Comparative Semantics for a Parallel Contextual Logic Programming Language[1]

Jean-Marie Jacquet[2] and Luís Monteiro[3]

Abstract

The purpose of this paper is to present and compare six semantics, ranging in the operational, declarative and denotational types, for a parallel version of contextual logic programming. Three operational semantics are first discussed. They all rest on a transition system but differ in their ability of describing success sets, failure sets, infinite computations and of handling multiple occurrences of computations. Two declarative semantics are then described. They extend, respectively, the Herbrand interpretation and the immediate consequence operator to our contextual framework. Finally, a denotational semantics based on processes, structured as trees, is given. The mathematical tools mainly used for these semantics are complete lattices for the declarative semantics and metric spaces for the other ones.

The parallel logic language under consideration is an elementary one: it uses or-parallelism and and-parallelism in an unrestricted manner. A reconciliation calculus is provided as a way of combining substitutions resulting from the reductions of conjoined goals. Despite its simplicity, we believe that the parallel language still constitutes a model of interest since it captures the basis of contextual logic programming and of parallel logic programming.

1 Introduction

Contextual logic programming ([MP89]) is an extension of the logic programming paradigm based on the idea of having both local and context-dependent predicate definitions. A language is proposed for supporting local definitions of predicates of the kind provided by systems of modules, and context-dependency in the form of predicate definitions implicitly supplied by the context. On the one hand, the clauses comprising a program are distributed over several modules (or "units", as they will be called here), and in that sense a predicate definition is local to the unit where the corresponding clauses occur. On the other hand, the definition of a predicate may depend on predicates not defined in the same unit, and in that case the definitions available in the context for those predicates are assumed by default.

We turn in this paper to a quite simple parallel version of the contextual logic programming framework. It just involves and-parallelism and or-parallelism without any concern for guard-like constructs, commitment, read-only annotations, mode declarations or other suspension constructs. We shall also not tackle negation here. However, our purpose is not to provide the reader with a practical parallel contextual logic language nor to discuss some parallel implementation. It is semantical and, more precisely, it consists of presenting and of relating semantical models for it.

[1]Part of this work was carried out in the context of ESPRIT Basic Research Action (3020) Integration.

[2]Centre for Mathematics and Computer Science, P.O. Box 4079, 1009 AB Amsterdam, The Netherlands.

[3]Departamento de Informática, Universidade Nova de Lisboa,2825 Monte da Caparica, Portugal.

With respect to this aim, we believe that the parallel contextual logic language treated here—subsequently referred to as CLL—is of interest since it captures the basis of contextual logic programming and of parallel logic programming. As an additional argument, our future research (under development) for more practical and more elaborated concurrent contextual logic languages will be based on the results exposed here.

The paper presents and compares six semantics issued from the logic programming and imperative traditions and ranging in the classical operational, declarative and denotational types. And-parallelism is treated in a quite close way to real concurrent executions : to allow a goal to progress from one step, it is sufficient that one of its subgoals performs one step, although all of them are allowed to do so. Restated in other terms, in contrast with work such as [BKRP89], our modelling of and-parallelism includes the interleaving perception of parallel computations as well as the true concurrent one. For simplicity of the exposition, or-parallelism is not treated in the same way but more implicitly as a choice. Some parallelism is however still captured in the sense that no order is imposed on the way clauses should be selected for reduction. It should also be noted that our modelling of or-parallelism and and-parallelism allows to capture the different (concurrently executed) and/or search subtrees, corresponding, in the parallel framework, to SLD-derivation paths. Repetition of such subtrees is furthermore taken into account in some semantics by means of multi-sets.

Our six semantics are composed of three operational semantics, two declarative semantics and one denotational semantics. Four of them, namely the operational semantics O_{bu} and O_{td}, and the two declarative semantics $Decl_m$ and $Decl_f$, take place in the logic programming tradition. The other ones, called O_{ch} and Den, are issued from the imperative tradition, especially from its metric branch.

The operational semantics O_{bu} rests on the so-called bottom-up derivation relation. It describes successful derivations of goals in a bottom-up fashion but does not produce any substitution. It is however interesting since it is close to the declarative reading of the clauses and thus help, in the one hand, in understanding the declarative semantics and, on the other hand, in relating the operational and declarative semantics.

The operational semantics O_{td} also rests on a derivation relation. It describes the derivation in a top-down manner and associates a computed answer substitution with each of them. It thus corresponds to the classical success set and failure set characterizations of programs.

The two declarative semantics $Decl_m$ and $Decl_f$, are based on model and fixed-point theory, respectively. They generalize the notions of Herbrand interpretation and consequence operator for classical Horn clause logic in order to take into account the context dependency of the truth of formulae. As suggested, an effort has been made to keep these semantics as simple as possible as well as in the main streams of logic programming semantics. However, context-dependency and parallel executions raise new problems, for which fresh solutions are proposed.

The third operational semantics O_{ch} completes the operational description of O_{bu} and O_{td} by handling multiple occurrences of computations as well as infinite computations. It furthermore tackles more closely the computation steps and, therefore, makes the modelling of and-parallelism fully apparent. Technically speaking, it is based on computation histories represented as streams of actions. Multiplicity of occurrences is handled by means of multi-sets.

The denotational semantics Den, defined as usual compositionally, further details the computation by handling choice-points i.e. points of possible alternatives

of use of unifiable clauses. It uses (as usual, too), processes organized in tree-like structures.

Although they are of classical inspiration, these last two semantics still present some originality with related work ([BZ82], [BKMOZ86], [BM88], [B88], [KR88], [BK88], [BKRP89], ...). It arises essentially from the four following points :

i) our concern with contextual logic programming, which has not been done before and which requires new solutions;
ii) the novel way (including interleaving and true concurrency) in which parallelism is modelled;
iii) the handling of multiple occurrences of computations;
iv) our use of local state and of reconciliation to combine them, which allows to define the denotational semantics more simply; in particular, the processes are expressed here just in terms of very intuitive computation steps - input substitutions, actions and output substitutions - rather than functions.

The semantical tools mainly used in this paper are of four types : sets, multi-sets, complete lattices and metric spaces. Despite this variety, the semantics have been related throughout the paper. Lack of space prevents us however from giving proofs. Nevertheless, all the propositions stated hereafter have been proved in [Mo89] and [Ja90].

The remainder of this paper is organized into 8 Sections. Section 2 describes the basic constructs of the language and explains our terminology. Section 3 recalls the basic semantical tools used in the paper. Section 4 presents (and compares) the three operational semantics according to their power of expression : O_{bu}, O_{td} and O_{ch}. Section 5 discusses the declarative models $Decl_m$ and $Decl_f$ and connects them with the operational semantics. Section 6 specifies the denotational semantics Den and compares it with the operational semantics O_{ch} and, consequently, in view of previous results, to the other semantics. Finally, section 7 sums up the relationship established in the paper and gives our conclusions.

2 The language CLL

As usual in logic programming, the language CLL comprises denumerably infinite sets of *variables*, *functions* and *predicates*, subsequently referred to as *Svar*, *Sfunct* and *Spred*, respectively. It also includes a set *Sunit* of so-called *unit names*, characterized by the property that every element u has attached a finite subset of Spred, called the *sort* of u and denoted by *sort(u)*. The sets Svar, Sfunct, Spred and Sunit are assumed to be pairwise disjoint.

The notions of term, atom, clause, substitution, unification, ...are defined as usual. We do not recall them here but rather specify some contextual related notions as well as some useful notations.

An *extension formula* is a formula of the form $u \gg \overline{G}$ where u is a unit name and $\overline{G}$ is a finite conjunction of atomic or extension formulae. A *general atom* (*g-atom*) is an atomic or an extension formula. It is typically denoted by the letters A, B, C, ... A *general goal* (*g-goal*) is a finite conjunction of g-atoms. It is typically denoted by the symbols $\overline{A}$, $\overline{B}$, $\overline{C}$, ..., $\overline{G}$, ... The empty g-goal is denoted by the $\triangle$ letter. Clauses take here the form $H \leftarrow \overline{B}$ and allow extension formulae to take place in their bodies. Given an atom $A = p(t_1,\ldots,t_m)$, we denote by *name(A)* the predicate name of A, namely p. A set of clauses is said to *define* a predicate p if it contains a clause whose head's name is p.

A *unit* is a formula of the form u : U, where u $\in$ Sunit and U is a finite set of clauses such that the set of predicates defined in U is sort(u). We call u the *name* or *head* of the unit and U its *body*. A *system of units* is a set $\mathcal{U}$ of units such that no two distinct units in $\mathcal{U}$ have the same name. For a unit in $\mathcal{U}$ with name u, we denote its body by $|u|_{\mathcal{U}}$, or simply |u| if $\mathcal{U}$ is understood. In the sequel we will often abuse language and refer to u as a unit in $\mathcal{U}$ when in fact we mean the unit u : |u|. The set of systems is subsequently referred to as *Ssyst*.

