Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

J.-M. Jacquet, L. Monteiro

Comparative semantics for a parallel contextual programming language

Computer Science/Department of Software Technology Report CS-R9018 May

---

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Research (N.W.O.).

Comparative Semantics for a Parallel Contextual Programming Language*

Jean-Marie Jacquet [1]
Luís Monteiro [2]

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

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

Abstract

Recently, contextual logic programming has been proposed as an extension to the logic programming paradigm aiming at structuring programs and logical derivations in a coordinated way ([MP89]). The purpose of this paper is to present and compare various semantics for a parallel version of it. Six semantics, ranging in the operational, declarative and denotational types, are discussed. Three operational semantics are presented. They all rest on a transition system but differ in their ability of describing success set, failure set, infinite computations and of handling repetitions:

i) the first operational semantics just describes the set of atoms having a successful bottom-up derivation;
ii) the second operational semantics precises, in addition, the computed answer substitutions;
iii) the third operational semantics moreover characterizes infinite derivations.

As far as the declarative semantics are concerned, a model-theoretic and a fixed-point semantics are examined. They extend the Herbrand interpretation and the immediate consequence operator to our contextual framework. Finally, a denotational semantics based on processes, structured as trees, are 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: the results obtained on it — in particular, the semantical ones — are bases for results about more elaborated and more practical concurrent versions.

Key words and phrases: Operational semantics, denotational semantics, declarative semantics, parallelism, contextual logic languages.

1985 Mathematics Subject Classification 68Q10, 68Q55

1987 Computing Reviews Categories: D.1.3, D.3.1, F.1.2, F.3.2, F.4.1

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

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 implictly 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.

From the language point of view, only two new notions are required, when compared to Horn clause logic. One is that of a unit $u{:}\mathcal{U}$, which is a set of clauses $\mathcal{U}$ with name u. The other is that of an extension formula $u \gg \overline{G}$, which intuitively states that $\overline{G}$ is true in the context extended with the denotation of u. Here $\overline{G}$ is a conjunction of atoms, and possibly also of extension formulae.

The other required notion is semantics in nature, and is that of context. A context is simply a stack of units. Operationally, to derive a goal in a context, derive the goal (using the clauses) in the top unit until one of the three following situations occurs : (i) the goal fails or succeeds, (ii) an extension formula $u \gg \overline{G}$ has been selected, or (iii) the predicate of the selected atom is not defined in the top unit. In the first case, end the derivation in failure or in success. In the second case, "extend" the context with unit u (push u on top of the stack) and derive $\overline{G}$ in the extended context. In the last case, pop the top unit and derive the goal in the new context.

After this "operational" explanation, it may be somewhat surprising that contextual logic programming has an intuitively simple declarative semantics, not very different from that of Horn clause logic. A context determines a subset of the Herbrand base, to be called a "situation", comprising the facts that are true in the context. A unit has a declarative reading analogous to a Horn clause logic program, except that is is parameterized by the predicates mentioned but not defined in the unit. Since situations supply the missing meaning for those predicates, the semantic type of units is that of functions from situations to situations, called "updates". Updates are the declarative counterparts of context extension.

Example 1 *Throughout the paper the following example will be used:*

```
u:  p(1).                    v:  r(2).
    p(X) ← v ≫ r(X).             r(X) ← s(X).
    q(f(3)).
    s(f(Z)).
```

There are two units, with names u *and* v*, defining, respectively, the predicates* p, q, s *and* r*. Note, for example, that the goal* u ≫ p(X) *can be derived in the empty context with substitution* {X/f(Z)}*. Indeed, since* u ≫ p(X) *is an extension formula, this reduces to derive* p(X) *in the context formed by* u *alone. Using the second clause for* p *in* u*, this amounts to deriving* v ≫ r(X) *in* u*. Extending* u *with* v *is denoted by* vu*, and the previous goal reduces to derive* r(X) *in* vu*. If the second clause for* r *in* v *is used,* s(X) *must be derived in the same context. Since* s *is not defined in* v*,* s(X) *must be derived in the context obtained by poping* v*, that is* u*. The derivation then succeeds with the required substitution. The declarative semantics of* u *and* v *is presented in subsection 5.2.* ∎

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. 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 four 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 repetition of computations as well as infinite computations. It furthermore tackles more closely the computation steps and, therefore, makes the modelling of and-parallelism expressed before fully apparent. Technically speaking, it is based on computation histories represented as streams of actions. Repetition 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 be 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 repetitions 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 to give 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 : sets, multi-sets, complete lattices and metric spaces. 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. Section 7 sums up the relationship established in the paper and gives our conclusions. Finally, Sections 8 and 9 presents our acknowledgments and references.

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 predicate, 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 Δ letter. Clauses take here the form $H \leftarrow \overline{B}$ and allow extension formula to take place in their body. 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 $u : \mathcal{U}$, 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 $Sunit^{<\omega}$. 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. $\mathcal{M}_{co}(E)$ denotes the set of non-empty and finite[1] multi-sets (resp. the 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 equations

$$\begin{cases} X_1 = t_1 \\ \ldots \\ X_m = t_m \end{cases}$$

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.

Definition 1 *Let S be a system of equations formed from terms. A* unifier *of S is a substitution θ such that, for any equality t=u of S, the terms $t\theta$ and $u\theta$ are identical. It is a* most general unifier *of S iff it is more general than any unifier of S. The system S is said to be* unifiable *iff it has one unifier.* ∎

It is possible to relate the unification of systems of equations 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 one 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 2 *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 one 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

Complete lattices and continuous functions will be used in the characterization of the fixed-point semantics. The main definitions are briefly recalled here, together with Tarski's lemma, which guarantees the existence of a least fixed point for every continuous function of a complete lattice to itself.

