics that is correct w.r.t. the input/output observables and compositional. In section 4 we present the denotational model and show that the two semantics are equivalent. Finally in section 5 we use these results to define a semantic framework for the analysis of *ccp* .

2 Concurrent constraint programming

In this section we recall the definition of concurrent constraint programming, its operational semantics and observational behavior. We refer to [11] for more details.

2.1 Cylindric constraint systems

Concurrent constraint programming is based on the notion of constraint system. Here we consider an abstract definition of such systems as lattices, following [11]

Definition 2.1 *A* cylindric constraint system *is a structure*

$$\mathbf{C} = \langle C, \leq, \sqcup, true, false, Var, \exists, d \rangle$$

such that

1. $\langle C, \leq, \sqcup, true, false \rangle$ *is a lattice, where* $\sqcup$ *is the lub operation (representing the logical and), and true, false are the least and the greatest elements of* C*, respectively*[1]*. The elements of* C *are called constraints.*

2. *Var is a denumerable set of variables , and for each* $x \in Var$ *the function* $\exists_x : C \to C$ *is a cylindrification operator [7], i.e. it satisfies the following properties:*

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

   (b) *if* $c \leq d$ *then* $\exists_x c \leq \exists_x d$,

   (c) $\exists_x(c \sqcup \exists_x d) = \exists_x c \sqcup \exists_x d$,

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

3. *For each* $x, y \in Var$, $d_{xy} \in C$ *is a* diagonal element *[7], i.e. it satisfies the following properties:*

   (a) $d_{xx} = true$,

   (b) *if* z *is different from* x, y *then* $d_{xy} = \exists_z(d_{xz} \sqcup d_{zy})$,

   (c) *if* x *is different from* y *then* $c \leq d_{xy} \sqcup \exists_x(c \sqcup d_{xy})$.

The cylindrification operators model a sort of existential quantification and are used for defining a hiding operator in the language. The diagonal elements are useful to model parameter passing. If $\mathbf{C}$ contains an equality theory, then the elements d_{xy} can be thought of as the formulas $x = y$.

2.2 The language

The syntax and semantics of *ccp* is parametric with respect to an underlying cylindric constraint system. Agents (Process) A, declarations D and programs P are described by the following syntax, where c, c_i represent constraints.

$$A ::= \mathbf{Stop} \mid \mathbf{tell}(c) \to A \mid \textstyle\sum_{i=1}^{n} \mathbf{ask}(c_i) \to A_i \mid A_1 \parallel A_2 \mid \exists_x A' \mid p(x)$$
$$D ::= p(x) :\!- A$$
$$P ::= \epsilon \mid D, P$$

We will denote by *Agents* the set of all *ccp* agents.

The agent **Stop** represents successful termination. The $\mathbf{ask}(c)$ and $\mathbf{tell}(c)$ operations work on a common *store* which ranges over C. If d is the current store, then the execution of $\mathbf{tell}(c)$ adds c to the store, that is, it sets the store to be $c \sqcup d$. The $\mathbf{ask}(c)$ operation is a *guard* and its execution does not modify the store: it just tests the current store. We say that $\mathbf{ask}(c)$ is *enabled* in d if $c \leq d$. The *guarded choice* agent $\sum_{i=1}^{n} \mathbf{ask}(c_i) \to A_i$ selects nondeterministically one $\mathbf{ask}(c_i)$ which is enabled in the current store, and then behaves like A_i. If no guards are enabled, then it *suspends*, waiting for other (parallel) agents to add information (constraints) to the store. Parallel composition of agents is represented by $\parallel$. We use $\exists_x$ also to indicate an *hiding operator* on agents. The intended meaning of $\exists_x A$ is that of an agent which behaves like A, but where x is considered *local* or *private* in A. Finally, the agent $p(x)$ is a procedure call, where p is the name of the procedure and x is the actual parameter. The meaning of $p(x)$ is given by a procedure declaration of the form $p(y) :\!- A$, where y is the formal parameter. In the following, we assume that for every procedure name there exists one and only one declaration in D.

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

Table 1: The transition system T_P

R1	$\langle \mathbf{tell}(c) \to A, d\rangle \longrightarrow \langle A, c \sqcup d\rangle$	
R2	$\langle \sum_{i=1}^{n} \mathbf{ask}(c_i) \to A_i, d\rangle \longrightarrow \langle A_j, d\rangle$	$j \in [1,n]$ *and* $c_j \leq d$
R3	$\dfrac{\langle A, c\rangle \longrightarrow \langle A', c'\rangle}{\begin{array}{l}\langle A \parallel B, c\rangle \longrightarrow \langle A' \parallel B, c'\rangle \\ \langle B \parallel A, c\rangle \longrightarrow \langle B \parallel A', c'\rangle\end{array}}$	
R4	$\dfrac{\langle A, d \sqcup \exists_x c\rangle \longrightarrow \langle B, d' \sqcup \exists_x c\rangle}{\langle \exists_x^d A, c\rangle \longrightarrow \langle \exists_x^{d'} B, c \sqcup \exists_x d'\rangle}$	
R5	$\langle p(y), c\rangle \longrightarrow \langle \Delta_y^x A, c\rangle$	$p(x)$: $-A \in P$

2.3 The operational model

The operational model of *ccp* , informally introduced above, is described in terms of a transition system $T_P = (Conf, \longrightarrow)$ which is specified with respect to a given program P. The configurations in *Conf* are pairs consisting of an agent, and a constraint representing the store. Table 1 describes the rules of T_P.

