The *AND*-compositionality of CLP computed answer constraints.

Roberto Bagnara, **Marco Comini**,
Francesca Scozzari and **Enea Zaffanella**
Dipartimento di Informatica,
Università di Pisa,
Corso Italia 40, 56125 Pisa, Italy
{bagnara,comini,scozzari,zaffanel}@di.unipi.it

Abstract

We present a semantic characterization of CLP languages for the class of *quick-checking* systems. We define a semantics of computed answer constraints which is *AND*-compositional, does not loose the distinction between active and passive constraints in the answers and has both a top-down and a bottom-up characterization. We also show that an incrementality property we impose on the constraint solver is essential in order to define an *AND*-compositional *atom-based* semantics.

1 Introduction

The semantics of CLP languages presented in [6] is (at least in its intentions) very general. Real CLP systems, in fact, are often equipped with incomplete constraint solvers and delay mechanisms. For instance, the arithmetic solver employed in the CLP($\mathcal{R}$) system [7] deals only with linear constraints, whereas non-linear ones are delayed. The languages truly implemented by these "real systems" are not captured by standard semantics constructions such as the original one of [5] and its refinements [4]. These semantics, indeed, refer to the *ideal* domain of computation, thus pretending that the system is endowed with a complete solver.

The idea of [6] consists in parameterizing the semantics construction with respect to the constraint system *and* two functions modeling the inferential power of the system's constraint solver and the delay mechanisms employed. The *infer* function models the solver's ability of deriving new information from the *constraint store* (i.e., the set of *active constraints*) and from the delayed constraints (which are termed as *passive*). The *consistent* function models the solver's ability of detecting the unsatisfiability of the constraint store. Despite this nice idea, the semantics of [6] has several drawbacks:

- The semantics is defined only for *most-general* atomic goals. However, it is not clear how it can be used to recover the computed answers for general goals. Indeed, this is not possible in the setting of [6] since the constraint solver is not assumed to be *incremental*. This property (which we define precisely in Section 3) amounts to say, roughly speaking, that feeding the solver with a set of constraints *all-in-one-shot*, or giving them to it *one-at-a-time* does not change the solver's results (namely, the sets of active and passive constraints will be the same).

- The distinction between active and passive constraints is taken into account only in the transition system used to define the operational semantics. In the end they are merged together so that everything is lost. This way, the two CLP($\mathcal{R}$) programs {`p(X) :- X = 2`} and {`p(X) :- X*X = 4`} are assigned the same semantics. Indeed, they exhibit a quite different behavior: in response to the query `?- p(Y)` the first one yields the answer `Y = 2` whereas, for the second one, the system's answer is `Y*Y =4 Maybe`, indicating that the constraint is still passive. Moreover, the operational semantics of [6] makes use also of the logical-truth relation on the (idealized) constraint domain, which is undesirable.

- The bottom-up characterization of computed answers is given only for the *ideal* systems (i.e., those endowed with a complete constraint solver).

In this work we present a semantic characterization of CLP languages which overcomes the limitations of the one given in [6]. In particular, for the important class of *quick-checking* systems (i.e., those in which every resolution step is followed by a satisfiability check, or, in other words, most if not all the existing CLP systems), we define a semantics of computed answer constraints which:

- under the very reasonable hypothesis of incrementality of the constraint solver, is *AND*-compositional, that is, the answers to general goals can be reconstructed from the answers to most-general atomic goals;

- does not loose the distinction between active and passive constraints in the answers;

- has both a top-down and a bottom-up characterization.

We also show that the incrementality assumption on the constraint solver is so reasonable that it is impossible, without it, to define an *atom-based semantics* which is both *AND*-compositional and independent from the computation rule.

2 Preliminaries

In this section, following [6], we briefly review the basic syntax and semantics of constraint systems and CLP languages.

A *signature* is a set of function and predicate symbols with arities. A Σ-*formula* is built from variables, function and predicate symbols in Σ by using the classical first order logical connectives and quantifiers. In a closed formula each occurrence of a variable lies in the scope of a quantifier for that variable; $\exists(\phi)$ denotes the existential closure of all the free variables occurring in formula ϕ. A Σ-*structure* $\mathcal{D}$ is a set D together with an assignment of functions and relations on D to the symbols of Σ. We assume that Σ always contains the predicate symbol $=$ which is interpreted as the identity over $\mathcal{D}$. A Σ-*theory* is a collection of closed Σ-formulas and a *model* of a Σ-theory T is a Σ-structure $\mathcal{D}$ such that all the formulas of T evaluate to *True* under the interpretation $\mathcal{D}$. We write $T, \mathcal{D} \models \phi$ to denote that the formula ϕ is valid in all the models of T extending $\mathcal{D}$. The set of *constraints* $\mathcal{L}$ is a subset of the set of Σ-formulas closed under variable renaming, conjunction and existential quantification. A pair $(\mathcal{D}, \mathcal{L})$ is a *constraint domain* and will be often denoted by $\mathcal{D}$.

A CLP program is a collection of rules of the form $A \leftarrow \boldsymbol{B}$ where A is an atom and $\boldsymbol{B}$ is a (finite) multiset of atoms and constraints. A *goal* $\boldsymbol{G}$ is a (finite) multiset of constraints and atoms, representing their conjunction. atoms($\boldsymbol{G}$) denotes the multiset of all the atoms in $\boldsymbol{G}$ and constr($\boldsymbol{G}$) denotes the multiset of all the constraints in $\boldsymbol{G}$. Multiset union is denoted by $\uplus$, whereas $\{\!|e|\!\}$ denotes the singleton multiset containing e. As usual, without any loss of generality, we assume that rules and goals are in standard form, meaning that all the arguments in atoms are variables and each variable occurs in at most one atom.