[1] 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)

Definition 3 *Let L be a partially ordered set, with partial order relation* $\leq$. *L is a* complete lattice *if L has a least element* $\perp$ *and every subset X of L has a least upper bound (lub)* $\vee L$. ∎

The requirement that $\perp$ exists is redundant, since $\perp$ is the lub of the empty set $\emptyset$. Also, the definition implies that L has a greatest element and every subset has a greatest lower bound, but those facts will not be needed. The simplest example of a complete lattice is probably the powerset of a set S, $\mathcal{P}(S)$, ordered by inclusion. The fixed-point semantics of Horn clause logic is based on such complete lattices, where S is the Herbrand base. The lattices required by the semantics of CLL are more complex, and are introduced in section 5.2.

Definition 4 *Let L be a complete lattice.*

1) *A subset D of L is* directed *if D is not empty and, for every* $x, y \in D$, *there is* $z \in D$ *such that* $x \leq z$ *and* $y \leq z$.
2) *A function* $f: L \rightarrow L'$ *from L to another complete lattice L' is* continuous *if, for every directed subset D of L,* $f(D) = \{f(x) : x \in D\}_s$ *is directed and* $\vee f(D) = f(\vee D)$.
3) *A* prefixed point *of a continuous function* $f: L \rightarrow L$ *from a complete lattice L to itself is an element* $x \in L$ *such that* $f(x) \leq x$; *it is a* fixed point *if it furthermore verifies* $f(x) = x$. ∎

The main result of (order-theroretic) fixed-point theory is the celebrated Tarski's lemma.

Proposition 5 (Tarski's lemma) *Let* $f: L \rightarrow L$ *be a continuous function from a complete lattice L to itself. The set of all prefixed points (resp. fixed points) of f is a complete lattice, for the order induced by L. The least prefixed point p of f is also a fixed point, and is given by*

$$p = \bigvee_{n \geq 0} f^n(\perp)$$

where f^n *is the* n^{th} *of f.* ∎

3.4 Metric spaces

Metric spaces will also be used as important semantical tools. The reason for their introduction arise from the following fact: it is quite natural to identify the more two histories the greater their common prefix is. This observation naturally leads to a distance between them, as formalized in example 2, and therefore to metric spaces. The reader is assumed to be familiar with metric spaces as well as with their related notions of convergent sequences, closed and compact subsets, completeness, ... He is referred to [En77], when need be. For sake of completeness, we however specify hereafter some practical language misuse, describe examples of metric spaces employed subsequently and recall the notion of contraction and its very useful property (due to S. Banach) of having one and only one fixed point in complete metric spaces.

Convention 6 *We subsequently just use metric spaces whose metric is bounded by 1 i.e metric spaces (M,d) such that for all* $x, y \in M$, $d(x,y) \leq 1$. *In view of this, we refer to them more simply as metric spaces. Furthermore, their metric d is often omitted when it is clearly understood.* ∎

Example 2 *Let A be an alphabet and let* $A^{\leq\omega}$ *denote the set of all finite and infinite words over A. Let furthermore, for any* $x \in A^{\leq\omega}$, *x[n] represent the prefix of x of length n. Define the mapping*

$$d_{stream} : A^{\leq\omega} \times A^{\leq\omega} \leftarrow [0,1]$$

as follows: for any $x, y \in A^{\leq\omega}$:

$$d_{stream}(x,y) = 2^{-sup\{n : x[n]=y[n]\}_s},$$

with the convention that $2^{-\infty} = 0$. *Then,* $(A^{\leq\omega}, d_{stream})$ *is a complete metric space.* ∎

Such use of a truncation function may be extended to define a metric on multi-sets.

Example 3 *Let E be some set and let $\perp$ be an element not in E. Let futhermore*[2]

$$.[.] : E \times \mathbf{N} \rightarrow E \cup \{\perp\}_s$$

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 too. 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 complete metric space. Moreover, the spaces $(\mathcal{M}_{nf}(E),d_{ms})$ and $(\mathcal{M}_{co}(E),d_{ms})$, where compactness[3] *is taken with respect to d_E, are metric spaces. They are complete if (E,d_E) is complete.* ∎

Example 4 *Let E be some set and (M,d), (M_1,d_1), (M_2,d_2) be complete metric spaces. Define*

i) the mapping d_{funct} on the set $X \rightarrow M$ of all functions from X to M as follows : for any $f_1, f_2 \in X \rightarrow M$,

$$d_{funct}(f_1,f_2) = \sup_{x \in X} d(f_1(x), f_2(x));$$

ii) the mapping d_{cart} on the cartesian product $M_1 \times M_2$ as follows : for any $(x_1,x_2),(y_1,y_2) \in M_1 \times M_2$,

$$d_{cart}((x_1,x_2),(y_1,y_2)) = max\{d_1(x_1,y_1), d_2(x_2,y_2)\}_s.$$

Then, the spaces $(X \rightarrow M, d_{funct})$ and $(M_1 \times M_2, d_{cart})$ are complete metric spaces. ∎

Definition 7 *Let (M_1,d_1) and (M_2,d_2) be two metric spaces. A function $f : M_1 \rightarrow M_2$ is called a* contraction *iff there is a real number $c \in [0,1)$ such that, for all $x, y \in M$,*

$$d_2(f(x),f(y)) \leq c.d_1(x,y).$$ ∎

Proposition 8 (Banach's theorem) *Let (M,d) be a complete metric space. Any contraction $f : M \rightarrow M$ has a unique fixed point.* ∎

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

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

[3] 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.

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 $\mathcal{U}$, 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 (R-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.