Rule *R1* describes the behavior of an agent of the form $\mathbf{tell}(c) \to A$: it adds c to the store and then behaves like A. Rule *R2* describes the fact that a choice agent selects one of the branches whose guard is enabled.This choice operator models global non-determinism: 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. Rule *R3* describes parallelism as an interleaving of the steps performed by single agents. Rule *R4* describes locality, where $\exists_x^d A$, represents an agent A where x is local and d is the information that has been produced locally on x. The local store is assumed to be initially empty, which amounts to regarding $\exists_x A$ as equivalent to $\exists_x^{true} A$. The execution of a procedure call is modeled by Rule *R5*. The symbol Δ_y^x stands for $\exists_z^{d_{**}} \exists_y^{d_{xy}}$, where z is assumed to occur neither in the declaration nor in the agent, and is used to establish the link between the formal and actual parameter.

Given an agent A and an initial store c, a *computation* from $\langle A, c\rangle$ is a sequence of transitions which starts from $\langle A, c\rangle$ and leads to a final configuration $\langle B, d\rangle$, final in the sense that no transitions can take place from $\langle B, d\rangle$. The standard notion of observables considers the input/output relation associated to an agent and can be defined as follows.

Definition 2.2 *The (input/output) observables of an agent A w.r.t. a program P are:*

$$\mathcal{O}_{io}[\![A]\!]_P = \{\langle c, d\rangle \mid \textit{there exists } B \textit{ s.t. } \langle A, c\rangle \longrightarrow^* \langle B, d\rangle \not\longrightarrow\}$$

where $\longrightarrow^*$ is the reflexive and transitive closure of $\longrightarrow$.

3 A compositional semantics

The operational semantics which associates to an agent A its observables $\mathcal{O}_{io}[\![A]\!]_P$ is not compositional. A compositional characterization can be obtained by using sequences of pairs of finite constraints, called *reactive sequences* [1], to represent *ccp* computations.

A *reactive sequence* has the form $\langle c_1, d_1\rangle \ldots \langle c_n, d_n\rangle\langle d, d\rangle$ and represents a computation of a *ccp* agent. Intuitively, a pair $\langle c_i, d_i\rangle$ represents a computation step performed by the agent A which transforms the global store from c_i to d_i. The last pair indicates that the agent has reached a *resting point*, i.e. in store d the agent does not produce any further information. It is natural to assume that reactive sequences are monotonically increasing, since in *ccp* computations the store evolves monotonically. In the following we will assume that each reactive sequence $\langle c_1, d_1\rangle \ldots \langle c_{n-1}, d_{n-1}\rangle\langle c_n, d_n\rangle$ satisfies: For $i \in [1, n-1]$ and $j \in [2, n]$ $d_i \vdash c_i$ and $c_j \vdash d_{i-1}$.

For $s = \langle c_1, d_1\rangle \ldots \langle c_n, d_n\rangle$, we define $Store(s) = d_n$, representing the global store after the computation steps of s.

We will denote by $\mathcal{S}$ the set of all reactive sequences with typical elements $s, s_1 \ldots$. Given two sequences s_1 and $s_2 \in \mathcal{S}$ we denote by $s_1.s_2 \in \mathcal{S}$ the sequence obtained from the concatenation of s_1

Table 2: The transition system T'_P

$R1'$	$\langle \mathbf{tell}(c) \to A, s\rangle \longrightarrow \langle A, s.\langle d, d \sqcup c\rangle\rangle$	$s \leq d$
$R2'$	$\langle \sum_{i=1}^{n} \mathbf{ask}(c_i) \to A_i, s\rangle \longrightarrow \langle A_j, s.\langle d, d\rangle\rangle$	$j \in [1,n],\ c_j \leq s$ *and* $s \leq d$
$R3'$	$\dfrac{\langle A, s\rangle \longrightarrow \langle A', s'\rangle}{\begin{array}{l}\langle A \parallel B, s\rangle \longrightarrow \langle A' \parallel B, s'\rangle \\ \langle B \parallel A, s\rangle \longrightarrow \langle B \parallel A', s'\rangle\end{array}}$	
$R4'$	$\dfrac{\langle A, \exists_x s\rangle \longrightarrow \langle B, \exists_x s.\langle c, e\rangle\rangle}{\langle \exists_x^d A, s\rangle \longrightarrow \langle \exists_x^d B, s.\langle \exists_x c, \exists_x e\rangle\rangle}$	$\exists_x c \sqcup \exists_x s \sqcup d = c$ $s.\langle c, e\rangle \in \mathcal{S}$, $s.\langle \exists_x c, \exists_x e\rangle \in \mathcal{S}$
$R5'$	$\langle p(y), s\rangle \longrightarrow \langle \Delta_y^x A, s.\langle d, d\rangle\rangle$	$p(x)\colon -A \in P$
$R6'$	$\langle A, s\rangle \longrightarrow \langle A, s.\langle d, d\rangle\rangle$	$s \leq d$

and s_2. This operation can be extended in the natural way to sets of sequences. We denote by $\mathbb{S}$ the complete lattice $(\mathcal{P}(\mathcal{S}), \subseteq)$.

To define the compositional semantics we use a transition system $T'_P = (Conf', \longrightarrow)$ specified with respect to a given program P. The configurations in $Conf'$ are pairs consisting of an agent, and a reactive sequence representing computations steps and the store. Table 2 describes the rules of T'_P.

The difference with the transition system T_P consists mainly in rule $R6'$, which models the interaction with the environment. The computation of an agent A is not immediately affected by actions made by the environment, only its future behavior will depend on them. An arbitrary constraint d can be then produced by the environment (adding the pair $\langle d, d\rangle$ to the actual sequence), without changing the state of A. This arbitrary steps are called *stuttering steps* [1].

The other rules correspond to the rules of T_P. Note that in rules $R2'$, $R5'$ the addition of a pair to the sequence s corresponds to the assumption that a computation step is performed. In the modified rules the operation $\exists_x$ applied to a sequence s denotes the sequence obtained from s by applying $\exists_x$ pointwise to the pairs of constraints of s. Furthermore, if $s = \langle c_1, d_1\rangle \ldots \langle c_n, d_n\rangle$ and d a constraint, then $s \leq d$ means $d_n \leq d$.