The operational semantics is given by defining a transition system on states. A *state* is a triple $\langle \boldsymbol{G}, C, S\rangle$ where $\boldsymbol{G}$ is a goal and C,S are finite (conjunctive) multisets of constraints, called the *active* and the *passive* constraints respectively. There is one additional state denoted **fail**. The initial state corresponding a goal $\boldsymbol{G}$ is $\langle \boldsymbol{G}, \emptyset, \emptyset\rangle$. The transition system is parametric with respect to the functions *consistent* and *infer*. Given a set of constraints C, the function consistent(C) is a consistency check such that $\mathcal{D} \models \exists(C)$ implies consistent(C). The function *infer* applied to a pair of finite multisets of constraints (C, S) yields a new pair of finite multisets of constraints (C', S') such that $\mathcal{D} \models (C \wedge S) \leftrightarrow (C' \wedge S')$.

The transition rules are the following:

$$\langle \boldsymbol{G} \uplus \{\!|A|\!\}, C, S\rangle \xrightarrow{r} \langle \boldsymbol{G} \uplus \boldsymbol{B}, C, S \uplus \{\!|A = H|\!\}\rangle$$

where A is the selected atom[1], $H \leftarrow \boldsymbol{B}$ is a renamed apart rule of P[2], H and A have the same predicate symbol and $\{\!|A = H|\!\}$ is an abbreviation for the conjunction of equations between the corresponding arguments.

$$\langle \boldsymbol{G} \uplus \{\!|A|\!\}, C, S\rangle \xrightarrow{r} \mathbf{fail}$$

where A is the selected atom and for every rule $H \leftarrow \boldsymbol{B}$ of P, H and A have different predicate symbols.

$$\langle \boldsymbol{G} \uplus \{\!|c|\!\}, C, S\rangle \xrightarrow{c} \langle \boldsymbol{G}, C, S \uplus \{\!|c|\!\}\rangle$$

[1] We assume as given a computation rule which selects a transition rule and an element of $\boldsymbol{G}$, if necessary.

[2] The program P is fixed in $\longrightarrow$.

if c is the selected constraint.

$$\langle \boldsymbol{G}, C, S\rangle \xrightarrow{i} \langle \boldsymbol{G}, C', S'\rangle \qquad \text{if } (C', S') = \text{infer}\,(C, S)$$
$$\langle \boldsymbol{G}, C, S\rangle \xrightarrow{s} \langle \boldsymbol{G}, C, S\rangle \qquad \text{if consistent}(C)$$
$$\langle \boldsymbol{G}, C, S\rangle \xrightarrow{s} \mathbf{fail} \qquad \text{if } \neg\,\text{consistent}(C)$$

A CLP system is thus defined by giving the constraint domain, the computation rule and definitions for *consistent* and *infer*.

Given states $s_0, \ldots, s_n$ and transitions $s_i \xrightarrow{x_i} s_{i+1}$ we will often write $s_0 \xrightarrow{x_0 \cdots x_{n-1}} s_n$ to denote the derivation $s_0 \xrightarrow{x_0} \ldots \xrightarrow{x_{n-1}} s_n$. Similarly, we write $s \xrightarrow{x|y} s'$ if $s \xrightarrow{x} s'$ or $s \xrightarrow{y} s'$ holds (according to the computation rule).

A CLP system is called *quick-checking* if its operational semantics can be described in terms of $\xrightarrow{ris}$ and $\xrightarrow{cis}$ (composite) rules only. A *successful* derivation starting from initial state $\langle \boldsymbol{G}, \emptyset, \emptyset\rangle$ is a finite derivation having $\langle \emptyset, C, S\rangle$ as the final state; (C, S) is an *answer constraint* for the initial goal $\boldsymbol{G}$.

As usual, the syntactic equivalence modulo variance is denoted by $\equiv$. For the sake of simplicity given the syntactic object E, E', F, we write $E \approx_F E'$ for $(F, E) \equiv (F, E')$, namely to denote that E can be obtained from E' by possibly renaming all the variables *but* those contained in F.

3 From *AND*-compositionality to incrementality

In this paper we are interested in computed answer constraints of quick-checking transition systems. Formally the set of *computed answer constraints* with respect to program P and goal $\boldsymbol{G}$ is defined as

$$\text{Cac}(P, \boldsymbol{G}) = \{\, \langle \boldsymbol{G}, C, S\rangle \mid \langle \boldsymbol{G}, \emptyset, \emptyset\rangle \xrightarrow{(r|c)is}{}^{*} \langle \emptyset, C, S\rangle \,\}$$

As usual computed answer constraints *must* not depend upon variable names, i.e., computed answer constraints of variant goals must be variant themselves. In the logic programming case this property is a consequence of the renaming apart condition used in *SLD*-resolution. In the CLP case to ensure this property the *infer* and *consistent* operations must compute their results independently upon variable names too, i.e.,

$$(C, S) \equiv (C', S') \Longrightarrow (C, S, \text{infer}\,(C, S)) \equiv (C', S', \text{infer}\,(C', S')) \tag{3.1}$$
$$C \equiv C' \Longrightarrow \text{consistent}(C) \leftrightarrow \text{consistent}(C') \tag{3.2}$$

An *atom-based semantics* $\mathcal{S}$ is a set of triples of the form $\langle A, C, S\rangle$ where A is an atom and C, S are finite multisets of constraints. An atom-based semantics differs from the Cac operational behavior defined above in that it gathers the computed answers constraints of atomic goals only. The lower complexity of atom-based semantics make them more suitable for the development of semantics based analysis tools. However, to be really useful, these semantics have to be provided with a mechanism to reconstruct the computed answer constraints of a general goal from the semantics of its components (atoms and constraints). The availability of such a mechanism is the well-known *AND*-compositionality property of the semantics. Hence, the top-down semantics can be defined using the *s*-semantics atom-based style [2].