Example 5 (Example 1 continued) *It can be shown that* $u \vdash^{bu}$ `p(f(3))` *as follows, where each line is justified by the previous one and a derivation rule:*

$u \vdash^{bu} \Delta$	(N-B)
$u \vdash^{bu}$ `s(f(3))`	(R-B)
$vu \vdash^{bu}$ `s(f(3))`	(X-B)
$vu \vdash^{bu}$ `r(f(3))`	(R-B)
$u \vdash^{bu}$ `v` $\gg$ `r(f(3))`	(E-B)
$u \vdash^{bu}$ `p(f(3))`	(R-B)

Similarly, one has $u \vdash^{bu}$ `q(f(3))`*, therefore, by (C-B), one has* $u \vdash^{bu}$ `[p(f(3)),q(f(3))]` ∎

Definition 9 *Let δ^- and δ^+ be two fresh symbols. The* bottom-up operational semantics *of CLL is the function $O_{bu} : Ssyst \to Scontext \to Sgoal \to \{\delta^-, \delta^+\}_s$ defined as follows: for any $\mathcal{U} \in Ssyst$, $c \in Scontext$ and $\overline{G} \in Sgoal$,*

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

■

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 $\overline{\mathrm{G}}$, it consists of a sequence of steps reducing $\overline{\mathrm{G}}$ to the null conjunction. Associated with it, there is a substitution θ representing the values computed for the variables of $\overline{\mathrm{G}}$. The expected result (presented in section 5) is that the universal closure of $\overline{\mathrm{G}}\theta$ is a logical consequence of $\mathcal{U}$, and that any instance of $\overline{\mathrm{G}}$ which is a consequence of $\mathcal{U}$ can be obtained as instance of $\overline{\mathrm{G}}\theta$ 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 $\overline{\mathrm{G}}$, $\mathrm{c} \vdash^{td}_{\mathcal{U}} \overline{\mathrm{G}}\ [\theta]$, or more simply $\mathrm{c} \vdash^{td} \overline{\mathrm{G}}\ [\theta]$ when $\mathcal{U}$ is understood, denotes the fact that there is a (successful) *top-down derivation of* $\overline{\mathrm{G}}$ *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

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

Conjunction

$$\text{(C-T)} \qquad \frac{\mathrm{c} \vdash^{td} \mathrm{A}_1\ [\theta_1], \ldots, \mathrm{c} \vdash^{td} \mathrm{A}_m\ [\theta_m]}{\mathrm{c} \vdash^{td} \overline{\mathrm{A}}_1, \ldots, \overline{\mathrm{A}}_m\ [\rho(\theta_1, \ldots, \theta_m)]}$$

Atomic formula—local reduction

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

Atomic formula—contextual definition

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

Extension formula

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

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. As usual, in rule (R-T) a suitable renaming of the clauses is assumed.

Example 6 (Example 1 continued) *In the following derivation, each line is justified by a derivation rule and the line following it:*

$\mathtt{u} \overset{td}{\vdash} \mathtt{p(X)}\ [\{\mathtt{X/f(Z)}\}_s]$ (R-T)
$\mathtt{u} \overset{td}{\vdash} \mathtt{v} \gg \mathtt{r(X)}\ [\{\mathtt{X/f(Z)}\}_s]$ (E-T)
$\mathtt{vu} \overset{td}{\vdash} \mathtt{r(X)}\ [\{\mathtt{X/f(Z)}\}_s]$ (R-T)
$\mathtt{vu} \overset{td}{\vdash} \mathtt{s(X)}\ [\{\mathtt{X/f(Z)}\}_s]$ (X-T)
$\mathtt{u} \overset{td}{\vdash} \mathtt{s(X)}\ [\{\mathtt{X/f(Z)}\}_s]$ (R-T)
$\mathtt{u} \overset{td}{\vdash} \Delta\ [\epsilon]$ (N-T)

Similarly, we have $\mathtt{u} \overset{td}{\vdash} \mathtt{q(X)}\ [\{\mathtt{X/f(3)}\}_s]$. *By (C-T), we obtain* $\mathtt{u} \overset{td}{\vdash} \mathtt{[p(X),q(X)]}\ [\{\mathtt{X/f(3)}\}_s]$ *since* $\{\mathtt{X/f(3),Z/3}\}_s$ *is the reconciliation of* $\{\mathtt{X/f(Z)}\}_s$ *and* $\{\mathtt{X/f(3)}\}_s$. ∎

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

Proposition 10 *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 11 *Define the* top-down operational semantics *as the following function*

$$O_{td} : Ssyst \rightarrow Scontext \rightarrow Sgoal \rightarrow \mathcal{P}(Ssubst)$$

as follows: for any $\mathcal{U} \in Ssyst$, $c \in Scontext$, *and* $\overline{G} \in Sgoal$:

$$O_{td}(\mathcal{U})(c)(\overline{G}) = \{\theta : c \overset{td}{\vdash}_{\mathcal{U}} \overline{G}\ [\theta]\}_s.$$ ∎

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

Proposition 12 *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 repetition of computations. As respective illustrations, the g-goals `p(X)` and `q(X)` are given the same semantics in the following contexts u_1 and u_2 although, on the one hand, the reduction of `p(X)` finitely fails in u_1 and is infinite in u_2, and, on the other hand, the reduction of `q(X)` computes $\{\mathtt{X/1}\}_s$ once in u_1 and twice in u_2.

```
u1 :  q(1).        u2 :  p(X) ←p(X).
                         q(1).
                         q(1).
```