The correspondence between T_P and T'_P is expressed by the following lemma, similar to the one stated in [2], but by using another kind of sequences.

Lemma 3.1 *Rules $R1'$–$R5'$ of T'_P are correct w.r.t. Rules $R1$–$R5$ of T_P, in the sense that if*

$$\langle A, s\rangle \longrightarrow \langle A', s'\rangle$$

is an Ri' transition step in T'_P, then

$$\langle A, Store(s)\rangle \longrightarrow \langle A', Store(s')\rangle$$

is an Ri transition step in T_P.

We obtain now a compositional semantics $\mathcal{O}$ by collecting, for each agent, the sets of reactive sequences generated by T'_P. We need to consider all completed sequences and to represent the finite approximations of infinite computations.

Definition 3.1 *The semantics $\mathcal{O}$ for an agent A w.r.t. a program P in a initial store c is defined as:*

$$\begin{array}{rl} \mathcal{O}[\![A]\!]_P = & \{s' \mid \langle A, \langle c, c\rangle\rangle \longrightarrow^* \langle \mathbf{Stop}, s'\rangle\} \\ & \cup \\ & \{s' \mid \langle A, \langle c, c\rangle\rangle \longrightarrow^* \langle A', s'\rangle \\ & \qquad \textit{and from } \langle A', s'\rangle \textit{ the only applicable rule of } T'_P \textit{ is } R6'\} \end{array}$$

The correctness of the semantics $\mathcal{O}$ is expressed by the following.

Theorem 3.1 (Correctness) *For any agent A we have*

$$\mathcal{O}_{io}[\![A]\!]_P = \{\langle c, d\rangle \mid \text{there exists } \langle c, d_1\rangle\langle c_2, d_2\rangle \ldots \langle c_n, d\rangle\langle d, d\rangle \in \mathcal{O}[\![A]\!]_P \text{ such that } c_i = d_{i-1} \text{ for each } i \in [2, n]\}.$$

A sequence of the form $\langle c, d_1\rangle\langle d_1, d_2\rangle \ldots \langle d_{n-1}, d\rangle\langle d, d\rangle$ represents a computation for an agent where c is the input constraint and the contributions of the environment have been already produced by the agent itself, i.e. the agent itself produces all needed constraints for its execution. The last pair $\langle d, d\rangle$ ensures that the computation has reached a resting point.

3.1 Compositionality of $\mathcal{O}$

The semantics $\mathcal{O}$ is compositional with respect to all the operators of the language $\rightarrow$, $\sum$, $\|$ and $\exists_x$. The semantics counterparts of these operators are $\rightsquigarrow$, $\widetilde{\sum}$, $\bar{\|}$ and $\widetilde{\exists_x}$ and are defined below.

[tell] : Prefixing the action **tell**(c) to an agent A corresponds to concatenate the pair $\langle d, d \sqcup c\rangle$ with sequences representing computations of A. For $S \subseteq \mathcal{S}$ and a constraint c we define

$$c \rightsquigarrow_t S = \{\langle d, d \sqcup c\rangle.s \in \mathcal{S} \mid s \in S\}$$

[ask] : The computation of **ask**$(c) \rightarrow A$ corresponds to take all sequences $s' = \langle d_1, d_1\rangle \ldots \langle d_m, d_m\rangle$ for which $d_j \not\vdash c$ for each $j \in [1, m]$, representing a "waiting period" for a constraint stronger than c. During this period only the environment is active by producing the constraints d_j by adding pairs of the form $\langle d_j, d_j\rangle$. When the store is strong enough to entail c we concatenate s' with the sequences s representing computations of A.

We define for a constraint c

$$\mathcal{S}^c_{ss} = \{s' \in \mathcal{S} \mid s' = \langle d_1, d_1\rangle \ldots \langle d_m, d_m\rangle \; d_j \not\vdash c \text{ for each } j \in [1, m-1], \; d_m \vdash c\}$$

and

$$\mathcal{S}^c_{dd} = \{s' \in \mathcal{S} \mid s' = \langle d_1, d_1\rangle \ldots \langle d_m, d_m\rangle \; d_j \not\vdash c \text{ for each } j \in [1, m]\}$$

Then for $S \subseteq \mathcal{S}$ we define

$$c \rightsquigarrow_a S = \{\mathcal{S}^c_{ss}.s \cup \mathcal{S}^c_{dd} \mid s \in S, \mathcal{S}^c_{ss}.s \in \mathcal{S}\}$$

[Choice] : Apart from the case of deadlock , an alternative choice can always be selected, therefore the successful computations of the choice operator is given by set union. On the other side, sequences representing deadlock are present in the result if and only if they are present in all sets. Formally,

$$\widetilde{\sum_{i=1}^n} c_i \rightsquigarrow S_i = \bigcup_{i=1}^n \mathcal{S}^{c_i}_{ss}.S_i \cup \bigcap_{i=1}^n \mathcal{S}^{c_i}_{dd} \text{ where } \mathcal{S}^{c_i}_{ss}.S_i \in \mathcal{S}$$

[Parallel composition] : The partial operator $\bar{\|}$ presented in [1] allows to combine reactive sequences that agree at each point with respect to the contribution of the environment and that have the same length. In all other cases it is assumed to be undefined. We have

$\langle c_1, d_1\rangle \ldots \langle c_n, d_n\rangle \bar{\|} \langle c_1, e_1\rangle \ldots \langle c_n, e_n\rangle = \langle c_1, d_1 \sqcup e_1\rangle \ldots \langle c_n, d_n \sqcup e_n\rangle$.

The extension of this operator to sets of sequences is made in the obvious way.