Definition 3.1 *Let P be a fixed program, p range over the set of predicate symbols and $\boldsymbol{x}$ be a tuple of distinct variables. Then the* top-down semantics *of P with respect to the computation rule R is*

$$SS_P^R = \{ \langle p(\boldsymbol{x}),\, C,\, S\rangle_{/\equiv} \mid \langle p(\boldsymbol{x}),\, \emptyset,\, \emptyset\rangle \xrightarrow[R]{(r|c)is}{}^{*} \langle \emptyset,\, C,\, S\rangle \}$$

When it is clear from the context, we will omit the computation rule index R.

Definition 3.2 *An atom-based semantics $\mathcal{S}$ is AND-*compositional *w.r.t. computed answer constraints if*

$$\begin{array}{l} \langle \boldsymbol{G},\, C,\, S\rangle \in \mathrm{Cac}(P, \boldsymbol{G}) \Longleftrightarrow \\ \quad \forall G_i \in \mathrm{atoms}(\boldsymbol{G})\ \exists \langle G_i',\, C_i,\, S_i\rangle \textit{ renamed apart versions of elements in } \mathcal{S} \\ \quad \textit{and } \exists \rho \textit{ variables substitution s.t. } G_i = G_i'\rho \textit{ and} \\ \quad (C',\, S') = \mathrm{infer}\,(\emptyset,\, \biguplus_i (C_i \uplus S_i)\rho \uplus \mathrm{constr}(\boldsymbol{G})) \textit{ and } \mathrm{consistent}(C') \end{array}$$

where $(C,\, S) \approx_{\boldsymbol{G}} (C',\, S')$.

In the general case, both the Cac set and the atom-based semantics S depend upon the choice of computation rule. When defining the *AND*-compositional property we are faced to the following choice: either the *AND*-compositionality mechanism depends on the computation rule, or the same *AND*-compositionality mechanism has to be used for all computation rules. Due to its greater formal and computational simplicity, in this paper we will opt for the latter. Moreover this mechanism has to be defined by means of the tools available inside the constraint system, namely in terms of the *infer* and *consistent* functions. In this way we avoid using the $\models$ relation, which is only weakly related to the real computation power of the constraint system.

From a technical point of view, all the active and passive constraints coming from the atoms of the composite goal are collapsed into a multiset of passive constraints. The alternative choice of keeping distinct the active and passive parts would fail in that, in general, the union of several consistent active constraints can yield an inconsistency.

Definition 3.3 *The operations* infer *and* consistent *are* incremental *if*

$$\begin{array}{l} \mathrm{infer}\,(\emptyset,\, S) = (C_1,\, S_1),\ \mathrm{consistent}(C_1),\ \mathrm{infer}\,(C_1,\, S_1 \uplus \{\!|\, c\,|\!\}) = (C_2,\, S_2), \\ \mathrm{consistent}(C_2) \Longleftrightarrow \mathrm{infer}\,(\emptyset,\, S \uplus \{\!|\, c\,|\!\}) = (C_2',\, S_2'),\ \mathrm{consistent}(C_2') \end{array} \tag{3.3}$$

where $(C_2,\, S_2) \approx_{S \uplus \{\!| c |\!\}} (C_2',\, S_2')$ and

$$\mathrm{infer}\,(\emptyset,\, \emptyset) = (\emptyset,\, \emptyset),\ \mathrm{consistent}(\emptyset) \tag{3.4}$$

Moreover the transition system is incremental *if the operations* infer *and* consistent *are incremental.*

The following Lemma shows that, given an incremental transition system, every quick checking computation rule R_1 can be simulated by a corresponding computation rule R_2 which omits all the non final i and s transitions. We call such a computation rule *late checking.*

Lemma 3.4 *Consider an incremental transition system. Then* $\forall \boldsymbol{G},\, n \geq 0$

$$\langle \boldsymbol{G},\, \emptyset,\, \emptyset \rangle \xrightarrow{(r|c)is}{}^{n} \langle \emptyset,\, C^q,\, S^q \rangle \Longleftrightarrow \langle \boldsymbol{G},\, \emptyset,\, \emptyset \rangle \xrightarrow{r|c}{}^{n} \xrightarrow{is} \langle \emptyset,\, C^\ell,\, S^\ell \rangle$$

where $(C^q,\, S^q) \approx_{\boldsymbol{G}} (C^\ell,\, S^\ell)$.

Proof. We prove a stronger result for induction on $n \geq 0$, namely

$$\langle \boldsymbol{G},\, \emptyset,\, \emptyset \rangle \xrightarrow{(r|c)is}{}^{n} \langle \boldsymbol{G}^q,\, C^q,\, S^q \rangle \Longleftrightarrow \langle \boldsymbol{G},\, \emptyset,\, \emptyset \rangle \xrightarrow{r|c}{}^{n} \xrightarrow{is} \langle \boldsymbol{G}^{\boldsymbol{\ell}},\, C^\ell,\, S^\ell \rangle$$

where $(\boldsymbol{G} \uplus \boldsymbol{G}^q,\, C^q,\, S^q) \equiv (\boldsymbol{G} \uplus \boldsymbol{G}^\ell,\, C^\ell,\, S^\ell)$. The $n = 0$ base case by (3.4) is trivial, while for the inductive case we have $\langle \boldsymbol{G},\, \emptyset,\, \emptyset \rangle \xrightarrow{(r|c)is}{}^{n} \langle \boldsymbol{G}^q,\, C^q,\, S^q \rangle$ which is equivalent to

$$\langle \boldsymbol{G},\, \emptyset,\, \emptyset \rangle \xrightarrow{(r|c)is}{}^{n-1} \langle \boldsymbol{G}_1,\, C_1,\, S_1 \rangle \xrightarrow{(r|c)is} \langle \boldsymbol{G}^q,\, C^q,\, S^q \rangle.$$