A *context* is a stack of units. It is referred to by its name, consisting of an arbitrary sequence of unit names. The set of context names, *Scontext*, is thus the free monoid $\text{Sunit}^{<\omega}$. Context names are represented by juxtaposition, as in uv. The empty sequence λ is employed as the name of the *empty* context. The context resulting from *extending* the context c with unit u (i.e. by putting u on top of the stack) is denoted by uc.

3 Mathematical preliminaries

3.1 Sets and multi-sets

Executions may result in computing a same answer or a same computation path several times. Multi-sets, allowing an element to be repeated, are used subsequently to capture this repetition. To clearly distinguish them from sets, they are denoted by adding the *ms* label to the {...} brackets, as in $\{a,a,b\}_{ms}$, whereas sets are denoted by the *s* label, as in $\{a,b\}_s$. The union symbol $\cup$ is also subscripted in this way for the same purpose. To avoid any ambiguity, let us further precise that, given two multi-sets S and T, we denote by $S \cup_{ms} T$ the collection of all elements of S and T repeated as many times as they occur in S and T.

The usual notations $\mathcal{P}(E)$ and $\mathcal{M}(E)$ are used to denote, respectively, the set of sets and multi-sets, with elements from E. The notations $\mathcal{P}_\pi(E)$ and $\mathcal{M}_\pi(E)$ are moreover employed to denote those sets and multi-sets verifying the property π. For instance, $\mathcal{M}_{nf}(E)$ (resp. $\mathcal{M}_{nco}(E)$) denotes the set of non-empty and finite[4] multi-sets (resp. the non-empty and compact multi-sets) with elements from E.

3.2 Reconciliation of substitutions

Full use of and-parallelism requires a way of combining substitutions issued from the concurrent reductions of subgoals of a g-goal in order to form answer substitutions for the whole g-goal. It has been provided under the name of *reconciliation of substitutions* in [Ja89] and has been extensively studied there. Concurrently, an equivalent notion, named parallel composition of substitutions, has been developed in [Pa88] and [Pa90]. We briefly recall this notion here for the sake of completeness. The reader is referred to the above three references for more details.

The reconciliation of substitutions is based on the interpretation of substitutions in equational terms. Precisely, any substitution $\theta = \{X_1/t_1, \ldots, X_m/t_m\}_s$ is associated with the system of the equations $X_1 = t_1, \ldots, X_m = t_m$, subsequently referred to as *syst(θ)*. Reconciling substitutions then consists of solving systems composed of the associated equations.

Concepts of unifiers and mgus can be defined for these systems in a straightforward way. It is furthermore possible to relate the unification of systems of equations

[4] To avoid confusion, we precise that a multi-set is finite iff it contains only a finite number of elements (and thus not iff any element occurs a finite number of times).

with that of terms in such a way that all properties of the unification of terms transpose to the unification of systems of equations. In particular, mgus of systems can be proved to be equal modulo renaming. We consequently use, in the following, the classical abuse of language and speak of *the* mgu of a unifiable system. It is referred to as *mgu_syst(S)*, where S is the system under consideration.

We are now in a position to define the notion of reconciliation of substitutions.

Definition 1 *The substitutions $\theta_1, \ldots, \theta_m$ ($m \geq 1$) are reconcilable iff the system composed of the equations of syst(θ_1), ..., syst(θ_m) is unifiable. When so, its mgu is called the* reconciliation *of the substitutions. It is denoted by $\rho(\theta_1, \ldots, \theta_m)$.* ■

The equational interpretation of substitutions requires, at some point, the idempotence of the substitutions. This is not a real restriction since any unifiable terms or systems of equations admit an idempotent mgu. It is furthermore to our point of view the natural one. For ease of the discussion, we will take the convention of using, from now on, idempotent substitutions only. Their set is referred to as *Ssubst.*

3.3 Complete lattices and metric spaces

Complete lattices and metric spaces will be used as important semantical tools. The reader is assumed to be familiar with them as well as with their related notions of convergent sequences, directed, closed and compact subsets, completeness, continuous and contracting functions .... He is also assumed to be familiar with Tarski's lemma, describing the set of prefixed points of continuous functions of complete lattices, and Banach's theorem, stating the existence of a unique fixed point of contractions in complete metric spaces. He is referred to [Ll87] and [En77], when need be. Furthermore, lack of space prevents us from describing all the metrics used in this paper. We will however employ essentially the classical ones and refer to [B88] for such a description. The only exception to this rule concerns the metric on multi-sets that we now make precise.

Proposition 2 *Let E be some set and let $\perp$ be an element not in E. Let furthermore*[5] *$.[.] : E \times \mathbf{N} \rightarrow E \cup_s \{\perp\}_s$ be a function such that,*

i) for any $e \in E$: $e[0] = \perp$;
ii) for any $e, f \in E$:if $e[n] = f[n]$, for all $n \in \mathbf{N}$, then e=f;
iii) for any $e \in E, m, n \in \mathbf{N}$: (e[m])[n]=e[min{m,n }$_s$].

Such a function is subsequently called truncation. Define, for any $S \in \mathcal{M}(E)$ and $n \in \mathbf{N}$, S[n] as the multi-set $\{s[n] : s \in S\}_{ms}$. Furthermore, define, for any $e, f \in E$, $S, T \in \mathcal{M}(E)$,

$$d_E(e, f) = 2^{-sup\{n : e[n] = f[n]\}_s};$$

$$d_{ms}(S, T) = 2^{-sup\{n : S[n] = T[n]\}_s}.$$

Then, the space (E, d_E) is a metric space. Moreover, the spaces ($\mathcal{M}_{nf}(E), d_{ms}$) and ($\mathcal{M}_{nco}(E), d_{ms}$), where compactness[6] *is taken with respect to d_E, are metric spaces. They are complete if (E, d_E) is complete.* ■

[5] We denote by **N** the set of non negative integers.

[6] Compactness is extended straightforwardly from sets to multi-sets : a mutli-set M is compact iff any sequence of elements of M contains a subsequence converging to an element of M.

4 Operational semantics

4.1 Bottom-up derivation

The first characterization of the operational semantics of CLL is expressed in terms of the notion of bottom-up derivation. Its interest arises from its closeness to the "declarative" reading of clauses. It is twofold. On the one hand, it helps in understanding the declarative semantics, to be presented in section 5. On the other hand, it helps in proving the equivalence between the operational and the declarative semantics.

The notion of bottom-up derivation is characterized indirectly by defining a "bottom-up derivation relation". It takes the form $c \vdash^{bu}_{\mathcal{U}} \overline{G}$, for a system of units $\mathcal{U}$, a context name c and a g-goal $\overline{G}$. The intended meaning is that *every* ground instance of $\overline{G}$ is true in the situation represented by the context c. The relation $\vdash^{bu}_{\mathcal{U}}$ is defined more formally by means of rules of the form

$$\frac{\text{Assumptions}}{\text{Conclusion}} \quad \textit{if} \quad \text{Conditions},$$

asserting the Conclusion whenever the Assumptions and Conditions hold. (Note that Assumptions and Conditions may be absent from some rules.) Precisely, it is defined as the smallest relation of Ssyst × Scontext × Sgoal satisfying the following rules (N-B) to (E-B), by case analysis on the form of $\overline{G}$. The notation *Sinst(U)* is used to denote the set of all (possibly non-ground) instances of clauses in U, and $\vdash^{bu}_{\mathcal{U}}$ is written simply as $\vdash^{bu}$, for readability.

Null formula

(N-B) $$\frac{}{c \vdash^{bu} \Delta}$$

Conjunction

(C-B) $$\frac{c \vdash^{bu} A_1, \ldots, c \vdash^{bu} A_m}{c \vdash^{bu} A_1, \ldots, A_m}$$

Atomic formula—local reduction

(R-B) $$\frac{uc \vdash^{bu} \overline{B}}{uc \vdash^{bu} H} \quad \textit{if} \quad H \leftarrow \overline{B} \in \text{Sinst}(|u|)$$

Atomic formula—contextual definition

(X-B) $$\frac{c \vdash^{bu} A}{uc \vdash^{bu} A} \quad \textit{if} \quad \text{name}(A) \notin \text{sort}(u)$$

Extension formula

(E-B) $$\frac{uc \vdash^{bu} \overline{G}}{c \vdash^{bu} u \gg \overline{G}}$$

The first three rules are essentially the same as for Horn clause logic. They state, respectively, that true can be derived in any context, that a conjunction is derivable if its conjuncts are, and that the head of a clause is derivable if its body is. The rule (X-B) explains the meaning of contextual definition: an atom is derivable in a context whose top unit does not define the atom's predicate name if the atom is derivable in the context with the top unit removed. The last rule (E-B) characterizes context extension: an extension formula is derivable in a context if the "inner" conjunction is derivable in the context extended with the unit mentioned in the extension formula.