[Hiding] : The hiding operator applied to a set S of reactive sequences, denoting the computation of an agent, should first take the sequences $s' \in S$ that are x-connected [1], i.e. those in which no information on x is present in the input constraints which has not been already accumulated by the computation of the agent. Given a sequence $s' = \langle c_1, d_1\rangle \ldots \langle c_n, d_n\rangle$ we say that s' is x-connected if

- $\exists_x c_1 = c_1$
- $\exists_x c_i \sqcup d_{i-1} = c_i$ for each $i \in [2, n]$

The resulting sequences s are then constructed by assuming that its computation steps do not provide more information on x, i.e. by assuming that they are x-invariant [1]. Given a sequence s we say that s is x-invariant if for all computation steps $\langle c, d\rangle$ of s, we have $d = \exists_x d \sqcup c$.

Finally we have to consider the contribution of the environment up to the information on x and for this purpose we require that $\exists_x s = \exists_x s'$.

We define then

$$\widetilde{\exists_x}(S) = \{s \in \mathcal{S} \mid \begin{array}{l}\text{there exists } s' \in S \text{ such that} \\ \exists_x s = \exists_x s', s' \text{ is } x\text{-connected and } s \text{ is } x\text{-invariant}\}\end{array}$$

By a standard case analysis of the transition system T'_P we have the following theorem:

Theorem 3.2 (Compositionality of $\mathcal{O}$) *For any agents A, A_i and B we have*

- $\mathcal{O}[\![\mathbf{tell}(c) \to A]\!]_P = c \rightsquigarrow_t \mathcal{O}[\![A]\!]_P$
- $\mathcal{O}[\![\mathbf{ask}(c) \to A]\!]_P = c \rightsquigarrow_a \mathcal{O}[\![A]\!]_P$
- $\mathcal{O}[\![\sum_{i=1}^n \mathbf{ask}(c_i) \to A_i]\!]_P = \widetilde{\sum_{i=1}^n} c_i \rightsquigarrow_a \mathcal{O}[\![A_i]\!]_P$
- $\mathcal{O}[\![A \parallel B]\!]_P = \mathcal{O}[\![A]\!]_P \,\widetilde{\|}\, \mathcal{O}[\![B]\!]_P$
- $\mathcal{O}[\![\exists_x A]\!]_P = \widetilde{\exists}_x \mathcal{O}[\![A]\!]_P$
- $\mathcal{O}[\![p(y)]\!]_P = \widetilde{\Delta}_y^x \mathcal{O}[\![A]\!]_P$, *where* $p(x) :\!- A \in P$ *and* $\widetilde{\Delta}_y^x = \widetilde{\exists}_z^{d_{xz}} \widetilde{\exists}_y^{d_{zy}}$

We define then the operational semantics of a program P as

$$\mathcal{O}[\![P]\!] = \lambda A.\mathcal{O}[\![A]\!]_P \text{ for } A \in \mathit{Agents}$$

4 The denotational semantics

We use now the operators defined in the previous section to define a denotational semantics, similar to the semantics presented in [1].

Definition 4.1 *$\mathcal{F}_P : \mathit{Agents} \to \mathbb{S}$ is the least function, with respect to the ordering induced by $\subseteq$, which satisfies the equations in Table 3.*

Note that the agent **Stop** cannot perform any computation step, so the result of a computation for **Stop** with input constraint c is always c. Therefore, the denotation of **Stop** consists of all finite sequences of stuttering steps which contain only the information provided by the input constraints.

In order to prove that the least function satisfying the equations in Table 3 actually exists we use fixed point theory. An interpretation I is a function $I : \mathit{Agents} \to \mathbb{S}$. Let us denote by $\mathbb{I}$ the set of all these interpretations and by $\sqsubseteq$ the ordering induced on $\mathbb{I}$ by set inclusion , i.e. $I \sqsubseteq I'$ if and only if $\forall A \in \mathit{Agents}. I(A) \subseteq I'(A)$. This partial order formalizes the evolution of the computation process.

We consider a monotonic mapping $\mathcal{T}_P$ on interpretations, associated to the program P, and defined in such a way that its fixed points are the solutions of Equations $F1$–$F6$.

Definition 4.2 *The mapping $\mathcal{T}_P : \mathbb{I} \to \mathbb{I}$ is defined as follows:*

1. $\mathcal{T}_P(I)(\mathbf{Stop}) = \{\langle c_1, c_1\rangle\langle c_2, c_2\rangle \ldots \langle c_n, c_n\rangle \in \mathcal{S} \mid n \geq 1\}$
2. $\mathcal{T}_P(I)(\mathbf{tell}(c) \longrightarrow A) = c \rightsquigarrow_t \mathcal{T}_P(I)(A)$
3. $\mathcal{T}_P(I)(\sum_{i=1}^n \mathbf{ask}(c_i) \to A_i) = \widetilde{\sum_{i=1}^n} c_i \rightsquigarrow_a \mathcal{T}_P(I)(A_i)$
4. $\mathcal{T}_P(I)(A \parallel B) = \mathcal{T}_P(I)(A) \,\widetilde{\|}\, \mathcal{T}_P(I)(B)$
5. $\mathcal{T}_P(I)(\exists_x A) = \widetilde{\exists}_x \mathcal{T}_P(I)(A)$
6. $\mathcal{T}_P(I)(p(y)) = \widetilde{\Delta}_y^x I(A)$ *where* $p(x) :\!- A \in P$.

