Extended Horn Clauses: the Framework and some Semantics[1]

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

Abstract

The purpose of this paper is twofold: to introduce a new extension of concurrent logic programming languages aiming at handling synchronicity and to present and compare several semantics for it. The extended framework essentially rests on an extension of Horn clauses, including multiple atoms in their heads and a guard construct, as well as a new operator between goals. The semantics discussed consist of four semantics. They range in the operational, declarative and denotational types and are issued both from the logic programming tradition and the imperative tradition. They are composed of an operational semantics, describing the (classical) success set and failure set, of two declarative semantics, extending the Herbrand interpretation and the immediate consequence operator to the extended framework, and of a denotational semantics, defined compositionally and on the basis of histories possibly involving hypothetical statements. The mathematical tools mainly used are complete lattices and complete metric spaces.

1 Introduction

So-called or-parallelism and and-parallelism are the two main ways of introducing parallel executions in logic programming. Basically, the former consists of reducing an atom by using all unifiable clauses in parallel and by reducing concurrently the induced instances of the clause bodies. The latter consists of reducing a conjunction of atoms by reducing all atoms in parallel. In that framework, communication between concurrent reductions is achieved by means of the sharing of variables between several conjoined atoms. It is often further ruled by suspension mechanisms that force the reduction of some subgoals to wait until the reduction of other subgoals has sufficiently instantiated the shared variables. Examples of such mechanisms are Concurrent Prolog read-only annotations ([20]), Parlog mode declarations ([12]) and GHC suspension rules ([21]). As pointed out in [7], a form of asynchronous communication results. In most classical logic programming languages (e.g. Concurrent Prolog, Parlog, GHC, cc languages ([19]), ...), there is however no other means to tackle synchronous communication than that of coding it by means of auxiliary manager procedures and of asynchronous communication. This paper investigates a way of introducing synchronous communication directly. For that purpose, Horn clauses are extended in so-called extended Horn clauses and the SLD-resolution principle is extended accordingly. The aim of this paper is to sketch the resulting framework as well as to present and compare various semantics for it.

As a snapshot, the extended Horn clauses take the form

$$H_1 \diamond \cdots \diamond H_m \leftarrow G \mid G_1 \diamond \cdots \diamond G_m$$

where $H_1, \ldots, H_m$ are atoms and G, $G_1, \ldots, G_m$ are conjunctions of atoms combined with the operators " ; ", " $\|$ " and " & ". All atoms may share variables. Compared with classical Horn clauses, the main innovations are thus

i) the presence of multiple atoms in the head of a clause,
ii) the presence of a special goal G,
iii) the possibility to combine atoms with several operators to form goals.

[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

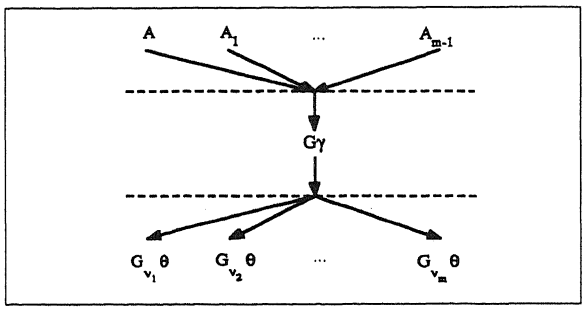

Figure 1: Synchronized reductions with extended Horn clauses

Particularly notice that the number of head atoms H_i equals the number of conjunctions G_i.

These extensions induce an extension of the SLD-resolution rule. Basically, the conjunction G acts as an additional test to the usual unification one: in order to use a clause for reduction, the instantiation of its G part by the corresponding mgu should in fact be completely reduced and this in isolation i.e. independently of concurrent processes.

The operators " ; " and " $\|$ " are used for sequential and parallel compositions, respectively. The operators " & " and " $\diamond$ " are employed, in a dual way, to specify synchronization. The operator " & " acts at the goal level and forces the reduction of conjuncts to be performed simultaneously. In a dual manner, the operator " $\diamond$ " acts at the clause level and forces the reduction of an atom A to wait for the presence of other (concurrent) atoms $A_1, \ldots, A_{m-1}$ such that the m-tuple $< A, A_1, \cdots, A_{m-1} >$ unifies with one permutation of the m-tuple $< H_1, \cdots, H_m >$, say $< H_{\nu_1}, \cdots, H_{\nu_m} >$. In that case, assuming the induced instance of G can be reduced successfully, say with the computed answer substitution θ, all atoms are simultaneously reduced to the instances by θ of the corresponding G_{ν_i}'s. This is schematized in figure 1.