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 extension (namely the two 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 13 (Histories) *The notations* unif(A,B) *and* cxt_ext(c,u) *are used to represent the actions of unifying the atoms A and B and of extending the context c by the unit u, 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 14 (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 15 (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

$$(R\text{-}H) \qquad \frac{}{A \text{ in } < \sigma, c > \xrightarrow{l} \overline{B} \text{ in } < \sigma^*, c >} \quad \text{if} \quad \begin{cases} (H \leftarrow \overline{B}) \in |u|^4 \\ A\sigma \text{ and } H \text{ are unifiable} \\ \sigma^* = \sigma \circ mgu(A\sigma, H) \\ l = \{unif(A\sigma, H)\}_{ms} \end{cases}$$

Atomic formula—contextual definition

$$(X\text{-}H) \qquad \frac{A \text{ in } < \sigma, c > \xrightarrow{l} \widetilde{B} \text{ in } < \sigma^*, c >}{A \text{ in } < \sigma, uc > \xrightarrow{l} \widetilde{B} \text{in } < \sigma^*, uc >} \quad \text{if} \quad name(A) \notin sort(u)$$

Extension formula

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

Conjunction

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

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

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

■

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

Rules (R-H), (X-H) and (E-H) essentially rephrase the rules (R-T), (X-T) and (E-T) defining how atoms and extension formulas should be treated. Rules (C-H_1), (C-H_2) and (C-H_3) defines 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).

The following property establishes that the transition relation is image finite.

Proposition 16 *For any $\widetilde{A} \in Sextgoal$, the multi-set*

$$\{(\widetilde{B}, l) \in Sextgoal \times \mathcal{M}_{nf}(Sact) : \widetilde{A} \xrightarrow{l} \widetilde{B}\}_{ms}$$

is finite. ∎

We are now in position to define the operational semantics O_{ch}. Note that the image-finiteness of Proposition 16 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 17

1) Define $\mathbf{O} : Ssyst \rightarrow Sextgoal \rightarrow \mathcal{M}_{co}(Shist)$ *as follows : for any $\mathcal{U} \in Ssyst$, any $\widetilde{G} \in Sextgoal$,*

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

2) Define the computational history operational semantics *as the following function :*

$$O_{ch} : Ssyst \rightarrow Scontext \rightarrow Ssubst \rightarrow Sgoal \rightarrow \mathcal{M}_{co}(Shist)$$

for any $\mathcal{U} \in Ssyst$, any $c \in Scontext$, any $\sigma \in Ssubst$, any $\overline{G} \in Sgoals$,

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

Example 7 (Example 1 continued) *As an illustration, the operational semantics*

$$O_{ch}(\mathcal{U})(\mathtt{u})(\{\mathtt{Y/X}\}_s)(\mathtt{p(Y)},\mathtt{q(Y)})$$

includes[5]

i) $\{unif(\mathtt{p(X)},\mathtt{p(X_1)}), unif(\mathtt{q(X)},\mathtt{q(f(3))})\}_{ms}.\{cxt\_ext(\mathtt{u},\mathtt{v})\}_{ms}$
$.\{unif(\mathtt{r(X_1)},\mathtt{r(X_2)})\}_{ms}.\{unif(\mathtt{s(X_2)},\mathtt{s(f(Z_1))})\}_{ms}$,

ii) $\{unif(\mathtt{p(X)},\mathtt{p(X_1)})\}_{ms}.\{cxt\_ext(\mathtt{u},\mathtt{v})\}_{ms}.\{unif(\mathtt{r(X_1)},\mathtt{r(X_2)})\}_{ms}$
$.\{unif(\mathtt{q(X)},\mathtt{q(f(3))})\}_{ms}.\{unif(\mathtt{s(X_2)},\mathtt{s(f(Z_1))})\}_{ms}$ ∎

It is worth noting that the auxiliary function **O** is 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.

[5]The variables $\mathtt{X_1}$, $\mathtt{X_2}$ and $\mathtt{Z_1}$ indicate renaming of the variables of the clauses.

Definition 18 *Define*

$$\Psi_{op} : [Ssyst \rightarrow Sextgoal \rightarrow \mathcal{M}_{co}(Shist)] \rightarrow [Ssyst \rightarrow Sextgoal \rightarrow \mathcal{M}(Shist)]$$

as follows: for any $F \in [Ssyst \rightarrow Sextgoal \rightarrow \mathcal{M}_{co}(Shist)]$, *any* $\mathcal{U} \in Ssyst$, *any* $\widetilde{G} \in Sextgoal$,

$$\begin{aligned}\Psi_{op}(\mathcal{U})(F)(\widetilde{G}) \;=\; & \{\delta^- : \widetilde{G} \neq \Delta_{ext}, \widetilde{G} \not\rightarrow \}_{ms} \\ & \cup_{ms}\{\delta^+ : \widetilde{G} = \Delta_{ext}\}_{ms} \\ & \cup_{ms}\{l.h : \widetilde{G} \xrightarrow{l} \widetilde{G}^*, h \in F(\mathcal{U})(\widetilde{G}^*)\}_{ms}\end{aligned}$$

■

Proposition 19

1) *The function* Ψ_{op} *is a contraction from* $[Ssyst \rightarrow Sextgoal \rightarrow \mathcal{M}_{co}(Shist)]$ *to* $[Ssyst \rightarrow Sextgoal \rightarrow \mathcal{M}_{co}(Shist)]$.
2) *The function* **O** *is a fixed point of* Ψ_{op}.

■

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 reconcile the results. This is more precisely achieved by means of three functions. The eq_act function translates each basic action into an equivalent unification equation. The syst_hist function returns the system of equations corresponding to unification actions of successful histories and the empty system for the others. The function α_2 solves the systems of equations corresponding to the histories of any given multi-set of $\mathcal{M}_{co}$(Shist). Relating O_{td} and O_{ch} then consists of noting, by an inductive reasoning on reduction 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.