Table 3: The denotational semantics

$F1$	$\mathcal{F}_P[\![\mathbf{Stop}]\!] = \{\langle c_1, c_1\rangle\langle c_2, c_2\rangle \ldots \langle c_n, c_n\rangle \in \mathcal{S} \mid n \geq 1\}$
$F2$	$\mathcal{F}_P[\![\mathbf{tell}(c) \longrightarrow A]\!] = c \rightsquigarrow_t \mathcal{F}_P[\![A]\!]$
$F3$	$\mathcal{F}_P[\![\sum_{i=1}^n \mathbf{ask}(c_i) \rightarrow A_i]\!] = \widetilde{\sum_{i=1}^n} c_i \rightsquigarrow_a \mathcal{F}_P[\![A_i]\!]$
$F4$	$\mathcal{F}_P[\![A \parallel B]\!] = \mathcal{F}_P[\![A]\!] \widetilde{\parallel} \mathcal{F}_P[\![B]\!]$
$F5$	$\mathcal{F}_P[\![\exists_x A]\!] = \widetilde{\exists}_x \mathcal{F}_P[\![A]\!]$
$F6$	$\mathcal{F}_P[\![p(y)]\!] = \widetilde{\Delta_y^x} \mathcal{F}_P[\![A]\!]$ where $p(x)$:-$A \in P$

The following proposition shows that the solutions of Equations $F1$–$F6$ are the fixed points of $\mathcal{T}_P$.

Proposition 4.1 *An interpretation I is a solution of the Equations $F1$–$F6$ iff $\mathcal{T}_P(I) = I$.*

The powers of the operator $\mathcal{T}_P$ are defined as

1. $\mathcal{T}_P \Uparrow 0 = I_\perp$,
2. $\mathcal{T}_P \Uparrow (n+1) = \mathcal{T}_P(\mathcal{T}_P \Uparrow n)$,
3. $\mathcal{T}_P \Uparrow \omega = \bigcup_n \mathcal{T}_P \Uparrow n$

where $I_\perp$ is the least interpretation, namely the interpretation which maps each agent into the empty set.

Then we have the following

Proposition 4.2 *$(\mathbb{I}, \sqsubseteq)$ is a complete lattice and $\mathcal{T}_P$ is continuous.*

Thus, from standard results, we have that the least fixed point of $\mathcal{T}_P$ exists and it coincides with $\mathcal{T}_P \Uparrow \omega$. From Proposition 4.1 we then obtain that $\mathcal{F}$ is well-defined, and that:

Corollary 4.1 *For each agent A we have $\mathcal{F}_P[\![A]\!] = \mathcal{T}_P \Uparrow \omega(A) = lfp(\mathcal{T}_P)$.*

We define then the denotation of a program P as

$$\mathcal{F}[\![P]\!] = \lambda A.\mathcal{F}_P[\![A]\!] \text{ for } A \in \mathit{Agents}$$

Finally we have the following

Theorem 4.1 (Equivalence of $\mathcal{O}$ and $\mathcal{F}$) *For all agents A and program P*

$$\mathcal{O}[\![A]\!]_P = \mathcal{F}_P[\![A]\!] \text{ and } \mathcal{O}[\![P]\!] = \mathcal{F}[\![P]\!]$$

Proof.
Straightforward from Definition 3.1, Theorem 3.2 and Definition 4.1
□

5 Compositional Analysis

The semantics presented in the previous sections can be used as the basis for a compositional analysis of *ccp* programs. The idea is to get an abstract denotational semantics from the concrete one, following the techniques of abstract interpretation [3], [4]. An analysis is then a computation in which the program is evaluated using a non-standard interpretation of data and operators in the program. According to this, the semantics 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 approximates the concrete ones.

5.1 The abstract semantics

We show now how the semantics presented in the previous sections can be used for program analysis. We present first some definitions from [5] and [6], and define then an abstract semantics.

Definition 5.1 *A description* $\langle \mathbf{A}, \alpha, \mathbf{C} \rangle$ *consists of an abstract domain* $\mathbf{A} = \langle \mathcal{A}, \leq^{\mathcal{A}} \rangle$, *a concrete domain* $\mathbf{C} = \langle \mathcal{C}, \leq^{\mathcal{C}} \rangle$, *and a monotonic abstraction function* $\alpha : \mathcal{C} \to \mathcal{A}$.

Given $a \in \mathcal{A}$, $c \in \mathcal{C}$, *we say that* a *approximates* c, *written* $a \propto c$, *iff* $a \leq^{\mathcal{A}} \alpha(c)$. *The approximation relation is lifted to functions, relations and sets as follows:*

- *Let* $\langle \mathbf{A}_1, \alpha_1, \mathbf{C}_1 \rangle$ *and* $\langle \mathbf{A}_2, \alpha_2, \mathbf{C}_2 \rangle$ *be descriptions,* $F : \mathcal{A}_1 \to \mathcal{A}_2$ *and* $G : \mathcal{C}_1 \to \mathcal{C}_2$ *be functions. Then* $F \propto G$ *iff for each* $d \in \mathcal{A}_1$ *and for each* $e \in \mathcal{C}_1$, $d \propto_1 e$ *implies* $F(d) \propto_2 G(e)$.
- *Let* $\langle \mathbf{A}_1, \alpha_1, \mathbf{C}_1 \rangle$ *and* $\langle \mathbf{A}_2, \alpha_2, \mathbf{C}_2 \rangle$ *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* $(c, c') \in R$ *implies that there exists* $\langle a, a' \rangle \in R'$ *and* $a' \propto_2 c'$.
- *Let* $\langle \mathbf{A}, \alpha, \mathbf{C} \rangle$ *be a description and let* $X \in \mathcal{P}(\mathcal{A})$ *and* $Y \in \mathcal{P}(\mathcal{C})$. *Then* $X \propto Y$ *iff for each* $e \in Y$ *there exists* $d \in X$ *such that* $d \propto e$.