Definition 3 *Let δ^- and δ^+ be two fresh symbols. The* bottom-up operational semantics *of CLL is the function O_{bu} : Ssyst $\rightarrow$ Scontext $\rightarrow$ Sgoal $\rightarrow \{\delta^-, \delta^+\}_s$ defined as follows: for any $\mathcal{U}$∈Ssyst, c∈Scontext and $\overline{G}$∈Sgoal,*

$$O_{bu}(\mathcal{U})(c)(\overline{G}) = \begin{cases} \delta^+, & \text{if } c \vdash^{bu}_{\mathcal{U}} \overline{G} \\ \delta^-, & \text{otherwise} \end{cases}$$

■

4.2 Top-down derivation

The operational semantics O_{bu} does not deliver that much information, just the possible existence of a successful derivation. The purpose of any computation is however far more richer: to compute bindings for the variables of the query. The notion of successful top-down derivation allows precisely to capture such an idea. Given a system of units $\mathcal{U}$ and a g-goal $\overline{G}$, it consists of a sequence of steps reducing $\overline{G}$ to the null conjunction. Associated with it, there is a substitution θ representing the values computed for the variables of $\overline{G}$. The expected result (presented in section 5) is that the universal closure of $\overline{G}\theta$ is a logical consequence of $\mathcal{U}$, and that any instance of $\overline{G}$ which is a consequence of $\mathcal{U}$ can be obtained as instance of $\overline{G}\theta$ for some computed substitution θ.

As before, the top-down derivation is not specified directly, but by means of a top-down derivation relation. For any context name c and g-goal $\overline{G}$, $c \vdash^{td}_{\mathcal{U}} \overline{G}\ [\theta]$, or more simply $c \vdash^{td} \overline{G}\ [\theta]$ when $\mathcal{U}$ is understood, denotes the fact that there is a (successful) *top-down derivation of* $\overline{G}$ *in* c *from* $\mathcal{U}$ *with substitution* θ. Again, $\vdash^{td}_{\mathcal{U}}$ is formally defined as the smallest relation of Ssyst × Scontext × Sgoal × Ssubst satisfying the rules below. The symbol ϵ denotes the empty (identity) substitution.

Null formula

(N-T) $$\frac{}{c \vdash^{td} \Delta\ [\epsilon]}$$

Conjunction

(C-T) $$\frac{c \vdash^{td} A_1\ [\theta_1], \ldots, c \vdash^{td} A_m\ [\theta_m]}{c \vdash^{td} A_1,\ldots,A_m\ [\rho(\theta_1, \ldots, \theta_m)]}$$

Atomic formula—local reduction

(R-T) $$\frac{uc \vdash^{td} \overline{B}\theta\ [\sigma]}{uc \vdash^{td} A\ [\theta\sigma]} \quad \textit{if} \quad \begin{cases} H \leftarrow \overline{B} \in |u|^7 \\ \theta = \text{mgu(A,H)} \end{cases}$$

Atomic formula—contextual definition

(X-T) $$\frac{c \vdash^{td} A\ [\theta]}{uc \vdash^{td} A\ [\theta]} \quad \textit{if} \quad \text{name(A)} \notin \text{sort(u)}$$

Extension formula

(E-T) $$\frac{uc \vdash^{td} \overline{G}\ [\theta]}{c \vdash^{td} u \gg \overline{G}\ [\theta]}$$

These rules have a reading similar to the bottom-up case. For example, rule (C-T) states that in order to derive a g-goal we must derive each g-atom and then reconcile the resulting substitutions.

The relationship between the top-down and bottom-up derivations is established by the following proposition.

[7] As usual, a suitable renaming of the clauses is assumed.

Proposition 4 *If* $c \overset{td}{\vdash} \overline{G}\ [\theta]$ *then* $c \overset{bu}{\vdash} \overline{G}\theta$. *Conversely, if* $c \overset{bu}{\vdash} \overline{G_o}$ *and* $\overline{G_o}$ *is an instance of* $\overline{G}$, *there is a substitution* θ *such that* $c \overset{td}{\vdash} \overline{G}\ [\theta]$ *and* $\overline{G_o}$ *is an instance of* $\overline{G}\theta$. ∎

The *top-down operational semantics* can now be characterized. As may be seen in the following definition, it corresponds to the usual notion of success set and failure set.

Definition 5 *Define the* top-down operational semantics *as the following function* $O_{td} : Ssyst \rightarrow Scontext \rightarrow Sgoal \rightarrow \mathcal{P}(Ssubst)$: *for any* $\mathcal{U} \in Ssyst$, $c \in Scontext$, *and* $\overline{G} \in Sgoal$, $O_{td}(\mathcal{U})(c)(\overline{G}) = \{\theta : c \overset{td}{\vdash}_{\mathcal{U}} \overline{G}\ [\theta]\}_s$. ∎

The following result relating O_{bu} and O_{td} is an immediate consequence of the previous proposition.

Proposition 6 *Let* $\alpha_1 : \mathcal{P}(Subst) \rightarrow \{\delta^-, \delta^+\}_s$ *be the function defined as*

i) $\alpha_1(\emptyset) = \delta^-$
ii) $\alpha_1(\Sigma) = \delta^+$, *if* $\Sigma \neq \emptyset$

One has $O_{bu} = \alpha_1 \circ O_{td}$. ∎

4.3 Computation histories

Although more powerful than O_{bu}, the operational semantics O_{td} suffers from two problems: it cannot cope with infinite computations and cannot distinguish multiple occurrences of computations. The operational semantics O_{ch} is introduced as a remedy. It essentially delivers the histories of the computations rather than just their results and collects all their repetitions. The main technicalities used for that purpose are as follows. Repetition is captured by using multi-sets rather than sets. Histories are modelled by words whose elements represent the multi-set of unifications and context updates (namely the basic operations of CLL) performed at each step, as well as the two termination status, failure and success. These histories are formally identified by means of a labelled transition system, in the style of [Pl81], whose labels correspond to the multi-sets of basic actions and whose configurations are some generalization of the goals. This extension is justified by the fact that, in order to represent truly concurrent executions, any g-atom of any g-goal must operate in a private working memory space, namely its own state and its own context.

This intuition given, let us define O_{ch} more precisely. The following notation and definitions specify the concepts sketched above.

Notation 7 (Histories) *The notations* unif(A,B), cxt_ext(c,u) *and* cxt_pop(c) *are used to represent the actions of unifying the atoms A and B, of extending the context c by the unit u, and of poping the context c, respectively. The set of such* basic actions *is referred to as* Sact. *The set of words formed from* $\mathcal{M}_{nf}(Sact)$ *and which finite elements are ended by one of the terminator operators* δ^-, *representing failure, and* δ^+, *representing success, is referred to as* Shist. *Elements of Shist are called* histories. ∎

Definition 8 (Extended g-atoms and goals) Extended g-atoms (eg-atoms) *are constructs of the form A* in $< \sigma, c >$ *where A is a g-atom,* σ *is a substitution and c is a context. They are typically denoted by the letters* $\widehat{A}$, $\widehat{B}$, $\widehat{C}$, ... Extended

g-goals (eg-goals) *are conjunctions of eg-atoms. They are typically referred to as* $\widetilde{A}$, $\widetilde{B}$, $\widetilde{C}$, ..., $\widetilde{G}$, ... *The empty eg-goal is denoted by the* Δ_{ext} *symbol. The set of eg-goals is referred to by* Sextgoal. *Finally,* $(A_1,\ldots,A_m)$ in $< \sigma, c >$ *is defined as the eg-goal* $(A_1$ in $< \sigma, c >)$, ..., $(A_m$ in $< \sigma, c >)$. ■

Definition 9 (Transition relation) *The transition relation used for specifying the operational semantics* O_{ch} *is defined as the smallest relation* $\rightarrow$ *of* $Ssyst \times Sextgoal \times \mathcal{M}_{nf}(Sact) \times Sextgoal$ *satisfying the following rules (R-H) to (C-H*$_3$*). For ease of reading, the more suggestive notation* $\widetilde{G} \xrightarrow{l} \widetilde{G^*}$ *is employed instead of* $(\mathcal{U}, \widetilde{G}, l, \widetilde{G^*})$.