Actually, the reduction possibilities are even richer in that it is allowed to group several clauses, previously renamed to avoid variable clashes, say

$$(L_1 \diamond \cdots \diamond L_p \leftarrow A \mid S_1 \diamond \cdots \diamond S_p), \cdots, (M_1 \diamond \cdots \diamond M_q \leftarrow B \mid T_1 \diamond \cdots \diamond T_q),$$

to form a clause

$$L_1 \diamond \cdots \diamond L_p \diamond \cdots \diamond M_1 \diamond \cdots \diamond M_q \leftarrow (A \parallel \cdots \parallel B) \mid S_1 \diamond \cdots \diamond S_p \diamond \cdots \diamond T_1 \diamond \cdots \diamond T_q$$

to consider in the same right as the one above.

Though simple, this extension to the classical logic programming framework is quite suited for handling synchronicity in logic programming. This fact is advocated in section 2. It is also shown that, as a side effect, extended Horn clauses provide a nice way of describing communication between objects and, hence, constitutes a means towards the integration of logic programming and object-oriented programming.

This paper also describes several semantics of extended Horn clauses, precisely of the concurrent language induced by the and-parallelism, the or-parallelism and the above operators. Four semantics are presented. They are composed of one operational semantics O_d, two declarative semantics, $Decl_m$ and $Decl_f$, and one denotational semantics Den. The three first ones take place in the logic programming tradition. The latter is issued from the imperative tradition, especially from its metric branch.

The operational semantics O_d rests on a derivation relation. It describes the derivations 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 conjoined 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, extended Horn clauses and synchronized executions raise new problems, for which fresh solutions are proposed.

The denotational semantics Den, defined as usual compositionally, completes the previous semantics by describing the behavior of compound goals in a closer way, including the modelling of parallelism just exposed, and by distinguishing various sources of failure: failure induced by the absence of suitable clauses (real failure), failure induced by infinite computations and failure induced by the absence of suitable concurrent goals that would allow synchronization to take place (suspension). In particular, the latter point is tackled by handling suitable hypotheses about the environment of goals.

Extended Horn clauses have already been presented in similar forms in [2], [3], [4], [8], [10], [16], [17] and [19]. The work reported here differs from them both from the language point of view and from the semantic point of view.

From the language point of view, our language differs in three main respects.

Firstly, it allows *arbitrary* sequential and parallel compositions inside goals as well as an unrestricted form of variable sharing. In particular, the duality of the expression of the synchronization in the goals and in the clause is peculiar to our work. In contrast with [2], we do not allow a forking primitive to take place in the body of clauses. However, this can be achieved easily in our model through or-parallelism.

Secondly, a notion of guard has been introduced; it is not present in any other work.

Thirdly, clauses always have the same number of heads and bodies. The reason for this requirement is that the reduction of a head by the corresponding body is seen as one step in the execution of the process corresponding to the head. As each process must have a continuation, even if to terminate, the continuation is represented by the corresponding body. It should be noted that this requirement, besides allowing to deal with unrestricted sequential composition, does not represent a real limitation as compared to the aforementioned languages. For example, the clauses $A_1 + A_2 \leftarrow A_3$ and $A_1 + A_2 \leftarrow A_3 + A_4 + A_5$ of Rose ([3]) may be rewritten respectively as $A_1 \diamond A_2 \leftarrow \Delta \mid A_3 \diamond \Delta$ and $A_1 \diamond A_2 \leftarrow \Delta \mid A_3 \diamond A_4 \parallel A_5$, with Δ denoting the empty conjunction of atoms.

From the semantic point of view, our work differs both from related work issued from the logic programming tradition and from the metric imperative tradition. To our best knowledge, semantics for extended Horn clauses have only been proposed in [2], [3], [10] and [16].

The semantics presented in [2] essentially refers to a new logic, called linear logic ([11]). It thus differs from our declarative and metric-based semantics.

In [3] and [10], the study of the declarative semantics is also conducted in terms of an extension of the Herbrand base containing parallel goals. Those goals, in the absence of a sequential composition operator, are parallel compositions of atomic formulae. By contrast, the extended Herbrand base appropriate to our language must consider parallel compositions of *arbitrary* goals. Another technical difference with our approach is our systematic use of t-contexts as an auxiliary tool in the definitions of both the operational and the declarative semantics. The main reason for introducing t-contexts was the need to find a concise way to specify the selection of atomic formulae in goals and their replacement by other goals. As can be appreciated from our semantic study, the use of t-contexts greatly simplifies the presentation of the semantic concepts of derivability and satisfiability. The declarative semantics presented here is also a clarified version of that presented in [16].