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

Definition 5.2 *Consider the cylindric constraint systems* $\mathbf{C}$ *and* $\mathbf{A}$ *with* $\mathbf{C} = \langle \mathcal{C}, \leq, \sqcup, true, false, Var, \exists, d \rangle$ *and* $\mathbf{A} = \langle \mathcal{A}, \leq^{\mathcal{A}}, \sqcup^{\mathcal{A}}, true^{\mathcal{A}}, false^{\mathcal{A}}, Var, \exists^{\mathcal{A}}, d^{\mathcal{A}} \rangle$. *A constraint system description* $\langle \mathbf{A}, \alpha, \mathbf{C} \rangle$ *is a description such that*

1. $\sqcup^{\mathcal{A}} \propto \sqcup$
2. *We have a function* $\Delta^{\mathcal{A}} : \mathcal{C} \longrightarrow \mathcal{A} \longrightarrow \mathcal{A}$ *such that for all constraints* $c', \lambda a'.c\Delta^{\mathcal{A}}a'$ *is extensive and approximates* $\lambda c'.c \sqcup c'$.
3. $\forall x \in Var.\ \exists_x^{\mathcal{A}} \propto \exists_x$.
4. $\forall c \in \mathcal{C}.\ \alpha(\exists_x(c)) = \exists_x^{\mathcal{A}} \alpha(c)$.
5. $\forall x, y \in Var.\ \alpha(d_{xy}) = d_{xy}^{\mathcal{A}}$.

To derive the abstract semantics we define now the abstract versions of the semantics operators from Section 3.1. For the constraint system $\mathbf{A}$ we denote by $\mathcal{S}_{\mathcal{A}}$ the set of *abstract* reactive sequences and by $\mathbb{A}$ the complete lattice $(\mathcal{P}(\mathcal{S}_{\mathcal{A}}), \subseteq)$.

We will make use of two relations defined in [6]. The *possible entailment* relation $\vdash_{pos}^{\mathcal{A}} \subseteq \mathcal{A} \times \mathcal{C}$ defined as

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

and the *definite entailment* relation $\vdash_{def}^{\mathcal{A}} \subseteq \mathcal{A} \times \mathcal{C}$ defined as

$$a \vdash_{def}^{\mathcal{A}} c \text{ iff for each } c' \text{ if } a \propto c' \text{ then } c \leq c'.$$

We define then for $S_{\mathcal{A}}$ and $S_{\mathcal{A}}^i \in \mathbb{A}$ and c a constraint

- $c \tilde{\rightarrow}_t\ S_{\mathcal{A}} = \{\langle a, a\Delta^{\mathcal{A}}c \rangle.s_{\mathcal{A}} \in \mathcal{S}_{\mathcal{A}} \mid s_{\mathcal{A}} \in S_{\mathcal{A}}\}$
- For $S_{\mathcal{A}} \subseteq \mathcal{S}_{\mathcal{A}}$

$$c \tilde{\rightarrow}_a\ S_{\mathcal{A}} = \{\mathcal{A}_{ss}^c.s_{\mathcal{A}} \cup \mathcal{A}_{dd}^c \mid s_{\mathcal{A}} \in S_{\mathcal{A}}\ ,\ \mathcal{A}_{ss}^c.s_{\mathcal{A}} \in \mathcal{S}_{\mathcal{A}}\}$$

where

$$\mathcal{A}_{ss}^c = \{s'_{\mathcal{A}} \in \mathcal{S}_{\mathcal{A}} \mid\ s'_{\mathcal{A}} = \langle a_1, a_1 \rangle \ldots \langle a_m, a_m \rangle \\ a_j \not\vdash_{def} c \text{ for each } j \in [1, m-1], \\ a_m \vdash_{pos} c\}$$

and

$$\mathcal{A}_{dd}^c = \{s'_{\mathcal{A}} \in \mathcal{S}_{\mathcal{A}} \mid\ s'_{\mathcal{A}} = \langle a_1, a_1 \rangle \ldots \langle a_m, a_m \rangle \\ a_j \not\vdash_{def} c \text{ for each } j \in [1, m]\}$$

Table 4: The abstract denotational semantics

$FA1$	$\mathcal{F}_P^{\mathcal{A}}[\![\mathbf{Stop}]\!] = \{\langle a_1, a_1\rangle\langle a_2, a_2\rangle \ldots \langle a_n, a_n\rangle \in \mathcal{S}_{\mathcal{A}} \mid n \geq 1\}$
$FA2$	$\mathcal{F}_P^{\mathcal{A}}[\![\mathbf{tell}(c) \longrightarrow A]\!] = c \hat{\rightsquigarrow}_t \mathcal{F}_P^{\mathcal{A}}[\![A]\!]$
$FA3$	$\mathcal{F}_P^{\mathcal{A}}[\![\sum_{i=1}^n \mathbf{ask}(c_i) \rightarrow A_i]\!] = \widehat{\sum_{i=1}^n} c_i \hat{\rightsquigarrow}_a \mathcal{F}_P^{\mathcal{A}}[\![A_i]\!]$
$FA4$	$\mathcal{F}_P^{\mathcal{A}}[\![A \parallel B]\!] = \mathcal{F}_P^{\mathcal{A}}[\![A]\!] \hat{\parallel} \mathcal{F}_P^{\mathcal{A}}[\![B]\!]$
$FA5$	$\mathcal{F}_P^{\mathcal{A}}[\![\exists_x A]\!] = \hat{\exists}_x \mathcal{F}_P^{\mathcal{A}}[\![A]\!]$
$FA6$	$\mathcal{F}_P^{\mathcal{A}}[\![p(y)]\!] = \widehat{\Delta_y^x} \mathcal{F}_P^{\mathcal{A}}[\![A]\!]$ where $p(x) :\!- A \in P$