Atomic formula - local reduction

$$(R\text{-}H) \quad \frac{}{A \text{ in } < \sigma, uc > \xrightarrow{l} \overline{B} \text{ in } < \sigma^*, uc >} \quad \text{if} \begin{cases} (H \leftarrow \overline{B}) \in |u|^{8} \\ A\sigma \text{ and } H \text{ are unifiable} \\ \sigma^* = \sigma \circ mgu(A\sigma, H) \\ l = \{unif(A\sigma, H)\}_{ms} \end{cases}$$

Atomic formula—contextual definition

$$(X\text{-}H) \quad \frac{}{A \text{ in } < \sigma, uc > \xrightarrow{l} A \text{ in } < \sigma^*, c >} \quad \text{if} \begin{cases} name(A) \notin sort(u) \\ l = \{cxt\_pop(uc)\}_{ms} \end{cases}$$

Extension formula

$$(E\text{-}H) \quad \frac{}{u \gg \overline{G} \text{ in } < \sigma, c > \xrightarrow{l} \overline{G} \text{ in } < \sigma, uc >} \quad \text{if} \quad l = \{cxt\_ext(c, u)\}_{ms}$$

Conjunction

$$(C\text{-}H_1) \quad \frac{\widetilde{A} \xrightarrow{l} \widetilde{A^*}}{\widetilde{A}, \widetilde{B} \xrightarrow{l} \widetilde{A^*}, \widetilde{B}} \qquad (C\text{-}H_2) \quad \frac{\widetilde{B} \xrightarrow{l} \widetilde{B^*}}{\widetilde{A}, \widetilde{B} \xrightarrow{l} \widetilde{A}, \widetilde{B^*}}$$

$$(C\text{-}H_3) \quad \frac{\widetilde{A} \xrightarrow{l_1} \widetilde{A^*}, \widetilde{B} \xrightarrow{l_2} \widetilde{B^*}}{\widetilde{A}, \widetilde{B} \xrightarrow{l_1 \cup_{ms} l_2} \widetilde{A^*}, \widetilde{B^*}}$$

■

Rules (R-H), (X-H) and (E-H) essentially rephrase the rules (R-T), (X-T) and (E-T) defining how atoms and extension formulae should be treated. Rules (C-H_1), (C-H_2) and (C-H_3) define the and-parallel execution of conjoined eg-atoms. It is worth noting that, thanks to rules (CH_1)and (C-H_2), all eg-atoms need not be reduced in one step in order to allow the whole conjunction to perform one reduction step. Such maximal parallel executions can however take place thanks to rule (C-H_3). Our modelling of and-parallelism is thus very close to the real practical operational executions: it expresses concurrent executions waiting for some processing resource as well as the fully concurrent executions when enough computing resources are available. Reformulated in a more conceptual level, our modelling of and-parallelism subsumes the interleaving approach to concurrency as well as the truly concurrent one (assuming, as usual, that all unfications take the same amount of time). Notice finally that no reconciliation takes place in the rule (C-H_3). Indeed, conjoined eg-atoms of a g-goal are first reduced before reconciliation is performed to determine the answer susbstitution for the whole g-goal. Hence, no reconciliation is present in the computation histories, capturing the only computed unifications and context operations. It will however be used to relate O_{td} and O_{ch}.

[8] As usual, a suitable renaming of the clauses is assumed.

We are now in position to define the operational semantics O_{ch}. Note that the finitely branching property of the transition relation is not sufficient to ensure that the codomain of **O** and O_{ch} is composed of non-empty and finite multi-sets of histories. However, it is strong enough to ensure that it is composed of compact ones.

Definition 10

1) *Define* $\mathbf{O} : Ssyst \to Sextgoal \to \mathcal{M}_{nco}(Shist)$ *as follows : for any* $\mathcal{U} \in Ssyst$, *any* $\widetilde{G} \in Sextgoal$[9],

$$\begin{aligned}\mathbf{O}(\mathcal{U})(\widetilde{G}) \;=\; & \{l_1.l_2.\cdots.l_n.\delta^- : \widetilde{G} \xrightarrow{l_1} \widetilde{A_1} \xrightarrow{l_2} \ldots \xrightarrow{l_n} \widetilde{A_n} \neq \Delta_{ext}, \widetilde{A_n} \not\to \}_{ms} \\ & \cup_{ms}\{l_1.l_2.\cdots.l_n.\delta^+ : \widetilde{G} \xrightarrow{l_1} \widetilde{A_1} \xrightarrow{l_2} \ldots \xrightarrow{l_n} \Delta_{ext}\}_{ms} \\ & \cup_{ms}\{l_1.l_2.\cdots.l_n.\cdots : \widetilde{G} \xrightarrow{l_1} \widetilde{A_1} \xrightarrow{l_2} \ldots \xrightarrow{l_n} \widetilde{A_n} \xrightarrow{l_{n+1}} \ldots\}_{ms}.\end{aligned}$$

2) *Define the* computational history operational semantics *as the following function* $O_{ch} : Ssyst \to Scontext \to Ssubst \to Sgoal \to \mathcal{M}_{nco}(Shist)$: *for any* $\mathcal{U} \in Ssyst$, *any* $c \in Scontext$, *any* $\sigma \in Ssubst$, *any* $\overline{G} \in Sgoals$,

$$O_{ch}(\mathcal{U})(c)(\sigma)(\overline{G}) = \mathbf{O}(\mathcal{U})(\overline{G} \text{ in } <\sigma, c>). \qquad \blacksquare$$

It is worth noting that the auxiliary function **O** can be related to the fixed point of the following higher-order contraction Ψ_{op} reflecting the recursive nature of **O**. This property combined with the fortunate circumstance that contractions have one fixed point will be usefull later to relate O_{ch} with the denotational semantics Den.

Definition 11 *Define* $\Psi_{op} : [Ssyst \to Sextgoal \to \mathcal{P}_{nco}(Shist)] \to [Ssyst \to Sextgoal \to \mathcal{P}(Shist)]$ *as follows: for any* $F \in [Ssyst \to Sextgoal \to \mathcal{P}_{nco}(Shist)]$, *any* $\mathcal{U} \in Ssyst$, *any* $\widetilde{G} \in Sextgoal$,

$$\begin{aligned}\Psi_{op}(F)(\mathcal{U})(\widetilde{G}) \;=\; & \{\delta^- : \widetilde{G} \neq \Delta_{ext}, \widetilde{G} \not\to \}_s \cup_s \{\delta^+ : \widetilde{G} = \Delta_{ext}\}_s \\ & \cup_s \{l.h : \widetilde{G} \xrightarrow{l} \widetilde{G^*}, h \in F(\mathcal{U})(\widetilde{G^*})\}_s.\end{aligned} \qquad \blacksquare$$

Proposition 12

1) *The function* Ψ_{op} *is a contraction from* $[Ssyst \to Sextgoal \to \mathcal{P}_{nco}(Shist)]$ *to* $[Ssyst \to Sextgoal \to \mathcal{P}_{nco}(Shist)]$.
2) *Let* $\beta : \mathcal{M}(Shist) \to \mathcal{P}(Shist)$ *be the function that transforms any multi-set in the corresponding set, namely the function defined by* $\beta(M) = \{m : m \in M\}_s$ *for any* $M \in \mathcal{M}(Shist)$. *The function* $\beta \circ \mathbf{O}$ *is a fixed point of* Ψ_{op}. $\blacksquare$

It is here worth noting that using multi-sets instead of sets in the above definition makes Ψ_{op} a non-contracting function. In fact, this multi-set version, say Ψ^*_{op}, has an infinite number of fixed points, obtained by progressively duplicating computation paths in **O**. Hence, multiplicity cannot be handled suitably at the stream level by contractions. This fact will force us to use two arguments to relate O_{ch} and Den: the classical contracting argument to establish that elements are preserved and an additional argument to prove that the multiplicity of occurrence of elements is also conserved.

[9] The multiplicity of occurences of elements of the multi-sets is left unformal and intuitive for the sake of clarity. A precise formal definition can however be stated by extending the transition relation with an additional argument specifying the points of duplication of transitions.

We conclude this section by relating the operational semantics O_{td} and O_{ch}. Roughly speaking, all we have to do is, for each history, to perform all the unifications it contains and to reconcile the results. This is more precisely achieved by means of the three following functions.

Definition 13

1) Define eq_act *as follows: for any* $l \in Sact$,

$$eq\_act(l) = \begin{cases} \{A = B\}_s & \text{if } l = unif(A,B) \\ \emptyset & \text{if } l = cxt\_ext(c,u) \text{ or } l = cxt\_pop(c) \end{cases}$$

2) Define mgu_syst_hist *as follows: for any* $h \in Shist$, $mgu\_syst\_hist(h)$ *is*

$$\begin{cases} \{mgu\_syst(\bigcup_{i=1,\ldots,m}{}_s \bigcup_{a \in l_i}{}_s eq\_act(a))\}_s, \text{ if } h = l_1 \ldots . l_m.\delta^+ \\ \emptyset, \text{ if } h = l_1 \ldots . l_m.\delta^- \text{ or if } h \text{ is infinite} \end{cases}$$

3) Define α_2 *as follows: for any* $MH \in \mathcal{M}_{nco}(Ssyst)$,

$$\alpha_2(MH) = \bigcup_{h \in MH}{}_s \{mgu\_syst\_hist(h)\}_s. \quad \blacksquare$$

Relating O_{td} and O_{ch} then consists of noting, by an inductive reasoning on reductions and histories, that, for any system $\mathcal{U}$, any context c and any g-goal $\overline{G}$, $O_{td}(\mathcal{U})(c)(\overline{G})$ and $[\alpha_2 \circ O_{ch}](\mathcal{U})(c)(\epsilon)(\overline{G})$ are identical.