By applying the inductive hypotheses this is equivalent to

$$\langle \boldsymbol{G},\, \emptyset,\, \emptyset \rangle \xrightarrow{r|c}{}^{n-1} \xrightarrow{is} \langle \boldsymbol{G}_2,\, C_2,\, S_2 \rangle \xrightarrow{(r|c)is} \langle \boldsymbol{G}^\ell,\, C_3,\, S_3 \rangle$$

where $(\boldsymbol{G} \uplus \boldsymbol{G}_1,\, C_1,\, S_1) \equiv (\boldsymbol{G} \uplus \boldsymbol{G}_2,\, C_2,\, S_2)$ and by (3.1) and (3.2) $(\boldsymbol{G} \uplus \boldsymbol{G}^q,\, C^q,\, S^q) \equiv (\boldsymbol{G} \uplus \boldsymbol{G}^\ell,\, C_3,\, S_3)$. This in turn is equivalent to

$$\begin{aligned} &\langle \boldsymbol{G},\, \emptyset,\, \emptyset \rangle \xrightarrow{r|c}{}^{n-1} \langle \boldsymbol{G}_2,\, \emptyset,\, S_4 \rangle \xrightarrow{is} \langle \boldsymbol{G}_2,\, C_2,\, S_2 \rangle \xrightarrow{r|c} \\ &\qquad \langle \boldsymbol{G}^\ell,\, C_2,\, S_2 \uplus \{\!|\, c \,|\!\} \rangle \xrightarrow{is} \langle \boldsymbol{G}^\ell,\, C_3,\, S_3 \rangle \end{aligned}$$

where $\mathrm{infer}\,(\emptyset,\, S_4) = (C_2,\, S_2), \mathrm{consistent}(C_2)$ and $\mathrm{infer}\,(C_2,\, S_2 \uplus \{\!|\, c \,|\!\}) = (C_3,\, S_3)$, $\mathrm{consistent}(C_3)$. The constraint $\{\!|\, c \,|\!\}$ is the one introduced by the r or c rule in the last derivation step.

Because of the incrementality of the transition system we obtain

$$\mathrm{infer}\,(\emptyset,\, S_4 \uplus \{\!|\, c \,|\!\}) = (C^\ell,\, S^\ell)$$

and $\mathrm{consistent}(C^\ell)$ where $(C_3,\, S_3) \approx_{S_4 \uplus \{\!|\, c \,|\!\}} (C^\ell,\, S^\ell)$. Because of the renaming apart property of *infer* we have that $(C_3,\, S_3) \approx_{\boldsymbol{G} \uplus \boldsymbol{G}^\ell} (C^\ell,\, S^\ell)$. Thus we can construct the following derivation

$$\langle \boldsymbol{G},\, \emptyset,\, \emptyset \rangle \xrightarrow{r|c}{}^{n-1} \langle \boldsymbol{G}_2,\, \emptyset,\, S_4 \rangle \xrightarrow{r|c} \langle \boldsymbol{G}^\ell,\, \emptyset,\, S_4 \uplus \{\!|\, c \,|\!\} \rangle \xrightarrow{is} \langle \boldsymbol{G}^\ell,\, C^\ell,\, S^\ell \rangle$$

By the transitivity of $\equiv$ the proof is so concluded. ∎

Next Lemma follows from the commutativity of multisets' union.

Lemma 3.5 *Let R_1 and R_2 be two late checking computation rules.*

$$\langle \boldsymbol{G},\, \emptyset,\, \emptyset\rangle \xrightarrow[R_1]{r|c}{}^{n} \xrightarrow{is} \langle \emptyset,\, C_1,\, S_1\rangle \Longleftrightarrow \langle \boldsymbol{G},\, \emptyset,\, \emptyset\rangle \xrightarrow[R_2]{r|c}{}^{n} \xrightarrow{is} \langle \emptyset,\, C_2,\, S_2\rangle$$

where $(C_1,\, S_1) \approx_{\boldsymbol{G}} (C_2,\, S_2)$.

An immediate consequence of the two previous lemmata is the independence of the semantics upon quick checking computation rules.

Theorem 3.6 *Consider an incremental transition system and let R_1 and R_2 be two quick checking computation rules. Then $SS_P^{R_1} = SS_P^{R_2}$.*

Lemma 3.7 *Consider an incremental transition system. Then $\forall \boldsymbol{G},\, S_0,\, n \geq 0$*

$$\begin{aligned}&\langle \boldsymbol{G},\, \emptyset,\, S_0\rangle \xrightarrow{r|c}{}^{n} \xrightarrow{is} \langle \emptyset,\, C,\, S\rangle \Longleftrightarrow\\ &\quad \forall G_i \in \mathrm{atoms}(\boldsymbol{G})\ \exists \langle G_i',\, C_i,\, S_i\rangle \textit{ renamed apart versions of}\\ &\quad \textit{elements in } SS_P \textit{ and } \exists \rho \textit{ variables substitution s.t. } G_i = G_i'\rho \textit{ and}\\ &\quad (C',\, S') = \mathrm{infer}\,(\emptyset,\, S_0 \uplus \biguplus_i (C_i \uplus S_i)\rho \uplus \mathrm{constr}(\boldsymbol{G})) \textit{ and}\\ &\quad \mathrm{consistent}(C')\end{aligned}$$

where $(C,\, S) \approx_{\boldsymbol{G} \uplus S_0} (C',\, S')$.