The operational semantics O_d differs from that of [3] and [10] by the use of the notion of t-context. It differs from [16] by the use of a semantic variant of the considered program P that allows several independent reductions to occur at the same time. The denotational semantics Den has no counterpart in [3], [10] and [16]. Although it is of classical metric inspiration, it still presents some originality with related work ([6], [5], [14], ...) which arises essentially from the two following points :

i) our concern with extended Horn clauses, which has not been done before and which requires new solutions; in particular, it should be noticed that the form of communication provided by the " $\diamond$ " and " & " operators is different from the monotonic asynchronous one of concurrent logic programming languages and from the synchronous one of CCS and CSP;

ii) our use of local states and of reconciliation to combine them.

Finally, the comparative study of semantics for extended Horn clauses issued both from the logic programming and from the imperative programming traditions is peculiar to our work.

The semantic tools mainly used in this paper are of two types: complete lattices and complete metric

spaces. Despite this variety, the semantics have been related throughout the paper.

The remainder of this paper is organized into 9 sections. Section 2 suggests the interest of extended Horn clauses through the coding of various producer/consumer schemes and of several examples integrating the logic and object-oriented styles of programming. Section 3 describes the basic constructs of the language and explains our terminology. Section 4 recalls the basic semantic tools used in the paper. Section 5 introduces a semantic translation simplifying the presentation of the semantics. Section 6 defines the auxiliary concepts of t-context and program completion. Section 7 presents the operational semantics O_d. Section 8 discusses the declarative models $Decl_m$ and $Decl_f$ and connects them with the operational semantics O_d. Section 9 specifies the denotational semantics Den and compares it with the operational semantics O_d and, consequently, in view of previous results, to the other semantics. Finally, section 10 sums up the relationships established in the paper and gives our conclusions.

2 Examples

2.1 Producer-consumer schemes

As first examples of the expressiveness power of extended Horn clauses, let us code, by using them, synchronous communication in various producer/consumer schemes. Assume we are given a producer, say `prod`, and a consumer, say `cons`, behaving successively as follows:

i) execute some internal actions, say `int_prod(M,X)` and `int_cons(Y)`, respectively; the former producing some message M;
ii) communicate synchronously the message M and treat it;
iii) apply some (undefined) resumption actions, say `prod_res(M,U)` and `cons_res(M,V)`, respectively.

As can be deduced from our sketchy description of section 1, this behavior can be simulated by the evaluation of the query `prod || cons` for the program[4]

```
prod ← int_prod(M,X) ; pexch(M)
cons ← int_cons(Y) ; cexch(M)
pexch(M) ◇ cexch(M) ← treat(M) | prod_res(M,U) ◇ cons_red(M,V)
```

Indeed, the parallel composition " || " makes the atoms `prod` and `cons` reduce concurrently. This is achieved by means of the first and the second clauses, respectively. As a result, the two atoms are reduced to the sequential compositions $\texttt{int\_prod}(M_1,X_1)$; $\texttt{pexch}(M_1)$ and $\texttt{int\_cons}(Y_2)$; $\texttt{cexch}(M_2)$, respectively, with M_1, M_2, X_1, Y_2 renamings of the variables M, X and Y. The reduction of the first conjunction consists of reducing $\texttt{int\_prod}(M_1,X_1)$, which is not defined by the above program segment but is assumed to instantiate M_1, and then of reducing the induced instance of $\texttt{pexch}(M_1)$. Similarly, the reduction of the second conjunction consists of reducing $\texttt{int\_cons}(Y_2)$, which is undefined here too, and then of reducing $\texttt{cexch}(M_2)$. Because of the parallel composition of `prod` and `cons`, the reductions of $\texttt{int\_prod}(M_1,X_1)$ and of $\texttt{int\_cons}(Y_2)$ can be performed in any order. However, because of the extended form of the third clause, the reduction of $\texttt{pexch}(M_1)$ can only start in the presence of an atom $\texttt{cexch}(M^*)$, with M_1 and M^* unifiable, i.e. when the reduction of `cons` has reached the point of the reduction of $\texttt{cexch}(M_2)$. And vice-versa for the reduction of $\texttt{cexch}(M_2)$ with respect to the atom $\texttt{pexch}(M_1)$. Furthermore, this synchronization in the reductions involves the common reduction of the (induced instance of) the atom `treat(M)` simulating the treatment of the message M. When this is done, and only then, the reduction of the induced instances of $\texttt{prod\_res}(M_3,U_3)$ and $\texttt{cons\_res}(M_3,V_3)$, with M_3, U_3 and V_3 renamings of the variables M, U, V, are launched concurrently as the continuation of the reductions of $\texttt{pexch}(M_1)$ and $\texttt{cexch}(M_2)$, that is of `prod` and `cons`, respectively.