Proposition 14 *For any* $\mathcal{U} \in Ssyst$, *any* $c \in Scontext$, $\overline{G} \in Sgoal$, *one has*

$$O_{td}(\mathcal{U})(c)(\overline{G}) = [\alpha_2 \circ O_{ch}](\mathcal{U})(c)(\epsilon)(\overline{G}). \quad \blacksquare$$

5 Declarative semantics

5.1 Model theory

The operational semantics is concerned with proof, the declarative semantics with truth. The declarative semantics of a system of units is characterized by the set of g-goals that are true in every model of the system of units. The purpose of this section is to show how such semantics can be defined.

The first task is to find an appropriate notion of interpretation for CLL. An interpretation of Horn clause logic is a subset of the Herbrand base, intended to record the set of all facts that are true under the interpretation. In CLL, which facts are true depend on the context. An interpretation of CLL must thus provide a way to associate a subset of the Herbrand base with every context. Such subsets are called "situations".

Let $S_I(c)$ be the situation associated with the context c under the interpretation I. One must have $S_I(\lambda)=\emptyset$ since no facts are true in the empty context. Moreover, the situation $S_I(uc)$ must be the "update" by u of the previous situation $S_I(c)$, and consequently the equality $S_I(uc)=I_u(S_I(c))$ must hold for some function I_u depending on u. A unit name u is then interpreted as a "situation update" I_u, formalized as a mapping from situations to situations. Furthermore, given the intended meaning of context extension, such a function must redefine the predicates defined in the unit and leave the other predicates unchanged. The required notion of interpretation is then a family of updates I_u indexed by $u \in Sunit$. These ideas are now made precise.

Definition 15 *The Herbrand base of CLL is as usual the set HB of all ground atoms built with Sfunct and Spred. A* situation *is a subset of HB. If $P \subseteq$ Spred and $S \subseteq HB$, the* restriction *of S to P is the set $S \downarrow [P] = \{p(t_1,\ldots,t_n) \in S : p \in P\}_s$. To simplify the notation, $S \downarrow$ [Spred-P] is abbreviated to $S \downarrow$ [-P] .* ■

Definition 16 *A continuous mapping $t : \mathcal{P}(HB) \rightarrow \mathcal{P}(HB)$ is called an* update *with respect to $P \subseteq$ Spred if, for every $S \subseteq HB$, it satisfies the two following conditions:*

i) Preservation: *$t(S) \downarrow [-P] = S \downarrow [-P]$ (atoms with names not in P are preserved by t).*

ii) Dependency: *$t(S) = t(S \downarrow [-P])$ (the update depends only on the preserved atoms).* ■

Definition 17 *An* interpretation *I of CLL is a family $I = (I_u)_{u \in Sunit}$, where each I_u is an update with respect to sort(u).* ■

Definition 18 *Given a situation S, a finite set F of formulae and an interpretation I, the fact that* the formulae in *F* are true in *S* with respect to *I, denoted $S \models_I F$, is defined by the cases below. Let ground(F) be the set of all ground instances of formulae in F. Let us furthermore write $S \models_I f$ instead of $S \models_I \{f\}_s$.*

i) Sets: *$S \models_I F$ if and only if $S \models_I f$ for every $f \in F$.*

ii) Units: *$S \models_I u{:}U$ if and only if $I_u(S) \models_I U$.*

iii) Clauses : *$S \models_I H \leftarrow \overline{B}$ if and only if $S \models_I H_0 \leftarrow \overline{B}_0$ for all $(H_0 \leftarrow \overline{B}_0) \in ground(H \leftarrow \overline{B})$.*

iv) Ground clauses : *$S \models_I H \leftarrow \overline{B}$ if and only if $S \models_I H$ whenever $S \models_I \overline{B}$.*

v) Ground extension formulae : *$S \models_I u \gg \overline{G}$ if and only if $I_u(S) \models_I \overline{G}$.*

vi) Ground atomic formulae : *$S \models_I A$ if and only if $A \in S$.* ■

The way this relation is defined is standard, with the possible exception of units and ground extension formulae. A unit is true in a given situation if the body is true in the situation updated by the denotation of the unit name. Extension formulae are interpreted similarly.

The notion of model, central for the declarative semantics, can now be defined.

Definition 19 *An interpretation I is a* model *of a set F of formulae if $S \models_I F$ for every $S \subseteq HB$. A formula f is a* consequence *of F, denoted $F \models f$, if every model of F is a model of f.* ■

The case of interest is when F is a system of units $\mathcal{U}$ and f is a g-goal $\overline{G}$. The (model-theoretic) declarative semantics will now be defined for this case. In the next definition, for a context name $c = u_1 \ldots u_n$, $c \gg \overline{G}$ will be used as a shorthand for the extension formula $u_n \gg \ldots \gg u_1 \gg \overline{G}$.

Definition 20 *Define the* declarative semantics *$Decl_m$: Ssyst $\rightarrow$ Scontext $\rightarrow$ Sgoal $\rightarrow \mathcal{P}$(Ssubst) as follows: for any $\mathcal{U} \in$ Ssyst, $c \in$ Scontext, and $\overline{G} \in$ Sgoal:*

$$Decl_m(\mathcal{U})(c)(\overline{G}) = \{\theta : \forall \overline{G}_0 \in ground(\overline{G}\theta), \mathcal{U} \models c \gg \overline{G_0}\}_s.$$ ■

5.2 Fixed-point theory

The models of a system of units $\mathcal{U}$ can be characterized as the prefixed points of a continuous operator $T_{\mathcal{U}}$: Sint $\rightarrow$ Sint associated with $\mathcal{U}$, where Sint is the set (complete lattice) of all interpretations. This characterization has two important

consequences that follow directly from Tarski's lemma. The first is that $\mathcal{U}$ always has a minimal model $M_{\mathcal{U}}$, given by the least fixed point of $T_{\mathcal{U}}$. The second is that there is a standard iterative procedure for computing the least fixed point of $T_{\mathcal{U}}$, and hence $M_{\mathcal{U}}$. These facts are used to define a fixed-point semantics $Decl_f$.

First, note that the set Sint of all interpretations $I = (I_u)_{u \in Sunit}$, partially ordered by $I \leq J$ if and only if $I_u(S) \subseteq J_u(S)$ for every $u \in$ Sunit and $S \subseteq HB$, is a complete lattice. The mapping $T_{\mathcal{U}}$: Sint $\rightarrow$ Sint associated with $\mathcal{U}$ can now be defined.

Definition 21 *Let $T_{\mathcal{U}}$: Sint $\rightarrow$ Sint be the mapping defined as follows : for every interpretation I, $T_{\mathcal{U}}(I)$ is the interpretation J such that*

$$J_u(S) = (S \downarrow [\text{-}sort(u)]) \cup_s \{A : \exists(A \leftarrow \overline{B}) \in ground(\lceil u \rceil), I_u(S) \models_I \overline{B}\}_s$$

for every $u \in$ Sunit and $S \subseteq HB$. ∎

This mapping is well defined (that is $T_{\mathcal{U}}(I)$ is an interpretation, for every interpretation I) and continuous. Moreover, it can be shown that an interpretation I is a model of $\mathcal{U}$ if and only if $T_{\mathcal{U}}(I) \leq I$. By Tarski's lemma, $T_{\mathcal{U}}$ has a least fixed point, which therefore is also the least model of $\mathcal{U}$. This statement is the contents of the next proposition.

Proposition 22 *Every system of units $\mathcal{U}$ has a minimal model $M_{\mathcal{U}}$, given as the least fixed point of $T_{\mathcal{U}}$.* ∎

The fixed-point semantics is defined in terms of the least fixed point of $T_{\mathcal{U}}$, as usual.

Definition 23 *Define the* fixed-point semantics *$Decl_f$: Ssyst $\rightarrow$ Scontext $\rightarrow$ Sgoal $\rightarrow \mathcal{P}$(Ssubst) as follows: for any $\mathcal{U} \in$ Ssyst, $c \in$ Scontext, and $\overline{G} \in$ Sgoal:*

$$Decl_f(\mathcal{U})(c)(\overline{G}) = \{\theta : \forall \overline{G}_0 \in ground(\overline{G}\theta), \emptyset \models_{M_{\mathcal{U}}} c \gg \overline{G_0}\}_s,$$

where $M_{\mathcal{U}}$ is the minimal model of $\mathcal{U}$. ∎

The equivalence between the declarative and the fixed-point semantics is based on the following observation. There are two quantifications involved in the consequence relation $\mathcal{U} \models \overline{G}$, one over all models of $\mathcal{U}$ and the other over all situations. It is possible to eliminate both, by considering only the minimal model of $\mathcal{U}$ and the truth of $\overline{G}$ in the empty situation with respect to that model.