Definition 20

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

$$eq\_label(l) = \begin{cases} \{A = B\}_{ms} & \text{if } l = unif(A,B) \\ \emptyset & \text{if } l = cxt\_ext(c,u) \end{cases}.$$

2) *Define* syst_hist *as follows: for any* $h \in Shist$,

$$syst\_hist(h) = \begin{cases} \cup_{\substack{ms \\ i=1,\ldots,m}} \; \cup_{\substack{s \\ a \in l_i}} eq\_act(a) & \text{if } h = l_1 \ldots . l_m.\delta^+ \\ \emptyset & \text{if } h = l_1 \ldots . l_m.\delta^+ \text{ or if } h \text{ is infinite} \end{cases}.$$

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

$$\alpha_2(MH) = \cup_{\substack{s \\ h \in MH}} \{mgu\_syst(syst\_hist(h))\}_s.$$

■

Proposition 21 *For any* $\mathcal{U} \in Ssyst$, *any* $c \in Scontext$, $\overline{G} \in Sgoal$, *one has*

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

■

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 22 *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 $S \downarrow [P] = \{ p(t_1,\ldots,t_n) \in S : p \in P\}$. To simplify the notation, $S \downarrow [Pred\text{-}P]$ is abbreviated to $S \downarrow [\text{-}P]$. Note that $S = (S \downarrow [P]) \cup (S \downarrow [\text{-}P])$.* ■

Definition 23 *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: *$t(S) \downarrow [\text{-}P] = S \downarrow [\text{-}P]$ (atoms with names not in P are preserved by t).*
ii) Dependency: *$t(S) = t(S \downarrow [\text{-}P])$ (the update depends only on the preserved atoms).* ■

Definition 24 *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 25 *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 $S \models_I \{f\}$.*

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 26 *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 27 *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 and to prove the equivalence between the top-down operational semantics O_{td} and the declarative semantics Decl_m. Along the way other interesting results are presented, relating the bottom-up and the top-down derivation relations with the consequence relation.

The first result states that the set of all interpretations is a complete lattice, as required.

Proposition 28 *The set Sint of all interpretations $I = (I_u)_{u \in \mathcal{U}}$, 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 29 *Let $T_{\mathcal{U}}$: Sint $\rightarrow$ Sint (or, more simply, T : Sint $\rightarrow$ Sint) be the mapping defined as follows : for every interpretation I, $T_{\mathcal{U}}(I)$ is the interpretation J such that*

$$J_u(S) = (S \downarrow [\text{-}sort(u)]) \cup \{A : \text{ there exists } (A \leftarrow \overline{B}) \in ground(|u|) \text{ such that } I_u(S) \models_I \overline{B}\}_s$$

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

The mapping T is well defined, as shown by the next proposition.

Proposition 30

1) *T(I) is an interpretation, for every interpretation I.*
2) *T is continuous.*
3) *An interpretation I is a model of $\mathcal{U}$ if and only if $T(I) \leq I$* ∎

An interpretation I such that $T(I) \leq I$ is sometimes kown as a "prefixed" point of T. By Tarski's lemma (see proposition 5), T has a least fixed point, which by the previous property is also the least model of $\mathcal{U}$. This statement is the contents of the next proposition.

Proposition 31 *Every system of units $\mathcal{U}$ has a minimal model $M_{\mathcal{U}}$, given as the least fixed point of $T_{\mathcal{U}}$.* ∎

Example 8 (Example 1 continued) *The fixed-point construction is now illustrated with example 1. For readability, M_{u} and M_{v} corresponding to the minimal model M are denoted more simply by* u *and* v*, respectively. By definition of the operator $T_{\mathcal{U}}$, of which* (u,v) *is of the least fixed point,* u *and* v *satisfy the following equations, for every $S \subseteq B$:*

$$\begin{aligned} \mathrm{u}(S) &= S \downarrow [\text{-p,q,s}] \cup_s \{\mathtt{p(1)},\mathtt{q(f(3))}\}_s \cup_s \{\mathtt{s(f(x))} : x \in \{1,2,3\}_s\}_s \\ &\quad \cup_s \{\mathtt{p(x)} : \mathrm{u}(S) \models \mathrm{v} \gg \mathtt{r(x)}\}_s \\ \mathrm{v}(S) &= (S \downarrow [\text{-r}]) \cup_s \{\mathtt{r(2)}\}_s \cup_s \{\mathtt{r(x)} : \mathrm{v}(S) \models \mathtt{s(x)}\}_s \end{aligned}$$

To "solve" them, let us first note that the condition $\mathrm{v}(S) \models \mathtt{s(x)}$ is equivalent to $\mathtt{s(x)} \in \mathrm{v}(S)$, and therefore to $\mathtt{s(x)} \in S$, by the form of $\mathrm{v}(S)$. One may thus write

$$\mathrm{v}(S) = (S \downarrow [\text{-r}]) \cup_s \{\mathtt{r(2)}\}_s \cup_s \{\mathtt{r(x)} : \mathtt{s(x)} \in S\}_s$$