The reader will appreciate the ease of coding in this example, as opposed to that obtained by using the asynchronous communication of usual concurrent logic programming languages. It is also worth noting that the synchronization between the producer and the consumer takes place from the communication of the message M to the end of the treatment of this message through `treat(M)`. As limit cases, one could think of an empty treatment of M or of empty continuations `prod_res(M)` and `cons_res(M)`. The first limit case

[4]Although any Horn clause $H \leftarrow B$ can be rewritten in an equivalent extended form $H \leftarrow \triangle \mid B$, we will stick, for the time being, to the classical Horn clause notation and reserve the extended form for clauses involving strictly more than one atom in their head.

corresponds to the situation where synchronization just acts on the communication of M. The second limit case is more in the philosophy of work such as [2], [3], [8], [10]; the synchronization then consists of the synchronous communication and the achievement of a common ending task.

One could be tempted to rewrite the above program as

$$\begin{array}{l}\texttt{prod} \leftarrow \texttt{int\_prod(M,X)} \ ; \ \texttt{pexch(M)} \ ; \ \texttt{prod\_res(M)} \\ \texttt{cons} \leftarrow \texttt{int\_cons(Y)} \ ; \ \texttt{cexch(M)} \ ; \ \texttt{cons\_red(M)} \\ \texttt{pexch(M)} \diamond \texttt{cexch(M)} \leftarrow \texttt{treat(M)} \mid \Delta \diamond \Delta\end{array}$$

and to infer therefrom that it is possible, in general, to rewrite extended Horn clauses in the format of the latter limit case. This is however not always feasible from a practical point of view, as suggested by the airline reservation system described below.

As final remarks, let us note that it is, of course, possible to refine the above basic scheme in several ways. For instance, one could add extra arguments to the predicates and complicate the definition of the predicates **prod_res** and **cons_res** at will.

2.2 Towards an integration of logic and object-oriented programming

Another interesting application of extended Horn clauses concerns the integration of logic and object-oriented programming. The behavior of objects is classically represented in logic programming by the evaluation of a call to a procedure defined recursively, the successive values of the arguments representing the successive states of the object. Following this line, the treatment of a message **mess(M)** by an object **obj(S)** by means of a method **method(M)** can be schematized by one of the two following clauses:

$$\begin{array}{l}\texttt{obj(S)} \diamond \texttt{mess(M)} \leftarrow \texttt{method(M)} \mid \texttt{obj(NewS)} \diamond \Delta \\ \texttt{obj(S)} \diamond \texttt{mess(M)} \leftarrow \texttt{method(M)} \mid \texttt{obj(NewS)} \diamond \texttt{mess(M)} \quad .\end{array}$$

according as the message **mess(M)** is consumed or not. In that framework, the object conceptually moves from the state S to the new state NewS.

An instance of this scheme is given by the following description[5] of the class of stacks:

$$\begin{array}{l}\texttt{stack(Id,S)} \diamond \texttt{push(Id,X)} \leftarrow \Delta \mid \texttt{stack(Id,[X|S])} \diamond \Delta \\ \texttt{stack(Id,[X|S])} \diamond \texttt{pop(Id,X)} \leftarrow \Delta \mid \texttt{stack(Id,S)} \diamond \Delta \\ \texttt{stack(Id,[X|S])} \diamond \texttt{top(Id,X)} \leftarrow \Delta \mid \texttt{stack(Id,[X|S])} \diamond \Delta \quad .\end{array}$$

Stacks are identified there by the Id argument of the **stack** predicate and their state, implemented as a list, moves respectively from S, [X|S], [X|S] to [X|S], S, [X|S] according as a **push**, **pop** or **top** message is received. The treatment of these messages is quite straightforward so that all the guards are reduced to Δ. Nevertheless, it is easy to slightly complicate the problem in order to end up with more elaborated guards. For instance, one could require that the treatment of a **push** message includes, in addition, the check that the argument X is of some type t. In that case, the first clause of the stack procedure becomes

$$\texttt{stack(Id,S)} \diamond \texttt{push(Id,X)} \leftarrow \texttt{t(X)} \mid \texttt{stack(Id,[X|S])} \diamond \Delta.$$