Proposition 24 *For every system of units $\mathcal{U}$ and g-goal $\overline{G}$, $\mathcal{U} \models \overline{G}$ if and only if $\emptyset \models_{M_{\mathcal{U}}} \overline{G}$. In particular, the declarative and the fixed-point semantics coincide, that is $Decl_m$=$Decl_f$.* ∎

The remainder of this section describes the connection between the operational and the declarative semantics. The relationship between the bottom-up derivation and the consequence relation $\models$ is established via the association of a situation with every context.

Definition 25 *Let I be an interpretation of a system of units $\mathcal{U}$. For every context name c, the* situation *$S_I(c)$* determined by *c* under *I is defined inductively as follows:*

i) $S_I(\lambda) = \emptyset$,

ii) $S_I(uc) = I_u(S_I(c))$. ∎

Proposition 26 *Given a system of units $\mathcal{U}$ with minimal model M, a context name c and a g-goal $\overline{G}$, one has $c \vdash^{bu}_{\mathcal{U}} \overline{G}$ if and only if $S_M(c) \models_M \overline{G_0}$ for every $\overline{G_0} \in ground(\overline{G})$.* ∎

The equivalence between the bottom-up operational semantics and the declarative semantics is an easy consequence of the previous result.

Proposition 27 *One has $O_{bu} = \alpha_1 \circ Decl_m$.* ∎

The next result relates the consequence relation to the top-down derivation relation.

Proposition 28 *If $\overline{G_0}$ is a ground instance of a g-goal $\overline{G}$ and if c is a context name, then $S_M(c) \models_M \overline{G_0}$ if and only if $c \vdash^{td}_{\mathcal{U}} \overline{G}[\theta]$ for some substitution θ such that $\overline{G_0}$ is an instance of $\overline{G}\theta$.* ∎

It is now easy to establish the equivalence between the top-down operational semantics and the declarative semantics.

Proposition 29 *Let $\alpha_3 : \mathcal{P}(Ssubst) \to \mathcal{P}(Ssubst)$ be defined as follows: for every $\Sigma \in \mathcal{P}(Ssubst)$, $\alpha_3(\Sigma) = \{\sigma\theta : \sigma \in \Sigma, \theta \in Ssubst\}_s$. One has $Decl_m = \alpha_3 \circ O_{td}$.* ∎

6 Denotational semantics

This section introduces our last semantics. It makes no use of transition systems and no reference to any declarative paradigm but is defined compositionally on the basis of processes. They may be formally described as multi-sets of computation steps followed by new processes, each computation step being described as an input multi-set of substitutions, a multi-set of basic actions[10] and an output multi-set of substitutions. They are organised in tree-like structures to capture the place of the various choice of use of clauses. Two auxiliary processes, $p^+$and p^-, are furthermore introduced to indicate termination in the successful and failure status. In view of this rough introduction, the set of processes Sproc could be recursively defined by the equation

$$\text{Sproc} = \{p^+, p^-\}_s \cup_s \mathcal{M}_{nf}(\text{Scstep} \times \text{Sproc}) \tag{1}$$

where Scstep[11] is $\mathcal{M}_{nf}(\text{Ssubst}) \times \mathcal{M}_{nf}(\text{Sact}) \times \mathcal{M}_{nf}(\text{Ssubst})$. Recursive equations of this type have been solved in a metric setting in [BZ82] and [AR89]. The careful reader will however note that no multi-sets are tackled in these references and, furthermore, that, to apply their results, $\mathcal{M}_{nf}(\text{Scstep} \times \text{Sproc})$ should be endowed with a distance, which cannot be inferred from previous definitions unless a truncation on Scstep × Sproc is clearly specified. Anyway, the simplicity of the equation (1) allows us to solve it directly. This is achieved as in [BZ82], by first defining an auxiliary space Sfproc, by then endowing it with a distance d and by finally defining the metric space Sproc as the completion of (Sfproc,d). We will furthermore take profit of this direct solving to restrict the processes to those verifying the following property:

> the multi-sets of the initial substitutions of the first computation steps are identical. (2)

[10] Recall notation 7 of section 4.3.

[11] Read Scstep as the set of computation steps.

Definition 30 *Define the set of finite processes Sfproc inductively by the following rules :*

i) $p^+, p^- \in Sfproc$

ii) *if* $\omega_1 = (\Lambda_1, S_1, \Upsilon_1), \ldots, \omega_m = (\Lambda_m, S_m, \Upsilon_m) \in Scstep$, $p_1, \ldots, p_m \in Sfproc$, $m > 0$ *and* $\Lambda_1 = \ldots = \Lambda_m$, *then* $\{(\omega_1, p_1), \ldots, (\omega_m, p_m)\}_{ms} \in Sfproc$. ∎

Endowing Sfproc with a distance can be achieved by considering the union of the equation (1) as disjoint and by defining the truncation function .[.] on processes.

Definition 31 *Define the truncation function .[.] on Sfproc by the following rules:*

i) $p[0] = \perp$, *for any* $p \in Sfproc$,

ii) $p^+[n] = p^+$, *for any* $n > 0$,

iii) $p^-[n] = p^-$, *for any* $n > 0$,

iv) $\{(\omega_1, p_1), \ldots, (\omega_m, p_m)\}_{ms}[1] = \{\omega_1, \ldots, \omega_m\}_{ms}$,
for any $\omega_1, \ldots, \omega_m \in Scstep$, $p_1, \ldots, p_m \in Sfproc$,

v) $\{(\omega_1, p_1), \ldots, (\omega_m, p_m)\}_{ms}[n] = \{(\omega_1, p_1[n-1]), \ldots, (\omega_m, p_m[n-1])\}_{ms}$,
for any $\omega_1, \ldots, \omega_m \in Scstep$, $p_1, \ldots, p_m \in Sfproc$, $n > 1$. ∎

Definition 32 *Define the metric on Sfproc as follows:*

i) $d(p^+, p^+) = d(p^-, p^-) = 0$,

ii) $d(p^-, p) = 1 = d(p, p^-)$, *for any* $p \in Sfproc$, *distinct from* p^-,

iii) $d(p^+, p) = 1 = d(p, p^+)$, *for any* $p \in Sfproc$, *distinct from* p^+,

iv) $d(p_1, p_2) = d_{ms}(p_1, p_2)$, *for any* $p_1, p_2 \in Sfproc$, *distinct from* p^+ *and* p^-. ∎

The metric space of processes Sproc is then defined from the space of finite processes Sfproc by taking in addition all limits of Cauchy sequences of Sfproc. This is achieved precisely by defining Sproc as the completion of (Sfproc,d). Two interesting properties of these new processes is that their structure ressemble that of finite processes and that they do verify property (2). The common value of these Λ_i's of any process p is subsequently referred to as *init(p)*.

Defining a compositional semantics requires to define an operator $\tilde{\|}$ equivalent at the denotational level to the parallel composition operator "," between g- atoms. It is defined according to our modelling of parallel composition of section 4.3 and by means of a suitable contraction to handle correctly recursivity with infinite structures. Note that, because of the tree-like structure of the processes, the contractivity problems of Ψ^*_{op} do not occur here.

Definition 33 *Define* $\Psi_{\|} : [Sproc \times Sproc \to Sproc] \to [Sproc \times Sproc \to Sproc]$ *as follows: for any* $F \in [Sproc \times Sproc \to Sproc]$, *for any* $p \in Sproc$, *for any* $p_1, p_2 \in Sproc \setminus \{p^+, p^-\}_s$:

i) $\Psi_{\|}(F)(p^-, p) = p^- = \Psi_{\|}(F)(p, p^-)$;

ii) $\Psi_{\|}(F)(p^+, p) = p = \Psi_{\|}(F)(p, p^+)$;

iii) $\Psi_{\|}(F)(p_1, p_2) =$

$$\begin{aligned}
&\{(\omega*, F(p'_1, p'_2)) : ((\Lambda_1, S_1, \Upsilon_1), p'_1) \in p_1, ((\Lambda_2, S_2, \Upsilon_2), p'_2) \in p_2,\\
&\qquad \omega* = ((\Lambda_1 \cup_{ms} \Lambda_2), S_1 \cup_{ms} S_2, (\Upsilon_1 \cup_{ms} \Upsilon_2))\}_{ms}\\
&\cup_{ms} \{(\omega*, F(p'_1, p_2)) : ((\Lambda_1, S_1, \Upsilon_1), p'_1) \in p_1, \Lambda_2 = init(p_2),\\
&\qquad \omega* = ((\Lambda_1 \cup_{ms} \Lambda_2), S_1, (\Upsilon_1 \cup_{ms} \Lambda_2))\}_{ms}\\
&\cup_{ms} \{(\omega*, F(p_1, p'_2)) : ((\Lambda_2, S_2, \Upsilon_2), p'_2) \in p_2, \Lambda_1 = init(p_1),\\
&\qquad \omega* = ((\Lambda_1 \cup_{ms} \Lambda_2), S_2, (\Upsilon_2 \cup_{ms} \Lambda_1))\}_{ms}.
\end{aligned}$$ ∎

Proposition 34 *The function* $\Psi_{\|}$ *is well-defined and is a contraction.* ∎

Definition 35 *Define the function* $\tilde{\|}$: $Sproc \times Sproc \rightarrow Sproc$ *as the fixed point of* $\Psi_{\|}$. ∎

Proposition 36 *For any* $p_1, p_2, p_3 \in Sproc$, $(p_1 \;\tilde{\|}\; p_2) \;\tilde{\|}\; p_3 = p_1 \;\tilde{\|}\; (p_2 \;\tilde{\|}\; p_3)$. ∎

We are now in position to specify the denotational semantics Den. In view of its compositional nature, it is mainly defined by stating the denotational meaning of the basic cases, namely the empty g-goal and the g-goal reduced to one g-atom, which is quite straightforward in view of the preceeding sections. As before, undesirable problems with recursivity are avoided by means of a (well-defined) higher-order contraction Ψ_{den}. It is stated in terms of extended g-goals rather than g-goals in order to ease the relationship between O_{ch} and Den. Nevertheless, a similar contraction can be defined directly on g-goals (in a similar way) and its fixed point can be proved equal to Den, defined as the following restriction of **D**.