Proof. The proof is by induction on n. The $n = 0$ base case is trivial, while for $n > 0$ we have two sub-cases, depending on the rule applied first

***c* rule]** Let c be the selected constraint and $\boldsymbol{G} = \boldsymbol{G}_0 \uplus \{\!|\, c \,|\!\}$.

$$\langle \boldsymbol{G},\, \emptyset,\, S_0\rangle \xrightarrow{c} \langle \boldsymbol{G}_0,\, \emptyset,\, S_0 \uplus \{\!|\, c \,|\!\}\rangle \xrightarrow{r|c}{}^{n-1} \xrightarrow{is} \langle \emptyset,\, C,\, S\rangle$$

For IH we have $\forall G_i \in \mathrm{atoms}(\boldsymbol{G}_0)\ \exists \langle G_i',\, C_i,\, S_i\rangle$ renamed apart versions of elements in SS_P and $\exists \rho$ s.t. $G_i = G_i'\rho$ and

$$(C',\, S') = \mathrm{infer}\,(\emptyset,\, S_0 \uplus \{\!|\, c \,|\!\} \uplus \biguplus_i (C_i \uplus S_i)\rho \uplus \mathrm{constr}(\boldsymbol{G}_0))$$

and consistent(C'), where $(C,\, S) \approx_{\boldsymbol{G}_0 \uplus S_0 \uplus \{\!|\, c \,|\!\}} (C',\, S')$. The thesis follows immediately from $\mathrm{atoms}(\boldsymbol{G}_0) = \mathrm{atoms}(\boldsymbol{G})$ and $\mathrm{constr}(\boldsymbol{G}_0) \uplus \{\!|\, c \,|\!\} = \mathrm{constr}(\boldsymbol{G})$.

***r* rule]** Let A be the selected atom, $\boldsymbol{G} = \boldsymbol{G}_0 \uplus \{\!|\, A \,|\!\}$ and $H \leftarrow \boldsymbol{B}$ be the used (renamed apart) clause of P. Then

$$\langle \boldsymbol{G},\, \emptyset,\, S_0\rangle \xrightarrow{r} \langle \boldsymbol{G}_0 \uplus \boldsymbol{B},\, \emptyset,\, S_0 \uplus \{\!|\, A = H \,|\!\}\rangle \xrightarrow{r|c}{}^{n} \xrightarrow{is} \langle \emptyset,\, C,\, S\rangle$$

For IH we have $\forall G_i \in \mathrm{atoms}(\boldsymbol{G}_0)$, $\forall B_j \in \mathrm{atoms}(\boldsymbol{B})$ $\exists \langle G_i',\, C_i,\, S_i\rangle$, $\exists \langle B_j',\, C_j,\, S_j\rangle$ renamed apart versions of elements in SS_P and $\exists \rho$ s.t. $G_i = G_i'\rho$, $B_j = B_j'\rho$ and

$$\begin{aligned}(C_1,\, S_1) = \mathrm{infer}\,(&\emptyset,\, S_0 \uplus \{\!|\, A = H \,|\!\} \uplus \biguplus_i (C_i \uplus S_i)\rho \uplus\\ &\biguplus_j (C_j \uplus S_j)\rho \uplus \mathrm{constr}(\boldsymbol{G}_0) \uplus \mathrm{constr}(\boldsymbol{B}))\end{aligned} \tag{Prf.1}$$

where consistent(C_1) and $(C, S) \approx_{\boldsymbol{G}_0 \uplus \boldsymbol{B} \uplus S_0 \uplus \{\!| A=H |\!\}} (C_1, S_1)$. Now, let $S' = S_0 \uplus \{\!| A = H |\!\} \uplus \biguplus_i (C_i \uplus S_i)\rho \uplus \biguplus_j (C_j \uplus S_j)\rho \uplus \mathrm{constr}(\boldsymbol{G}_0) \uplus \mathrm{constr}(\boldsymbol{B})$ and $S'' = \{\!| A = H |\!\} \uplus \biguplus_j (C_j \uplus S_j)\rho \uplus \mathrm{constr}(\boldsymbol{B})$. Moreover let $(C_2, S_2) = \mathrm{infer}(\emptyset, S'')$. Because $S'' \subseteq S'$ by (3.3) we have consistent(C_2). By IH again on $\boldsymbol{B}$, for some $m < n$ we can construct the derivation $\langle \boldsymbol{B}, \emptyset, \{\!| A = H |\!\}\rangle (\xrightarrow{r|c})^m \xrightarrow{is} \langle \emptyset, C_3, S_3\rangle$ where

$$(C_3, S_3) \approx_{\boldsymbol{B} \uplus \{\!| A=H |\!\}} (C_2, S_2). \tag{Prf.2}$$

Hence $\langle A, C_3, S_3\rangle_{/\equiv} \in SS_P$ because we can build the derivation

$$\langle A, \emptyset, \emptyset\rangle \xrightarrow{r} \langle \boldsymbol{B}, \emptyset, \{\!| A = H |\!\}\rangle (\xrightarrow{r|c})^m \xrightarrow{is} \langle \emptyset, C_3, S_3\rangle$$

Then we can choose a renamed apart $\langle A', C_q, S_q\rangle$ and a suitable variables substitution ρ' s.t. (Prf.1) holds for ρ' instead of ρ and

$$\langle A', C_q, S_q\rangle \rho' = \langle A, C_3, S_3\rangle. \tag{Prf.3}$$

Now let S''' be $S' - S''$ with ρ' instead of ρ and let

$$(C_4, S_4) = \mathrm{infer}(\emptyset, (S''') \uplus C_3 \uplus S_3). \tag{Prf.4}$$

Because of the renaming apart property of $\langle A', C_q, S_q\rangle$ and from (Prf.3) follows $(C_2, S_2) \approx_{\boldsymbol{G} \uplus A \uplus S_0} (C_q, S_q)$. Now by Lemma 3.7, (Prf.1), (Prf.2), (Prf.4) and (3.3) several times on $S' - S''$ we obtain $(C_4, S_4) \approx_{\boldsymbol{G} \uplus S_0} (C, S)$ This and (3.2) implies consistent(C_4). The (3.3) property can be iteratively applied because consistent(C_1).