The classical airline reservation system provides another interesting instance of the above scheme. The task consists here of simulating an airline reservation system composed of n agencies communicating with a global database about m flights. Using extended Horn clauses, this can be achieved by evaluating the query

$$\texttt{agency(Id1)} \parallel \cdots \parallel \texttt{agency(Idn)} \parallel \texttt{airline\_syst(DB\_init)}$$

where **agency(Idj)** represents the j^{th} agency, identified by Idj, and where **DB_init** represents the initial information about the m flights. The exact description of the agencies is out of the scope of this paper. For our illustrative purposes, it is sufficient to assume that some internal actions successively generates queries for the database and behaves correctly according to the answers. We will consider two kinds of messages: **reserve(Flight_id,Nb_seats,Ans)** and **ask_seats(Flight_id,Free_seats)**. Their goals are respectively

[5] This description has actually been inspired by that of [4].

i) to ask for the reservation of `Nb_seats` in the flight `Flight_id`, which yields the answer `Ans`;
ii) to ask the number of free seats in the flight `Flight_id`.

According to the above scheme and using the auxiliary predicates `make_reservation` and `free_seats`, with obvious meanings, the treatment of these messages can be coded as follows.

```
airline_syst(DB) ◇ reserve(Flight_id,Nb_seats,Ans) ←
    make_reservation(Flight_id,Nb_seats,DB,New_DB,Ans) | airline_syst(New_DB) ◇ △
airline_syst(DB) ◇ ask_seats(Flight_id,Free_seats) ←
    free_seats(Flight_id,Free_seats) | airline_syst(DB) ◇ △
```

The following points are worth noting. Firstly, accessing the database is achieved without explicitly handling lists of messages and without using merge processes, as usual in concurrent logic programming languages. Secondly, mutual exclusive access to the database is ensured by the synchronous mechanism. In that, our solution also contrasts with the classical concurrent logic one which involves commitment and merge processes. Finally, in opposition to the functional languages, answers are back communicated implicitly thanks to the unification mechanism and this without the use of identifiers.

2.3 More examples

Other examples, including semaphores, the seminal dining philosophers problem, generative communication in a Linda style, can be programmed with similar ease in the extended Horn clause framework. We refer the interested reader to [3], [8], [10], [16], [17], [19] for such or similar programming.

3 The language

As usual in logic programming, the extended language, subsequently referred to as *ELP*, comprises denumerably infinite sets of *variables*, *functions* and *predicates*. They are referred to as *Svar*, *Sfunct* and *Spred*, respectively. The notions of term, atom, substitution, unification, ... are defined therefrom as usual. We assume the reader to be familiar with them and will not recall them here. Rather, we now specify the extensions of goals and Horn clauses sketched in Section 1.

Definition 1

1) The extended goals *are defined inductively as follows:*
  i) $\triangle$ *is an extended goal (representing the empty goal),*
  ii) any atom is an extended goal,
  iii) if $\overline{G}_1$ *and* $\overline{G}_2$ *are extended goals, then* $(\overline{G}_1 \,;\, \overline{G}_2)$, $(\overline{G}_1 \parallel \overline{G}_2)$ *and* $(\overline{G}_1 \,\&\, \overline{G}_2)$ *are extended goals. Extended goals are typically denoted by the* $\overline{G}$ *letter, possibly subscripted. Their set is subsequently referred to as* Segoal.

2) The extended Horn clauses *are defined as clauses of the form*

$$H_1 \diamond \cdots \diamond H_m \leftarrow \overline{G} \mid \overline{G}_1 \diamond \cdots \diamond \overline{G}_m$$

where the H_i*'s are atoms and the* $\overline{G}$ *and* $\overline{G}_i$*'s are extended goals. By extension, these atoms and goals are still called the heads and bodies of the extended clause, respectively.*

3) The extended programs *or* programs, *for short, are sets of extended Horn clauses. Their set is subsequently referred to as* Sprog. ∎

Particularly notice from the above definition that clauses are considered, from now on, in their extended form only. This is justified by uniformity purposes in subsequent treatments. As a consequence, any Horn clause $H \leftarrow B$ is now rewritten in its equivalent form $H \leftarrow \triangle \mid B$.

4 Mathematical preliminaries

4.1 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 bold brackets, as in $\{\!\{a, a, b\}\!\}$, whereas sets are denoted by simple brackets, as in $\{a, b\}$. The union symbol is kept unchanged but its use is disambiguated by the nature of its operands. To avoid any ambiguity, let us further precise that, given two multi-sets S and T, we denote by $S \cup T$ the collection of all elements of S and T repeated as many times as they occur in S and T.