Definition 37 *Define* Ψ_{den} : $[Ssyst \rightarrow Sextgoal \rightarrow Sproc] \rightarrow [Ssyst \rightarrow Sextgoal \rightarrow Sproc]$ *as follows: for any* $F \in [Ssyst \rightarrow Sextgoal \rightarrow Sproc]$, *for any* $\mathcal{U} \in Ssyst$,

i) $\Psi_{den}(F)(\mathcal{U})(\Delta_{ext}) = p^{+}$

ii) $\Psi_{den}(F)(\mathcal{U})(A \text{ in } < \sigma, c >) = p^{-}$, *if* $c{=}\lambda$ *or if the following conditions holds:* $c = uc'$, *name(A)* $\in$ *sort(u), no clause of u unifiable with A.*

iii) $\Psi_{den}(F)(\mathcal{U})(A \text{ in } < \sigma, c >) = \{(\omega_1, F(\mathcal{U})(\widetilde{B_1})), \ldots, (\omega_m, F(\mathcal{U})(\widetilde{B_m}))\}_{ms}$, *if the following conditions holds :*

- $c = uc'$,
- *name(A)* $\in$ *sort(u),*
- $H_1 \leftarrow \overline{B_1}$, ..., $H_m \leftarrow \overline{B_m}$ *are all the clauses of u unifiable with* $A\sigma$, *say with (idempotent) mgu* θ_1, ..., θ_m, *respectively,*
- $\omega_i = (\{\sigma\}_{ms}, \{unif(A\sigma, H_i)\}_{ms}, \{\sigma \circ \theta_i\}_{ms})$, *for all i,*
- $\widetilde{B_i} = \overline{B_i} \text{ in } < \sigma \circ \theta_i, c >$.

iv) $\Psi_{den}(F)(\mathcal{U})(A \text{ in } < \sigma, c >) = \{(\omega, F(\mathcal{U})(A \text{ in } < \sigma, c' >)\}_{ms}$,
if $c{=}uc'$, *name(A)* $\notin$ *sort(u),* $\omega = (\{\sigma\}_{ms}, \{cxt\_pop(c)\}_{ms}, \{\sigma\}_{ms})$.

v) $\Psi_{den}(F)(\mathcal{U})(u \gg \overline{G} \text{ in } < \sigma, c >) = \{(\omega, F(\mathcal{U})(\overline{G} \text{ in } < \sigma, uc >)\}_{ms}$,
where $\omega = (\{\sigma\}_{ms}, \{cxt\_ext(c,u)\}_{ms}, \{\sigma\}_{ms})$.

vi) $\Psi_{den}(F)(\mathcal{U})((\widehat{A_1}, \ldots, \widehat{A_m}) \text{ in } < \sigma, c >)$
$= \Psi_{den}(F)(\mathcal{U})(\widehat{A_1} \text{ in } < \sigma, c >) \;\tilde{\|}\; \ldots \;\tilde{\|}\; \Psi_{den}(F)(\mathcal{U})(\widehat{A_m} \text{ in } < \sigma, c >)$. ∎

Proposition 38 *The function* Ψ_{den} *is well-defined and is a contraction.* ∎

Definition 39

1) Define $\mathbf{D}$: $Ssyst \rightarrow Sextgoal \rightarrow Sproc$ *as the (unique) fixed point of* Ψ_{den}.

2) Define the denotational semantics *Den* : $Ssyst \rightarrow Scontext \rightarrow Ssubst \rightarrow Sgoal \rightarrow Sproc$ *as follows: for any* $\mathcal{U} \in Ssyst$, *any* $c \in Scontext$, *any* $\sigma \in Ssubst$, *any* $\overline{G} \in Sgoal$, $Den(\mathcal{U})(c)(\sigma)(\overline{G}) = \mathbf{D}(\mathcal{U})(\overline{G} \text{ in } < \sigma, c >)$. ∎

We conclude this section by relating Den with the operational semantics O_{ch}. This achieved by relating the auxiliary functions **O** and **D**. Function **O** handles linear structures whereas function **D** manipulates tree-like structures. To relate them, we thus first need to introduce a function that, given some tree, produces the streams it contains. We then need to select the appropriate part of the computational set of O_{ch}, namely the action multiset part. This is the purpose of the

following function α_4. Its recursive nature suggests to define it as the fixed point of some higher-order contraction. However, because of the stream structure of the results, contractivity problems already encountered for Ψ^*_{op} makes the definition of such a contraction impossible. Anyway, the function α_4 can be defined correctly by taking advantage of the tree-like structure of the processes, especially by defining a notion corresponding to that of predecessors in trees. This is subsequently denoted by $\omega \hookrightarrow_p \omega^*$, $\omega \hookrightarrow_p \delta^+$, $\omega \hookrightarrow_p \delta^-$, with respective meaning that ω is a predecessor of ω^*, of δ^+ and of δ^- in p.

Definition 40 *Define* $\alpha_4 : Sproc \rightarrow \mathcal{M}(Shist)$ *as follows:*

i) $\alpha_4(p^-) = \{\delta^-\}_{ms}$,

ii) $\alpha_4(p^+) = \{\delta^+\}_{ms}$,

iii) $\alpha_4(p) = \{S_1.\cdots.S_m.\delta : (\Lambda_1, S_1, \Upsilon_1) \hookrightarrow_p \ldots \hookrightarrow_p (\Lambda_m, S_m, \Upsilon_m) \hookrightarrow_p \delta,$
$((\Lambda_1, S_1, \Upsilon_1), p^*) \in p, \delta \in \{\delta^+, \delta^-\}_s\}_{ms}$
$\cup_{ms} \{S_1.\cdots.S_m.\cdots : (\Lambda_1, S_1, \Upsilon_1) \hookrightarrow_p \ldots (\Lambda_m, S_m, \Upsilon_m) \hookrightarrow_p \cdots,$
$((\Lambda_1, S_1, \Upsilon_1), p^*) \in p\}_{ms}$[12]

for any $p \in Sproc \setminus \{p^+, p^-\}_s$. ∎

Proposition 41 *For any* $p \in Sproc$, $\alpha_4(p)$ *is non-empty and compact.* ∎

Relating **O** and **D** consists of proving two properties :

i) firstly that the function $\alpha_4 \hat{\circ} \mathbf{D} : Ssyst \rightarrow Sextgoal \rightarrow \mathcal{M}_{nco}(Shist)$ defined as

$$(\alpha_4 \hat{\circ} \mathbf{D})(\mathcal{U})(\widetilde{G}) = \alpha_4(\mathbf{D}(\mathcal{U})(\widetilde{G}))$$

is, after composition with the function β[13], a fixed point of Ψ_{op}. As $\beta \circ \mathbf{O}$ is also a fixed point of Ψ_{op}(see proposition 12) and since contractions have only one fixed point, it is then proved that the functions **O** and $\alpha_4 \hat{\circ} \mathbf{D}$ return the same elements. Hence, so are the corresponding restrictions O_{ch} and $\alpha_4 \hat{\circ}$Den (the function $\alpha_4 \hat{\circ}$Den is defined similarly to $\alpha_4 \hat{\circ} \mathbf{D}$).

ii) secondly, that the sources of duplication of elements are the same in **O** and $\alpha_4 \hat{\circ} \mathbf{D}$. If so, the elements of the multi-sets delivered by **O** and $\alpha_4 \hat{\circ} \mathbf{D}$ occur with the same multiplicity. Taking the respective restrictions, the same property then also holds for O_{ch} and $\alpha_4 \hat{\circ}$Den.

These two properties can indeed be proved so that one can claim the following proposition.