The proof is so concluded. ■

Theorem 3.8 *If the transition system is incremental and R is quick checking then the SS_P^R semantics is AND-compositional w.r.t. computed answer constraints.*

Proof. We can apply Lemma 3.4 and prove the equivalent result

$$\begin{aligned}&\langle \boldsymbol{G}, \emptyset, \emptyset\rangle \xrightarrow{r|c}{}^n \xrightarrow{is} \langle \emptyset, C, S\rangle \iff \\ &\quad \forall G_i \in \mathrm{atoms}(\boldsymbol{G})\ \exists \langle G_i', C_i, S_i\rangle \text{ renamed apart versions of elements in} \\ &\quad SS_P \text{ and } \exists \rho \text{ variables substitution s.t. } G_i = G_i'\rho, \\ &\quad (C', S') = \mathrm{infer}(\emptyset, \textstyle\biguplus_i (C_i \uplus S_i)\rho \uplus \mathrm{constr}(\boldsymbol{G})) \text{ and consistent}(C')\end{aligned}$$

where $(C, S) \approx_{\boldsymbol{G}} (C', S')$. The thesis follows immediately from Lemma 3.7. ■

Theorem 3.9 *Let R be a quick checking computation rule. Suppose that SS_P^R is AND-compositional w.r.t. computed answer constraints and independent with respect to all quick checking computation rules. Then the transition system is incremental.*

Proof. Let $S = \{\!| c_1 \ldots, c_n |\!\}$. For proving (3.4) simply apply the *AND*-compositionality of SS_P on $\boldsymbol{G} = \emptyset$. To prove (3.3) let c be a constraint, $\mathrm{infer}(\emptyset, S \uplus \{\!| c |\!\}) = (C_1, S_1)$ and consistent(C_1). By applying the *AND*-compositionality of SS_P on

$\boldsymbol{G} = S \uplus \{\!| c |\!\}$ this is equivalent to $\langle \boldsymbol{G}, C_2, S_2\rangle \in \mathrm{Cac}(P, \boldsymbol{G})$ where $(C_1, S_1) \approx_{\boldsymbol{G}} (C_2, S_2)$. For the independence of computed answer constraints upon the selection rule this is equivalent to $\langle S \uplus \{\!| c |\!\}, \emptyset, \emptyset\rangle \xrightarrow{cis}{}^{n} \langle \{\!| c |\!\}, C_3, S_3\rangle \xrightarrow{cis} \langle \emptyset, C_4, S_4\rangle$ where $(C_2, S_2) \approx_{\boldsymbol{G}} (C_4, S_4)$. This is equivalent to

$$\langle S, \emptyset, \emptyset\rangle \xrightarrow{cis}{}^{n} \langle \emptyset, C_5, S_5\rangle \tag{Prf.1}$$

and $\langle \{\!| c |\!\}, C_5, S_5\rangle \xrightarrow{cis} \langle \emptyset, C_6, S_6\rangle$ where $(C_3, S_3) \approx_S (C_5, S_5)$ and $(C_4, S_4) \approx_{\{\!| c |\!\}} (C_6, S_6)$. By applying the *AND*-compositionality of SS_P on (Prf.1) this is equivalent to

$$(C_7, S_7) = \mathrm{infer}\,(\emptyset, S),\ \mathrm{consistent}(C_7)$$

where $(C_5, S_5) \approx_S (C_7, S_7)$. Hence for *cis* rule definition

$$(C_6, S_6) = \mathrm{infer}\,(C_5, S_5 \uplus \{\!| c |\!\}), \mathrm{consistent}(C_6).$$

Now let $(C_8, S_8) = \mathrm{infer}\,(C_7, S_7 \uplus \{\!| c |\!\})$, then for (3.1) we have

$$(C_7, S_7, C_8 \uplus S_8) \equiv (C_5, S_5, C_6 \uplus S_6)$$

and by (3.2) consistent(C_8). By $\equiv$ transitivity $(C_8, S_8) \approx_{\boldsymbol{G}} (C_1, S_1)$. Finally from (3.2) we have that consistent(C_1) which is the claim. ■

Note that, from Theorem 3.8 and Theorem 3.9 it follows that, under the hypothesis of independence upon the computation rule, the incrementality condition is equivalent to the *AND*-compositionality of SS_P^R. However, it is possible to relax our notion of *AND*-compositionality, by combining the atoms according to a fixed computation rule, thus obtaining a weaker incrementality condition.

4 Bottom-up semantics

In this section we show a bottom-up characterization of our semantics and prove that, for every incremental transition system, it is equivalemnt to the top-down construction.

The immediate consequences operator is defined by exploiting the properties of incremental transition system, namely Lemma 3.4 and 3.7.

Definition 4.1 *The* immediate consequences operator *of the program P is*

$$\begin{aligned} T_P(I) = \{\, & \langle p(\boldsymbol{x}), C, S\rangle_{/\equiv} \mid \\ & \quad \textit{$\boldsymbol{x}$ are new variables, } H \leftarrow \boldsymbol{B} \in P,\ \forall B_i \in \mathrm{atoms}(\boldsymbol{B}) \\ & \quad \exists\langle B'_i, C_i, S_i\rangle \textit{ renamed apart versions of elements of } I \textit{ and a} \\ & \quad \textit{variables substitution } \rho \textit{ s.t. } B_i = B'_i\rho, \\ & \quad (C, S) = \mathrm{infer}\,(\emptyset, \{\!| p(\boldsymbol{x}) = H |\!\} \uplus \mathrm{constr}(\boldsymbol{B}) \uplus \textstyle\biguplus_i (C_i \uplus S_i)\rho) \\ & \quad \textit{and } \mathrm{consistent}(C)\ \} \end{aligned}$$

The following lemma can be proved using standard techniques.