The usual notations $\mathcal{P}(\mathrm{E})$ and $\mathcal{M}(\mathrm{E})$ are used to denote, respectively, the set of sets and multi-sets, with elements from E. The notations $\mathcal{P}_\pi(\mathrm{E})$ and $\mathcal{M}_\pi(\mathrm{E})$ are moreover employed to denote those sets and multi-sets verifying the property π. For instance, $\mathcal{P}_{ncl}(\mathrm{E})$ denotes the set of the non-empty and closed sets with elements from E.

4.2 Reconciliation of substitutions

Full use of and-parallelism requires a way of combining substitutions issued from the concurrent reductions of subgoals of an extended goal in order to form answer substitutions for the whole extended goal. It has been provided under the name of *reconciliation of substitutions* in [13] and has been extensively studied there. Concurrently, an equivalent notion, named parallel composition of substitutions, has been developed in [18]. We briefly recall this notion here for the sake of completeness. The reader is referred to the above two references for more details.

The reconciliation of substitutions is based on the interpretation of substitutions in equational terms. Precisely, any substitution $\theta = \{X_1/t_1, \ldots, X_m/t_m\}$ 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 the system 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 with that of terms in such a way that all the 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 2 *The substitutions $\theta_1, \ldots, \theta_m$ ($m \geq 1$) are reconcilable iff the system composed of the equations of syst(θ_1), $\ldots$, 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*.

4.3 Complete lattices and metric spaces

Complete lattices and metric spaces will be used as important semantic tools. The reader is assumed to be familiar with them as well as with their related notions of convergent sequences, directed and closed 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 [15] and [9], when need be. Furthermore, lack of space prevents us from describing all the metrics used in this paper. We will however employ the classical ones and refer to [5] for such a description.

5 Semantic translation

As pointed out in section 1, synchronization can be specified in two places: in goals, by means of the operator " & ", and in clauses, by means of the operator " ◇ ". These two operators thus act in a dual way. It turns out, however, that it is possible to simulate the former by the latter, of a more dynamic nature. For instance, assuming that the predicates `a(X)` and `b(X)` are defined by the only two clauses

$$\texttt{a(Y)} \leftarrow \Delta \mid \texttt{r(Y)}$$
$$\texttt{b(Z)} \leftarrow \Delta \mid \texttt{s(Z)} \quad ,$$

the reduction of the conjunction $\texttt{a(X)}$ & $\texttt{b(X)}$ may be simulated by the reduction of $\texttt{p}_a\texttt{(X)} \parallel \texttt{p}_b\texttt{(X)}$ with $\texttt{p}_a$ and $\texttt{p}_b$ two new predicates defined by the only clause

$$\texttt{p}_a\texttt{(X)} \diamond \texttt{p}_b\texttt{(X)} \leftarrow \Delta \mid \texttt{r(X)} \diamond \texttt{s(X)}$$

Note that, with this device, we still have the possibility of using $\texttt{a(Y)}$ and $\texttt{b(Z)}$ separately.

The operator " & " is thus in some sense redundant with respect to the operator " $\diamond$ ". However, we believe that, from a language point of view, specifying synchronization in both goals and clauses is desirable and, therefore, we provide both constructs in the language. Nevertheless, this redundancy allows us to design semantics in two ways. One consists of translating the programs in the sublanguage of ELP without the operator " & " and of designing semantics for this sublanguage. The other one consists of designing semantics directly for the whole language. We have adopted here the first approach because it allows us to expose the semantics in a simpler framework – and thus in a clearer way – and because the semantics developed using the second approach can be obtained therefrom by simple extensions.

6 Auxiliary concepts

6.1 The t-contexts

Forcing atoms to synchronize introduces a need for a means to express which atoms in an extended goal are allowed to synchronize and for a means to create the goals resulting from the synchronized reductions. These means are provided by the notion of t-context. Basically, a t-context consists of a partially ordered structure where the place holder $\Box$ has been inserted in some top-level places i.e. places not constrained by the previous execution of other atoms. Atoms that can synchronize are then those that can be substituted by a place holder $\Box$ in a t-context. Furthermore, the extended goals resulting from the synchronized reductions are obtained by substituting the place holder by the corresponding bodies $\overline{G}_i$'s of the extended Horn clause used.

The precise definition of the t-contexts is as follows.