- $\widehat{\sum_{i=1}^n} c_i \ \tilde{\rightsquigarrow}_a \ S_{\mathcal{A}}^i = \bigcup_{i=1}^n \mathcal{A}_{ss}^{c_i}.S_{\mathcal{A}}^i \cup \bigcap_{i=1}^n \mathcal{A}_{dd}^{c_i}$ where $\mathcal{A}_{ss}^{c_i}.S_{\mathcal{A}}^i \in \mathcal{S}_{\mathcal{A}}$
- $\langle a_1, b_1\rangle \ldots \langle a_n, b_n\rangle \hat{\parallel} \langle a_1, b_1'\rangle \ldots \langle a_n, b_n'\rangle = \langle a_1, b_1 \sqcup^{\mathcal{A}} b_1'\rangle \ldots \langle a_n, b_n \sqcup^{\mathcal{A}} b_n'\rangle$.
- $\widehat{\exists_x}(S_{\mathcal{A}}) = \{s_{\mathcal{A}} \in \mathcal{S}_{\mathcal{A}} \mid$ there exists $s'_{\mathcal{A}} \in S_{\mathcal{A}}$ such that $\exists_x^{\mathcal{A}} s_{\mathcal{A}} = \exists_x^{\mathcal{A}} s', s'_{\mathcal{A}}$ is x-connected and $s_{\mathcal{A}}$ is x-invariant$\}$

Definition 5.3 *$\mathcal{F}_P^{\mathcal{A}} : Agents \rightarrow \mathbb{A}$ is the least function, with respect to the ordering induced by $\subseteq$, which satisfies the equations in Table 4.*

The abstract semantics $\mathcal{F}_P^{\mathcal{A}}$ can be used to approximate the observables, as shown in the following definition.

Definition 5.4 $O_{io}^{\mathcal{A}}[\![A]\!]_P = \{\langle a, b\rangle \mid$ *there exists* $\langle a, b_1\rangle\langle a_2, b_2\rangle \ldots \langle a_n, b\rangle\langle b, b\rangle \in \mathcal{F}_P^{\mathcal{A}}[\![A]\!]$ *such that* $a_i = b_{i-1}$ *for each* $i \in [2, n]\}$.

Intuitively, for an agent A, $O_{io}^{\mathcal{A}}[\![A]\!]_P$ represent the *abstract observables* of A retrieved from $\mathcal{F}_P^{\mathcal{A}}[\![A]\!]$. Finally we have the following

Theorem 5.1 *For all agents A and programs P, $O_{io}^{\mathcal{A}}[\![A]\!]_P \propto O_{io}[\![A]\!]_P$.*

5.2 An Example of Compositional Groundness Analysis

In this section we illustrate the use of the abstract denotational semantics $\mathcal{F}^{\mathcal{A}}$ by a very simple example of groundness analysis for ccp programs over term equations.

Let $t_1, t_2 \ldots$ be terms on a signature and Var be a set of variables. Let $\mathcal{E}$ be the set of existentially quantified conjunctions of equations, i.e. the least set $\mathcal{E}$ such that

- for any pair of terms t, t', $t = t' \in \mathcal{E}$,
- if $e \in \mathcal{E}$ then $\exists x.e \in \mathcal{E}$,
- if $e, e' \in \mathcal{E}$ then $e \sqcup e' \in \mathcal{E}$.

We denote by **Eqn** the Herbrand (cylindric) constraint system whose elements are those in $\mathcal{E}$ modulo logical equivalence. The ordering in **Eqn** is defined by

$$[e] \leq [e'] \text{ iff } e' \models e.$$

The operations in **Eqn** are the obvious, i.e. $\sqcup$ is logical conjunction and $\exists_x$ is the existential quantifier. The diagonal element d_{xy} corresponds to the equation $x = y$

Definition 5.5 *An element $e \in \mathcal{E}$ is* solved *if e is of the form*

$$\exists \vec{y}. x_1 = t_1 \sqcup \cdots \sqcup x_n = t_n$$

where each x_i is a distinct variable not occurring in any of the terms t_i and each $y \in \vec{y}$ occurs in some t_j.

It is well known that any satisfiable $e \in \mathcal{E}$ can be transformed into an equivalent one $Sol(e)$ which is solved. If e is not satisfiable we define $Sol(e) = false$.

The idea of groundness analysis is to infer statically which variables in the initial state are bound to ground terms in all possible successful computations.

In our setting a description of an element $e \in \mathcal{E}$ will be a set of variables, meaning that any unifier of e binds these variables to ground terms. This description can be defined as

$$\alpha(e) = \begin{cases} \{x \mid x = t \in Sol(e) \text{ and } t \text{ is } ground\} & \text{if } Sol(e) \neq false \\ Var & \text{if } Sol(e) = false \end{cases}$$

The abstract constraint system $\mathbf{A}$ has domain $\mathcal{P}(Var)$ (i.e. $\mathcal{A} = \mathcal{P}(Var)$) and operations defined as follows. For any $X, Y \in \mathcal{P}(Var)$:

1. $X \leq^{\mathcal{A}} Y$ iff $X \subseteq Y$,
2. $X \sqcup^{\mathcal{A}} Y = X \cup Y$,
3. $\exists_x^{\mathcal{A}} X = X \setminus \{x\}$,

It is easy to verify that $\langle \mathbf{A}, \alpha, \mathbf{Eqn} \rangle$ is a constraint system description, where $\Delta^{\mathcal{A}}(e)(X) = \alpha(e) \cup X$, for $e \in \mathcal{E}$ and $X \in \mathcal{A}$.