Lemma 4.2 *The T_P operator is continuous.*

Proof. It is easy to see that T_P is monotone. We have to prove that is also finitary, i.e., $T_P(\bigcup^\infty I_n) \subseteq \bigcup^\infty T_P(I_n)$.

Let $I_0 \subseteq I_1 \subseteq \dots I_n \subseteq \dots$ be a chain. If $g \in T_P(\bigcup^\infty I_n)$ then for T_P definition there must exist a finite number of goals $g_1, \dots, g_n \in \bigcup^\infty I_n$ from which g is built. In particular it must exist a finite k s.t. $g_1, \dots, g_n \in I_k$ and then, applying T_P definition again $g \in T_P(I_k)$ which implies $g \in \bigcup^\infty T_P(I_n)$. ■

Definition 4.3 *Let P be a program. The* bottom-up semantics *of P is $F_P = T_P \uparrow \omega$.*

Theorem 4.4 *Consider an incremental transition system. For any quick checking computation rule R we have $SS_P^R = F_P$*

Proof. Let's prove the two inclusions separately.

$\boldsymbol{SS_P \subseteq F_P}$] Let $\langle p(\boldsymbol{x}),\, C^q,\, S^q\rangle \in SS_P$, that is, for some n there exists the quick checking derivation $\langle p(\boldsymbol{x}),\, \emptyset,\, \emptyset\rangle \xrightarrow{(r|c)is}{}^n \langle \emptyset,\, C^q,\, S^q\rangle$. From Lemma 3.4 there exists the late checking derivation $\langle p(\boldsymbol{x}),\, \emptyset,\, \emptyset\rangle \xrightarrow{r|c}{}^n \xrightarrow{is} \langle \emptyset,\, C^\ell,\, S^\ell\rangle$ where $(C^q,\, S^q) \approx_{p(\boldsymbol{x})} (C^\ell,\, S^\ell)$.

We prove the inclusion by induction on the number of r or c transitions of the late checking system. Note that the first rule is always an r rule. For $n = 1$ there exists a renamed apart clause $H.$ of P such that

$$\langle p(\boldsymbol{x}),\, \emptyset,\, \emptyset\rangle \xrightarrow{r} \langle \emptyset,\, \emptyset,\, \{\!|p(\boldsymbol{x}) = H|\!\}\rangle \xrightarrow{is} \langle \emptyset,\, C^\ell,\, S^\ell\rangle$$

By definition of T_P we have that $\langle p(\boldsymbol{x}),\, C^\ell,\, S^\ell\rangle_{/\sim} \in T_P \uparrow 1$.

For $n > 1$ without loss of generality consider the following derivation where $H \leftarrow \boldsymbol{B}$ is a renamed apart version of clause of P.

$$\langle p(\boldsymbol{x}),\, \emptyset,\, \emptyset\rangle \xrightarrow{r} \langle \boldsymbol{B},\, \emptyset,\, \{\!|p(\boldsymbol{x}) = H|\!\}\rangle \xrightarrow{r|c}{}^{n-1} \langle \emptyset,\, \emptyset,\, S_0\rangle \xrightarrow{is} \langle \emptyset,\, C^\ell,\, S^\ell\rangle$$

Note that for $n = 1$ the body $\boldsymbol{B}$ is empty and most of the following formulas became trivial. The derivation

$$\langle \boldsymbol{B},\, \emptyset,\, \{\!|p(\boldsymbol{x}) = H|\!\}\rangle \xrightarrow{r|c}{}^{n-1} \langle \emptyset,\, \emptyset,\, S_0\rangle$$

does not depend on the accumulated constraints because no i and s transitions are applied. So we can build the derivation

$$\langle \boldsymbol{B},\, \emptyset,\, \emptyset\rangle \xrightarrow{r|c}{}^{n-1} \langle \emptyset,\, \emptyset,\, S_0 - \{\!|p(\boldsymbol{x}) = H|\!\}\rangle \xrightarrow{is} \langle \emptyset,\, C_1,\, S_1\rangle$$

where, if we let $(C_2,\, S_2) = \text{infer}\,(\emptyset,\, \{\!|p(\boldsymbol{x}) = H|\!\} \uplus C_1 \uplus S_1)$, $(C^\ell,\, S^\ell) \approx_{\boldsymbol{B}} (C_2,\, S_2)$ by (3.3). Then, by Theorem 3.8, for each atom $B_i \in \text{atoms}(\boldsymbol{B})$ there exists $\langle B_i',\, C_i,\, S_i\rangle$ renamed apart versions of elements in SS_P and ρ such that $B_i = B_i'\rho$ and

$$(C_3,\, S_3) = \text{infer}\,(\emptyset,\, \text{constr}(\boldsymbol{B}) \uplus \biguplus_i (C_i \uplus S_i)\rho)$$

where consistent(C_3) and $(C_1, S_1) \approx_{\boldsymbol{B}} (C_3, S_3)$. All the previous derivations are smaller than n so applying the IH we obtain $\langle B_i', C_i, S_i\rangle \in T_P \uparrow \omega$. In particular, because of the finiteness of $\boldsymbol{B}$, there exists some k such that for each i $\langle B_i', C_i, S_i\rangle \in T_P \uparrow k$. Moreover