Definition 3 *The t-contexts are the functions inductively defined on the extended goals by the following rules.*

i) A nullary t-context is associated with any extended goal. It is represented by the extended goal and is defined as the constant mapping from $Segoal^0$ to this goal with the goal as value.

ii) $\Box$ is a unary t-context that maps any extended goal to itself. For any extended goal $\overline{G}$, this application is subsequently referred to as $\Box[\overline{G}]$.

iii) If c is an n-ary t-context and if $\overline{G}$ is an extended goal, then $(c\,;\overline{G})$ is an n-ary t-context. Its application is defined as follows : for any extended goals $\overline{G}_1, \ldots, \overline{G}_n$,

$$(c\,;\overline{G})[\overline{G}_1,\cdots,\overline{G}_n] = (c[\overline{G}_1,\cdots,\overline{G}_n]\,;\overline{G})$$

iv) If c_1 and c_2 are m-ary and n-ary t-contexts and if $n+m>0$, then $c_1 \parallel c_2$ is an $(m+n)$-ary t-context. Its application is defined as follows : for any extended goals $\overline{G}_1, \ldots, \overline{G}_{m+n}$,

$$(c_1 \parallel c_2)[\overline{G}_1,\cdots,\overline{G}_{m+n}] = (c_1[\overline{G}_1,\cdots,\overline{G}_m]) \parallel (c_2[\overline{G}_{m+1},\cdots,\overline{G}_{m+n}])$$

In the above rules, we further state that the structure $(Segoal, ;, \parallel, \Delta)$ is a bimonoid. Moreover, in the following, we will simplify the extended goals resulting from the application of t-contexts accordingly. ∎

The following points in the above definition are worth noting.

- Rule iii) forces the place holder $\Box$ to occur only in a position corresponding to atoms that can be reduced in the first reduction step of an associated extended goal.
- Rule iv) forces a composed t-context $c_1 \parallel c_2$ to include one place holder in at least one c_i although both can contain one. This corresponds to the fact that, to allow a composed goal $\overline{G}_1 \parallel \overline{G}_2$ to perform one reduction step, at least one of the conjunct $\overline{G}_i$ must perform one reduction step although both can do so simultaneously.

6.2 Program expansion

The extended clauses to consider to reduce extended goals are those obtained from the clauses of the (written) program by permuting them and by grouping them. To avoid handling this permutation and groupment explicitly, we now associate to any program P the program P^* that performs this task implicitly.

Definition 4 *For any program P, the* expansion *of P, denoted P^*, is defined as the following program:*

i) any clause of P is a clause of P^*;

ii) if $(L_1 \diamond \cdots \diamond L_p \leftarrow \overline{A} \mid \overline{S}_1 \diamond \cdots \diamond \overline{S}_p)$ and $(M_1 \diamond \cdots \diamond M_q \leftarrow \overline{B} \mid \overline{T}_1 \diamond \cdots \diamond \overline{T}_q)$ are clauses of P^*, renamed to avoid variable clashes, then

$$L_1 \diamond \cdots \diamond L_p \diamond M_1 \diamond \cdots \diamond M_q \leftarrow (\overline{A} \parallel \overline{B}) \mid \overline{S}_1 \diamond \cdots \diamond \overline{S}_p \diamond \overline{T}_1 \diamond \cdots \diamond \overline{T}_q$$

is a clause of P^*;

iii) if $(H_1 \diamond \cdots \diamond H_m \leftarrow \overline{G} \mid \overline{G}_1 \diamond \cdots \diamond \overline{G}_m)$ is a clause of P^* and if $(\nu_1, \ldots, \nu_m)$ is a permutation of $(1, \ldots, m)$, then $(H_{\nu_1} \diamond \cdots \diamond H_{\nu_m} \leftarrow \overline{G} \mid \overline{G}_{\nu_1} \diamond \cdots \diamond \overline{G}_{\nu_m})$ is a clause of P^*. ∎

7 Operational semantics

A first semantics of ELP may be expressed operationally in terms of a derivation relation, written as $P \vdash \overline{G}$ *with* θ that, basically, expresses the property that, given the program P, the extended goal $\overline{G}$ has a successful derivation producing the substitution θ. It is defined by means of rules of the form

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

asserting the Conclusion whenever the Assumptions and Conditions hold. Note that Assumptions and Conditions may be absent from some rules. Precisely, the derivation relation is defined as the smallest relation of *Sprog* × *Segoal* × *Ssubst* satisfying the following rules (N-I) and (E-I). As usual, the above notation is used instead of the relational one with the aim of suggestiveness.