Now $\mathrm{u}(S) \models \mathrm{v} \gg \mathtt{r(x)}$ if and only if $\mathtt{r(x)} \in \mathrm{v}(\mathrm{u}(S))$. By the form of $\mathrm{v}(\mathrm{u}(S))$, this is equivalent to $\mathtt{x} = 2$ or $\mathtt{s(x)} \in \mathrm{u}(S)$, that is, by the form of $\mathrm{u}(S)$, $\mathtt{x} \in \{2, \mathtt{f(1)}, \mathtt{f(2)}, \mathtt{f(3)}\}_s$. Thus one has

$$\begin{aligned} \mathrm{u}(S) &= (S \downarrow [\text{-}\{\mathtt{p,q,s}\}_s]) \cup_s \{\mathtt{p(1)},\mathtt{p(2)},\mathtt{q(f(3))}\}_s \\ &\quad \cup_s \{\mathtt{s(f(x))} : \mathtt{x} \in \{1,2,3\}_s\}_s \cup_s \{\mathtt{p(f(x))} : \mathtt{x} \in \{1,2,3\}_s\}_s \end{aligned}$$ ∎

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

Definition 32 *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 33 *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 34 *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 35 *Given a system of units* $\mathcal{U}$ *with minimal model* M, $c \vdash^{bu}_{\mathcal{U}} \overline{G}$ *if and only if* $S_M(c) \models_M \overline{G}$ *for every context name* c *and g-goal* $\overline{G}$. ∎

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

Proposition 36 *One has* $O_{bu} = \alpha_1 \circ Decl_m$. ∎

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

Proposition 37 *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 38 *Let* $\alpha_3 : \mathcal{P}(Ssubst) \rightarrow \mathcal{P}(Ssubst)$ *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 is defined compositionally and on the basis of processes, organised in tree-like structures. It makes no use of transition systems as well as no reference to any declarative paradigm. It is called denotational in view of these properties.

Processes introduced in this section are intended to capture at a conceptual level (thus not at the implementation one) what could be roughly called the behavior of goals. Such a behavior is here essentially seen as a sequence of computation steps, each one described by an input multi-set of substitutions, a multi- set of basic actions[6] and an output multi-set of substitutions. It involves trees rather than streams in order to further capture the place of the various choice of uses of clauses. Hence, as agreed by our intuitive perception of the computations, the reduction of `p(a,X)` behaves differently in the following units u_1 and u_2. Note that the g-goal `p(a,X)` however receives the same operational and declarative semantics.

```
u1 :  p(a,Y) ← q(Y).        u2 :  p(a,Y) ← r(Y).
      q(b).                       p(a,Y) ← s(Y).
      q(c).                       r(b).
                                  s(c).
```

Tree-like structures with possible repetition of alternative branches may be formalized by means of multi-sets. Processes may then be formally described as multi-sets of computation steps followed by new processes. Two auxiliary processes, $p^+$and p^-, are furthermore introduced to indicate termination in the successful and failure status. In view of them, it may be ensured that any choice (resulting from alternative use of unifiable clauses) is always composed of at least one alternative. It is also composed of a finite number of alternatives since programs are composed of a finite number of clauses. Multi-sets representing choices may thus be assumed non-empty and finite. It is also easy to ensure the non-emptyness and finiteness of the multi-sets of computation steps since computations always start with one substitution and perform, at each step, only a non-empty and finite number of elementary reduction steps. Summing up, 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[7] 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)

Definition 39 *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, reflecting the property that the closer two processes are the bigger their common prefix is, can be achieved by considering the union of the equation (1) as disjoint and by defining the truncation function .[.] on processes.

Definition 40 *Define the truncation function .[.] on Sfproc by the following rules:*

[6] Recall notation 13 of section 4.3

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

i) $p[0] = \bot$, *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 41 *Define the metric on Sfproc as follows:*
i) $d(p^+, p^+) = d(p^-, p^-) = 0$,
ii) $d(p^-, p) = 1$, *for any* $p \in Sfproc$, *distinct from* p^-,
iii) $d(p^+, p) = 1$, *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 process 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).

Definition 42 *Define Sproc as the completion of (Sfproc,d).* ■

Proposition 43 *Any element of Sproc is either* p^+, p^- *or of the form* $\{(\omega_1, p_1), \ldots, (\omega_m, p_m)\}_{ms}$ *with, for any i,* $p_i \in Sproc$ *and* $\omega_i \in Scstep$. *Furthermore, in the last case, if, for any i,* ω_i *is rewritten as* $(\Lambda_i, S_i, \Upsilon_i)$, *then the* ω_i*'s verify the equalities* $\Lambda_1 = \ldots = \Lambda_m$. ■

Definition 44 *Define, for any process* $p = \{(\omega_1, p_1), \ldots, (\omega_m, p_m)\}_{ms}$, init(p) *as the common value of the first component of the* ω_i*'s.* ■

Example 9 (Example 1 continued) *Returning to example 1, an example of a process is*

$$\{(\omega_1, \{(\omega_2, \{(\omega_3, \{(\omega_4, p^+)\}_{ms})\}_{ms})\}_{ms})\}_{ms}$$

where

$$\begin{aligned}
\omega_1 &= (\{\sigma_1, \sigma_1\}_{ms}, \{\mathit{unif}(\mathtt{p(X)}, \mathtt{p(X)}_1), \mathit{unif}(\mathtt{q(X)}, \mathtt{q(f(3))})\}_{ms}\{\sigma_2, \sigma_3\}_{ms}) \\
\omega_2 &= (\{\sigma_2\}_{ms}, \{\mathit{cxt_ext}(\mathtt{u}, \mathtt{v})\}_{ms}, \{\sigma_2\}_{ms}), \\
\omega_3 &= (\{\sigma_2\}_{ms}, \{\mathit{unif}(\mathtt{r(X)}, \mathtt{r(X}_2\mathtt{)})\}_{ms}, \{\sigma_4\}_{ms}), \\
\omega_4 &= (\{\sigma_4\}_{ms}, \{\mathit{unif}(\mathtt{s(X)}_2, \mathtt{s(f(Z))})\}_{ms}, \{\sigma_5\}_{ms}),
\end{aligned}$$