Consider now the following program P, defining three procedures p,q and r.

$$\begin{aligned} p(x,y) \quad &:- \quad \mathbf{ask}(x = a) \rightarrow \mathbf{Stop} \\ &\quad\quad + \\ &\quad\quad \mathbf{ask}(y = b) \rightarrow \mathbf{Stop}. \end{aligned}$$

$$q(x,y) \quad :- \quad \mathbf{tell}(x = y) \rightarrow \mathbf{Stop}.$$

$$r(x,y) \quad :- \quad p(x,y) \parallel q(x,y).$$

We want to analyze the agent $r(x,y)$. By applying the bottom-up construction of the least solution of equations $FA1$–$FA6$ we have, [2] first by using the rules for **ask** and choice

$$\begin{aligned} \mathcal{F}_P^{\mathcal{A}}[\![p(x,y)]\!] \quad = \quad & \{\langle\{\{y\},\{y\}\rangle\langle\{x,y\},\{x,y\}\rangle\} \quad \cup \\ & \{\langle\{\{x\},\{x\}\rangle\} \quad \cup \\ & \{\langle\{\{x\},\{x\}\rangle\langle\{x,y\},\{x,y\}\rangle\} \quad \cup \\ & \{\langle\{\{y\},\{y\}\rangle\} \quad \cup \\ & \{\langle\{\{x,y\},\{x,y\}\rangle\} \end{aligned}$$

which means that the agent $p(x,y)$ will bind to a ground term the variable x or the variable y or both. Then, by using the rule for **tell** we have

$$\begin{aligned} \mathcal{F}_P^{\mathcal{A}}[\![q(x,y)]\!] \quad = \quad & \{\langle\{\{y\},\{x,y\}\rangle\langle\{x,y\},\{x,y\}\rangle\} \quad \cup \\ & \{\langle\{\{x\},\{x,y\}\rangle\langle\{x,y\},\{x,y\}\rangle\} \quad \cup \\ & \{\langle\{\{x,y\},\{x,y\}\rangle\} \end{aligned}$$

whichs means that the agent $q(x,y)$ binds the variable x to a ground term if and only if it binds the variable y to a ground term. Finally, by applying the rule of parallel composition we have

$$\mathcal{F}_P^{\mathcal{A}}[\![r(x,y)]\!] \quad = \quad \{\langle\{\{x,y\},\{x,y\}\rangle\}$$

whichs in turn means that the agent $r(x,y)$ binds both variables x and y to ground terms.

The above stated conclusions can be shown better by retrieving the abstract observables of each agent from its abstract semantics. By applying definition 5.4 we have

$$\mathcal{O}_{io}^{\mathcal{A}}[\![p(x,y)]\!]_P = \{\langle\{\{x\},\{x\}\rangle,\langle\{\{y\},\{y\}\rangle,\langle\{\{x,y\},\{x,y\}\rangle\}$$

$$\mathcal{O}_{io}^{\mathcal{A}}[\![q(x,y)]\!]_P = \{\langle\{\{x\},\{x,y\}\rangle,\langle\{\{y\},\{x,y\}\rangle\}$$

$$\mathcal{O}_{io}^{\mathcal{A}}[\![r(x,y)]\!]_P = \{\langle\{\{x,y\},\{x,y\}\rangle\}$$

[2] To simplify, we do not consider sequences that contains the same steps more than once, i.e. of the form $\langle c,c\rangle.\langle c,c\rangle.\langle c,c\rangle\ldots$

6 Conclusion and future work

We have presented an operational semantics for *ccp* by using reactive sequences, which is compositional and equivalent to a denotational one. We used these results to compositional analysis by applying ideas of abstract interpretation and show that our abstract semantics approximates the input/output behavior of an agent w.r.t. to a program.

The future work will be concentrated in defining a framework for the analysis of *ccp* where we can reason about properties such as the existence of abstract transition systems, i.e. a transition system on an abstract domain which defines an abstract operational semantics. Then we plan to define a theory according to which the semantics properties of *ccp* computations are inherited by the denotations which model abstractions of these computations.

References

[1] F.S. de Boer and M. Gabbrielli. Modeling Real-time in Concurrent Constraint Programming. In J. Lloyd editor, *Proc. Int'l Logic Programming Symposium*, ILPS'95, pages 528-542. The MIT Press, 1995.

[2] F.S. de Boer and C. Palamidessi. A Fully Abstract Model for Concurrent Constraint Programming. in: S.Abramsky and T.S.E. Maibaum, eds., *Proc. TAPSOFT/CAAP'91*, Lecture Notes in Computer Science, Vol. 493 (Springer, Berlin, 1991) 296–319.

[3] P. Cousot and R. Cousot, Abstract Interpretation and Applications to Logic Programs, *Journal of Logic Programming*, 13(2 and 3):103–179,1992.

[4] P. Cousot and R. Cousot. Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation, in: M. Bruynooghe and M. Wirsing, editors, *Proc. of PILP'92* Volume 631 of *Lecture Notes in Computer Science*, 269–295. Springer Verlag,1992.

[5] M. Falaschi, M. Gabbrielli, K. Marriott and C. Palamidessi. Confluence in Concurrent Constraint Programming. *Theoretical Computer Science* 183, 1997. To appear.

[6] M. Falaschi, M. Gabbrielli, K. Marriott and C. Palamidessi. Compositional Analysis for Concurrent Constraint Programming. in: *Proc. 8th IEEE Symp. on Logic In Computer Science* (IEEE Computer Society Press, Los Alamitos, CA, 1993) 210–221.

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

[8] M. C. Meo. A Framework for reasoning about the semantics of Logic Programs. Ph.D. Thesis: TD-6/96 Dipartimento di Informatica, Università di Pisa.

[9] V.A. Saraswat. *Concurrent Constraint Programming*, Logic Programming Series. The MIT Press, Cambridge, MA, 1993.

[10] V.A. Saraswat and M. Rinard. Concurrent constraint programming, in: *Proc. 17th Annual ACM Symp. on Principles of Programming Languages*. ACM Press, New York, 1990, 232–245.

[11] V.A. Saraswat, M. Rinard and P. Panangaden. Semantics foundations of Concurrent Constraint Programming, in: *Proc. 18th Annual ACM Symp. on Principles of Programming Languages*. ACM Press, New York, 1991, 333–353.