$$
\begin{aligned}
&\text{infer}\,(\emptyset,\, \{\!|p(\boldsymbol{x}) = H|\!\} \uplus (\text{constr}(\boldsymbol{B}) \uplus \biguplus_i (C_i \uplus S_i)\rho)) \approx_{p(\boldsymbol{x})} && [\text{by } (3.3)]\\
&\text{infer}\,(\emptyset,\, \{\!|p(\boldsymbol{x}) = H|\!\} \uplus C_3 \uplus S_3) \approx_{p(\boldsymbol{x})} && [\text{by } (3.1)]\\
&\text{infer}\,(\emptyset,\, \{\!|p(\boldsymbol{x}) = H|\!\} \uplus C_1 \uplus S_1) \approx_{p(\boldsymbol{x})}\\
&(C^\ell, S^\ell) \approx_{p(\boldsymbol{x})}\\
&(C^q, S^q)
\end{aligned}
$$

and consistent(C^q). Hence by T_P definition $\langle p(\boldsymbol{x}), C^q, S^q\rangle_{/\sim} \in T_P \uparrow (k+1)$. The thesis then follows by T_P monotonicity.

$\boldsymbol{SS_P} \supseteq \boldsymbol{F_P}$] We prove by induction on n that $T_P \uparrow n \subseteq SS_P$.

The $n = 0$ case is trivial, because $T_P \uparrow 0 = \emptyset \subseteq SS_P$. For $n \geq 1$ let $\langle p(\boldsymbol{x}), C, S\rangle \in T_P \uparrow n$. Then for some clause $H \leftarrow \boldsymbol{B} \in P$ $\forall B_i \in$ atoms$(\boldsymbol{B})$ $\exists \langle B_i', C_i, S_i\rangle$ renamed apart versions of elements of $T_P \uparrow (n-1)$ and ρ s.t. consistent(C) and $(C, S) =$ infer$\,(\emptyset, U \uplus \text{constr}(\boldsymbol{B}) \uplus \biguplus_i K_i)$ where $U = \{\!|p(\boldsymbol{x}) = H|\!\}$ and $K_i = (C_i \uplus S_i)\rho$. Note that for $n = 1$ the body $\boldsymbol{B}$ is empty and most of the following formulas become trivial.

By applying Lemma 3.7 with $S_0 = U$ we have that $\langle \boldsymbol{B}, \emptyset, U\rangle \longrightarrow^* \langle \emptyset, C', S'\rangle$ where $(C, S) \approx_{\boldsymbol{B} \uplus U} (C', S')$. Then $\langle p(\boldsymbol{x}), \emptyset, \emptyset\rangle \longrightarrow^* \langle \emptyset, C', S'\rangle$ which is the thesis.

■

5 Conclusions

In this paper we have presented a semantic characterization of "real" CLP languages. This is made possible by parameterizing the semantic construction with respect to functions which model the behavior of the actual constraint solver employed. This idea is due to [6]. However, the semantics presented in [6] suffers from several drawbacks. We have overcome these limitations, presenting a semantics for quick-checking systems which is *AND*-compositional (under the assumption of incrementality of the constraint solver), captures both active and passive constraints, and has natural top-down and bottom-up construction processes. Our incrementality assumption (which is missing in [6]) does not exclude any reasonable system. Moreover, we have shown that if the incrementality assumption does not hold then no *atom-based semantics* can be both *AND*-compositional and independent from the computation rule.

The semantic treatment presented in this paper is not the only way to characterize CLP systems employing incomplete constraint solvers. For instance, the delay mechanism can be captured at the constraint system level, instead of carrying over a distinguished set of delayed (or passive) constraints in the semantics. In [1] a class of constraint systems is introduced, where constraints are cc agents [8]. In this approach, the CLP($\mathcal{R}$) constraint $X = Y * Z$ is considered as a shorthand for

$$\boldsymbol{ask}\big(\mathit{ground}(Y) \vee \mathit{ground}(Z)\big) \to \boldsymbol{tell}(X = Y * Z),$$

expressing the fact that $X = Y * Z$ (being non-linear) is delayed until it becomes linear, namely, until either Y or Z are constrained to take a single value.

If we are interested in complete constraint solvers only, then the semantics defined in [6] boils down to the CLP version of the s-semantics [4]. In this case the *AND*-compositionality problem has been deeply studied, even with respect to a much more concrete semantics, yielding stronger results [3].

References

[1] R. Bagnara. A hierarchy of constraint systems for data-flow analysis of constraint logic-based languages. Technical Report TR-96-10, Dipartimento di Informatica, Università di Pisa, 1996. To appear on a special issue of "Science of Computer Programming".

[2] A. Bossi, M. Gabbrielli, G. Levi, and M. Martelli. The s-semantics approach: Theory and applications. *Journal of Logic Programming*, 19-20:149–197, 1994.

[3] M. Comini and G. Levi. An algebraic theory of observables. In M. Bruynooghe, editor, *Proceedings of the 1994 Int'l Symposium on Logic Programming*, pages 172–186. The MIT Press, Cambridge, Mass., 1994.

[4] M. Gabbrielli, G. M. Dore, and G. Levi. Observable semantics for constraint logic programs. *J. Logic Computation*, 5(2):133–171, 1995.

[5] J. Jaffar and J. L. Lassez. Constraint Logic Programming. In *Proc. Fourteenth Annual ACM Symp. on Principles of Programming Languages*, pages 111–119. ACM, 1987.

[6] J. Jaffar and M. J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19-20:503–581, 1994.

[7] J. Jaffar, S. Michaylov, P. Stuckey, and R. Yap. The CLP($\mathcal{R}$) Language and System. *ACM Transactions on Programming Languages and Systems*, 14(3):339–395, 1992.

[8] V. A. Saraswat. *Concurrent Constraint Programming.* MIT Press, Cambridge, Mass., 1993.