$$\begin{aligned}
\sigma_1 &= \{\mathtt{Y/X}\}_s, \\
\sigma_2 &= \{\mathtt{X/X}_1, \mathtt{Y/X}_1\}_s, \\
\sigma_3 &= \{\mathtt{X/f(3)}, \mathtt{Y/f(3)}\}_s, \\
\sigma_4 &= \{\mathtt{X/X}_2, \mathtt{Y/X}_2, \mathtt{X}_1\mathtt{/X}_2\}_s, \\
\sigma_5 &= \{\mathtt{X/f(Z}_1\mathtt{)}, \mathtt{Y/f(Z}_1\mathtt{)}, \mathtt{X}_1\mathtt{/f(Z}_1\mathtt{)}, \mathtt{X}_2\mathtt{/f(Z}_1\mathtt{)}\}_s
\end{aligned}$$
■

Defining a compositional semantics requires to define an operator $\tilde{\|}$ equivalent at the denotational level to the parallel composition operator "," between g- atoms. According to our modelling of parallel composition of section 4.3 and to our denotational choice concern, it should be such that, given two processes p_1 and p_2, $p_1 \tilde{\|} p_2$ is the process interleaving some steps of p_1 and p_2, performing some others simultaneously and conserving their choice points. A recursive definition for $p_1 \tilde{\|} p_2$ might be suggested by the recursive nature of the processes. The possible infinite nature of p_1 and p_2 makes it however incorrectly stated. We circumvent this problem by using a higher-order function $\Psi_{\|}$, of the same recursive nature but that turns out to be a well-defined contraction. The operator $\tilde{\|}$ is then defined as its fixed point. Miming the "," operator, it is also associative.

Definition 45 *Define* $\Psi_{\|} : [Sproc \times Sproc \rightarrow Sproc] \rightarrow [Sproc \times Sproc \rightarrow Sproc]$ *as follows: for any* $F \in [Sproc \times Sproc \rightarrow Sproc]$, *for any* $p \in Sproc$, *for any* $p_1, p_2 \in Sproc$ *distinct from* p^+ *and* p^-:

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

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

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

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

■

Proposition 46 *The function* $\Psi_{\|}$ *is well-defined and is a contraction.* ■

Definition 47 *Define the function* $\tilde{\|} : Sproc \times Sproc \rightarrow Sproc$ *as the fixed point of* $\Psi_{\|}$. ■

Proposition 48 *For any* $p_1, p_2, p_3 \in Sproc$, *one has* $(p_1 \tilde{\|} p_2) \tilde{\|} p_3 = p_1 \tilde{\|} (p_2 \tilde{\|} p_3)$ ■

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 49 *Define* $\Psi_{den} : [Ssyst \rightarrow Sextgoal \rightarrow Sproc] \rightarrow [Ssyst \rightarrow Sextgoal \rightarrow Sproc]$ *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);*
- $H_1 \leftarrow \overline{B_1}, \ldots, H_m \leftarrow \overline{B_m}$ *are all the clauses of u unifiable with* $A\sigma$, *say with (idempotent) mgu* $\theta_1, \ldots, \theta_m$, *respectively,*
- $\omega_i = (\{\sigma\}_{ms}, \{unif(A\sigma, H_i)\}_{ms}, \{\sigma \circ \theta_i\}_{ms})$, *for all i,*
- $\widetilde{B_i} = \overline{B_i}$ in $< \sigma \circ \theta_i, c >$

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

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

■

Proposition 50 *The function* Ψ_{den} *is well-defined and is a contraction.* ■

Definition 51

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* $\overline{G} \in Sgoal$,

$$Den(\mathcal{U})(c)(\sigma)(\overline{G}) = \mathbf{D}(\mathcal{U})(\overline{G} \text{ in } < \sigma, c >).$$

■

Example 10 (Example 1 continued) *The process of example 9 is an element of* $Den(\mathcal{U})(u)(\{\mathtt{Y}/\mathtt{X}\}_s)(\mathtt{p}(\mathtt{Y}),\mathtt{q}(\mathtt{Y}))$. ■

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. It is defined as the fixed point of a suitable contraction in order to handle correctly infinite structures.

Definition 52 *Define*

$$\Psi_{stream} : [Sproc \rightarrow \mathcal{M}_{co}(Shist)] \rightarrow [Sproc \rightarrow \mathcal{M}_{co}(Shist)]$$

as follows: for any $F \in [Sproc \rightarrow \mathcal{M}_{co}(Shist)]$,

i) $\Psi_{stream}(F)(p^-) = \{\delta^-\}_{ms}$

ii) $\Psi_{stream}(F)(p^+) = \{\delta^+\}_{ms}$

iii) $\Psi_{stream}(F)(\{((\Lambda, S, \Upsilon), p)\}_{ms}) = \{S.\alpha : \alpha \in F(p)\}_{ms}$

iv) $\Psi_{stream}(F)(\{(\omega_1, p_1), \ldots, (\omega_m, p_m)\}_{ms}$
$= \Psi_{stream}(F)(\{(\omega_1, p_1)\}_{ms} \cup_{ms} \ldots \cup_{ms} \Psi_{stream}(F)(\{(\omega_m, p_m)\}_{ms})$ ■

Proposition 53 *The function* Ψ_{stream} *is well-defined and is a contraction.* ■

Definition 54 *Define* $\alpha_4 : Sproc \rightarrow \mathcal{M}_{co}(Shist)$ *as the fixed point of* Ψ_{stream}. ■

Relating **O** and **D** simply consists of proving that the function

$$\alpha_4 \hat{\circ} \mathbf{D} : \text{Ssyst} \rightarrow \text{Sextgoal} \rightarrow \mathcal{M}_{\text{CO}}(\text{Shist})$$

defined as

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

is a fixed point of Ψ_{op}. Indeed, as **O** is also a fixed point of Ψ_{op}(see proposition 19) and since contractions have only one fixed point, the functions **O** and $\alpha_4 \hat{\circ} \mathbf{D}$ are then proved identical. Hence, using the corresponding restrictions, one has

$$\text{O}_{ch} = \alpha_4 \hat{\circ} \text{Den}$$

(the function $\alpha_4 \hat{\circ}$Den is defined similarly to $\alpha_4 \hat{\circ} \mathbf{D}$). Such a fixed-point and equality properties may indeed be proved, as claimed by the following proposition.