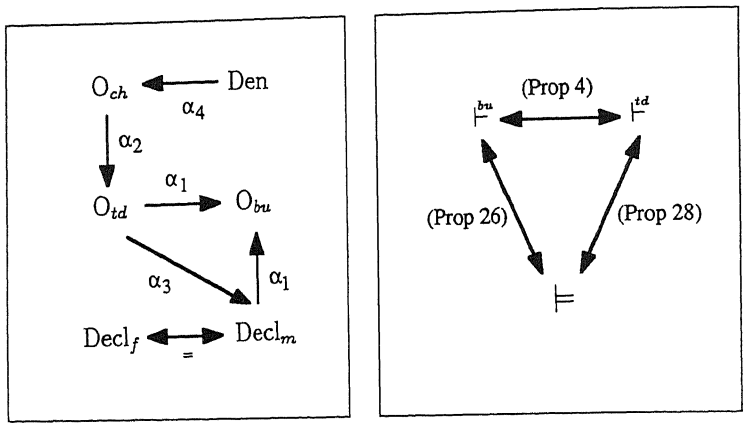
Proposition 42 *One has* $\mathbf{O} = \alpha_4 \hat{\circ} \mathbf{D}$ *and therefore* $O_{ch} = \alpha_4 \hat{\circ} Den$. ∎

7 Comparison and conclusion

The paper has presented six semantics ranging in the operational, declarative and denotational types. Four of them are inspired by the traditional logic programming paradigm. They consist of the operational semantics O_{bu} and O_{td}, based on the notions of bottom-up and top-down derivations, respectively, and of the declarative semantics $Decl_m$ and $Decl_f$, based on model theory and fixed-point theory, respectively. The two other semantics are issued from the imperative tradition, and, more

[12] As in definition 10, the multiplicity of elements of the multi-sets is left informal and intuitive for the sake of clarity. A formal definition can however be achieved by inserting an additional argument in the predecessor relation that distinguishes the nodes of the process tree.

[13] Recall the function β introduced in proposition 12.



a) The minimal relations

b) Equivalence of the bottom-up derivation, top-down derivation and satisfaction relation

Figure 1: Relations between the semantics

particularly, from its metric semantical branch ([BZ82], [BKMOZ86], [BM88], [B88], [KR88], [BK88], ...). They consist of the operational semantics O_{ch}, characterizing computations by means of streams, and of the denotational semantics Den, characterizing them, in a compositional way, via tree-like structures. All these semantics have been related throughout the paper, thanks to propositions 6, 14, 24, 27, 29, 42. They are summed up in figure 1a).

The minimal relations have only been stated in the paper. From them, it is possible to deduce other relations, for instance to connect Den with $Decl_f$. It is furthermore impossible to add nonredundant relations. For instance, it is impossible to relate O_{td} and O_{ch} since the former essentially delivers the results of the computation and the latter basically delivers histories of the computation. Similarly, it seems impossible to guess the choice point to build Den from O_{ch}. However, it is worth noting that although the semantics are different, it is possible to further connect the bottom-up derivation, the top-down derivation and the model theory. Propositions 4, 26, 28 and have, respectively, established the equivalence between the bottom-up and top-down derivations, the equivalence between the bottom-up derivation and the satisfaction relation, and the equivalence between the top-down derivation and the satisfaction relation. Hence, figure 1a) can be completed by figure 1b).

The parallel version of contextual logic programming presented in this paper is quite simple: it just includes or-parallelism and and-parallelism. It is not considered as a practical language to program with but rather as a first case study model. It is however quite interesting in the sense that it captures both the basis of contextual logic programming and of parallel logic programming. Our future work, under development, will be based on the results presented in this paper. It will be concerned with more elaborated versions including inheritance, guard-like constructs with related commitment operations, suspension conditions and some other more elaborated mechanisms (under development) for communication and concurrency. Also, we are trying to develop semantics closer to real computation in treating or-parallelism as real parallelism and not just as non-deterministic choice as in the

paper. Finally, we are investigating the relationship of our model with semantics of the partial order type such as pomsets or event structures (see e.g. [Gr81], [Pr86], [BW90]).

Acknowledgments

The idea of contextual logic programming has been developed in joint work with A. Porto, with whom we have discussed many of the topics of this paper. We also thank the C.W.I. concurrency group, composed by J.W. de Bakker, F. de Boer, F. van Breughel, A. de Bruin, E. Horita, P. Knijnenburg, J. Kok, J. Rutten, E. de Vink and J. Warmerdam, for comments on a previous version of this paper. In particular, the first author wishes to thank E. Horita and J. Warmerdam for their "every-day" intensive discussions.

The research reported herein has been partially supported by Esprit BRA 3020 (Integration). The first author likes to thank also the Belgian National Fund for Scientific Research as well as the University of Namur for having supported his past research, from which some ideas of this paper have arisen. The second author also thanks the Instituto Nacional de Investigação Científica for partial support.

References

[AR89] America P., Rutten J.J.M.M., Solving reflexive domain equations in a category of complete metric spaces, *Journal of Computer and System Sciences*, Vol 39, no. 3, 1989, pp. 343-375.

[B88] de Bakker J.W., Comparative Semantics for Flow of Control in Logic Programming without Logic, Report CS-R8840, Center for Mathematics and Computer Science, Amsterdam, The Netherlands, 1988, to appear in *Information and Computation.*

[BK88] de Bakker J.W., Kok J.N., Uniform Abstraction, Atomicity and Contractions in the Comparative Semantics of Concurrent Prolog, *Proc. of FGCS*, 1988, pp. 347-355.

[BKMOZ86] de Bakker J.W., Kok J.N., Meyer J.-J.Ch, Olderog E.-R., Zucker J.I., Contrasting Themes in the Semantics of Imperative Concurrency, in *Current Trends in Concurrency : Overviews and Tutorials* (J.W. de Bakker, W.P. de Roever, G. Rozengerg, eds.), Lecture Notes in Computer Science, Vol. 224, Springer-Verlag, 1986, pp. 51-121.

[BKRP89] de Boer F.S., Kok J.N., Palamidessi C., Rutten J.J.M.M., Semantic Models for a Version of PARLOG, *Proc. of the 6th Int. Conf. on Logic Programming*, 1989, pp. 621-636.

[BM88] de Bakker J.W., Meyer J.-J.Ch., Metric Semantics for Concurrency, *BIT*, 28, 1988, pp. 504-529.

[BW90] de Bakker J.W., Warmerdam J., Metric Pomset Semantics for a Concurrent Language with Recursion, to appear in *Proc. of the 18e Ecole de Printemps d'Informatique Théorique*, La Roche-Posay, France, 1990.

[BZ82] de Bakker J.W., Zucker J.I., Processes and the Denotational Semantics of Concurrency, *Information and Control* 54, 1982, pp.70-120.

[En77] Engelking R., *General Topology*, Polish Scientific Publishers, 1977.

[Gr81] Grabowski J., On Partial Languages, *Fundamenta Informaticae* IV.2, 1981, pp. 427-498.

[Ja89] Jacquet J.-M., *Conclog : a Methodological Approach to Concurrent Logic Programming*, Ph.D. Thesis, University of Namur, Belgium, 1989, to appear as Lecture Notes in Computer Science, Springer-Verlag.

[Ja90] Jacquet J.-M., *Semantics for a Concurrent Contextual Logic Programming Language*, to appear as Technical Report, Center for Mathematics and Computer Science, Amsterdam, The Netherlands.

[KR88] Kok J.N., Rutten J.J.M.M., Contractions in Comparing Concurrency Semantics, *Proc. 15th ICALP* (T. Leistö, A. Salomaa, eds.), Lecture Notes in Computer Science, Vol. 317, Springer-Verlag, 1988, to appear in *Theoretical Computer Science*.

[Ll89] Lloyd J., *Foundations of Logic Programming*, Springer-Verlag, 1987.

[M86] Miller D., A Theory of Modules for Logic Programming, *Proc. of the 1986 Symposium on Logic Programming*, 1986, pp. 106-114.

[M89] Miller D., A Logical Analysis of Modules in Logic Programming, *Journal of Logic Programming* (6), 1989, pp. 79-108.

[Mo89] Monteiro L., *The Semantics of Contextual Logic Programming*, Technical Report UNL DI-5/89, Departamento de Informática, Universidade Nova de Lisboa, Portugal, 1989.

[MP89] Monteiro L., Porto A., Contextual Logic Programming, *Proc. of the 6th Int. Conf. on Logic Programming*, 1989, pp. 284-299.

[Pa88] Palamidessi C., *A Fixpoint Semantics for Guarded Horn Clauses*, Technical Report CS-R8833, Center for Mathematics and Computer Science, Amsterdam, The Netherlands, 1988.

[Pa90] Palamidessi C., Algebraic Properties of Idempotent Substitutions, Technical Report TR-32/89, Dipartimento di Informatica, University of Pisa, Pisa, Italy, 1989, to appear in *Proc. of the 17th ICALP*, 1990.

[Pl81] Plotkin G.D., *A Structural Approach to Operational Semantics*, Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.

[Pr86] Pratt V., Modelling Concurrency with Partial Orders, *Int. Journal of Parallel Programming*, 15, 1986, pp. 33-71.