Proposition 55

1) The function $\alpha_4 \hat{\circ} \mathbf{D}$ *is a fixed point of* Ψ_{op}.

2) 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. One contribution of this paper is precisely to show how the classical logical semantical framework can be extended to contextual logic programming and, in particular, to the parallel version studied in this paper, including or-parallelism and and-parallelism.

More precisely, the main innovations with respect to these semantics studies are as follows. Firstly, there is the use of derivation relations instead of the notion of derivation to characterize the operational semantics. Secondly, a bottom-up derivation relation has been introduced, to help grasp the declarative meaning of clauses, and to simplify the proof of the equivalence between the operational and the declarative semantics. And finally, the notion of interpretation for Horn clause logic has been appropriately generalized, in order to take into account the context-dependency of the truth of formulae. An effort has thus been made to keep these semantics as simple as possible, and well in the mainstream of logic programming semantics. Context-dependency, however, raises new problems, requiring fresh solutions. The approach followed in this paper consisted of modelling the "information content" of contexts by situations, and units by situation updates. It is hoped that this approach can be adapted to other cases, where different kinds of situation transformations can be used to represent different kinds of context extension.

The language CLL has a certain similarity with the language proposed by Miller ([M86], [M89]), which is based on the deduction theorem of the predicate calculus. There are however two main differences between the two languages. The context mechanism in Miller's language consists of adding new clauses to the clauses in the context. In CLL, the clauses are kept separated, and new predicates are added instead, overriding all existing predicates with the same name. The use of clauses instead of predicates complicates enormously the semantics, as can be appreciated in the semantics put forward by Miller.

The two other semantics are issued from the imperative tradition, and, more 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. With respect to these semantics, our contribution is fourfold. First, the parallel contextual logic programming framework is tackled; this requires new solutions that were not addressed before. Second, parallel computations have been modelled in a very realistic manner: our perception includes both interleaving and true concurrency. Restated in other terms, in order to allow a parallel compound statement to proceed, it is sufficient that one subcomponent performs one step although, all of them are allowed to do so. Third, repetition of computations has been taken into account. Fourth, use of local states and of reconciliation to combine them has allowed to express the denotational semantics very simply: processes are here not composed of full functions but of quite intuitive computation steps: input substitutions, actions and output substitutions.

All these semantics have been related throughout the paper, thanks to propositions 12, 21, 33, 36, 38, 55. They are summed up in Figure 1.

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 10, 35, 37 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 1 can be completed by figure 2.

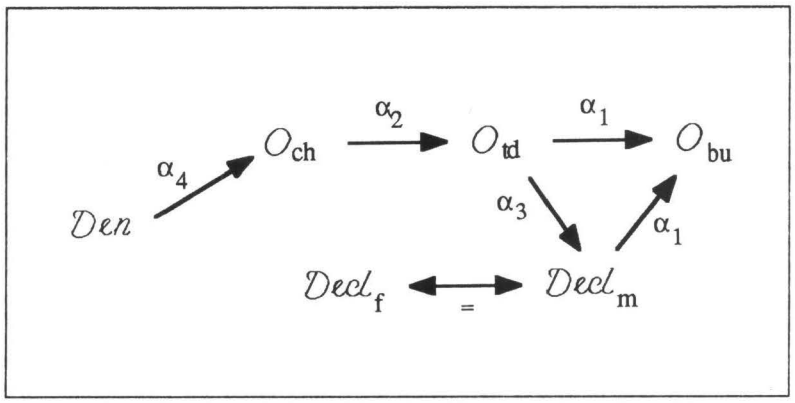

Figure 1: The minimal relations

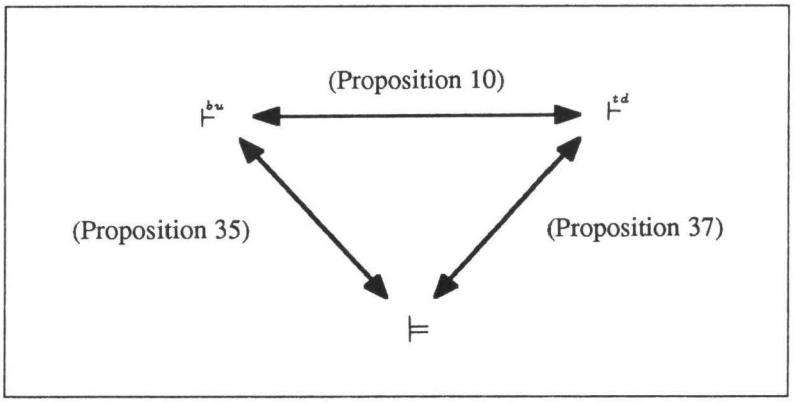

Figure 2: Equivalence of the bottom-up derivation, top-down derivation and satisfaction relation

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]).

8 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.

9 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.*

[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.