Definition 5 (The derivation relation)

Null formula

(N-I) $\quad \dfrac{}{P \vdash \Delta \textit{ with } \epsilon}$

Extended formula

(E-I) $\quad \dfrac{P \vdash \overline{G}\theta \textit{ with } \sigma \quad P \vdash c[\overline{G_1}, \ldots, \overline{G_m}]\theta\sigma \textit{ with } \gamma}{P \vdash c[A_1, \ldots, A_m] \textit{ with } \theta\sigma\gamma}$

$$\text{if} \quad \begin{cases} (H_1 \diamond \cdots \diamond H_m \leftarrow \overline{G} \mid \overline{G_1} \diamond \cdots \diamond \overline{G_m}) \in P^* \ [6] \\ < A_1, \cdots, A_m > \textit{ and } < H_1, \cdots, H_m > \textit{ unify with mgu } \theta \end{cases}$$

∎

The derivation operational semantics can be derived therefrom as follows.

Definition 6 (The derivation operational semantics) *Define the derivation operational semantics as the following function O_d : Sprog → Segoal → $\mathcal{P}$(Ssubst): for any $P \in$ Sprog, $\overline{G} \in$ Segoal, $O_d(P)(\overline{G}) = \{\theta : P \vdash \overline{G} \textit{ with } \theta\}$.* ∎

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

8 The declarative semantics

One of the distinctive features of a logic programming language is that its semantics can be understood in at least two complementary ways, inherited from logic. The operational semantics, based on proof theory, describes the method for executing programs. The declarative semantics, based on model theory, explains the meaning of programs in terms of the set of their logical consequences. Any claim to the effect that a given language is a logic programming language must be substantiated by providing suitable logic-based semantic characterizations. The operational semantics of the language ELP under consideration in this paper has been studied in the previous section. The present section is devoted to the discussion of the declarative semantics.

One might at first think that the usual notion of (Herbrand) interpretation for Horn clause logic carries through to ELP. Thus an interpretation would be a set of ground atomic formulae, with the intended meaning that the formulae in the set are true under the interpretation. The truth of compound formulae would then be derived in a compositional manner. The problem with this is that the parallel composition is not a propositional operation in that its truth or falsity can not be derived from that of its arguments. More precisely, if both arguments are true then their parallel composition is also true, but if one or both are false then the parallel composition may be true or false. For example, A and B are false both for the empty program and for the program consisting of the clause $A \diamond B \leftarrow \Delta \mid \Delta \diamond \Delta$ alone. However, $A \parallel B$ is false for the first program and true for the second one.

Note that the sequential composition is not affected by a similar problem. Indeed, a sequential composition of goals is true if and only if the component goals are true, so that declaratively the sequential composition is just the logical conjunction. In any case we can not hope to be able to specify which formulae are true by giving only the true atomic formulae. We are thus led to consider an extended Herbrand base containing parallel compositions of ground extended goals, and take its subsets as our interpretations.

Definition 7 *The* extended Herbrand base *EB is the set of all ground atomic formulae A together with all parallel compositions $\overline{G}_1 \parallel \overline{G}_2$ of nonempty ground extended goals $\overline{G}_1$ and $\overline{G}_2$. An* interpretation *is a subset I of EB.* ∎

Definition 8 *Given a formula F, its truth in I, written $\models_I F$, is defined inductively by the following rules:*

i) If F is a clause or an extended goal, $\models_I F$ if $\models_I F_0$ for every ground instance F_0 of F.
ii) For a ground clause, $\models_I (A_1 \diamond \cdots \diamond A_n \leftarrow \overline{G} \mid \overline{G}_1 \diamond \cdots \diamond \overline{G}_n)$ if, for every n-ary ground t-context c, $\models_I c[A_1, \ldots, A_n]$ whenever $\models_I (\overline{G} \mid c[\overline{G_1}, \ldots, \overline{G_n}])$.
iii) $\models_I \Delta$.
iv) If $\overline{G}$ and $\overline{H}$ are ground goals, $\models_I (\overline{G}\,;\overline{H})$ if $\models_I \overline{G}$ and $\models_I \overline{H}$.
v) If $\overline{G}$ and $\overline{H}$ are ground goals, $\models_I (\overline{G} \mid \overline{H})$ if $\models_I \overline{G}$ and $\models_I \overline{H}$.
vi) If $\overline{G}$ and $\overline{H}$ are ground goals, $\models_I (\overline{G} \parallel \overline{H})$ if $(\overline{G} \parallel \overline{H}) \in I$ or $\models_I \overline{G}$ and $\models_I \overline{H}$.
vii) If A is a ground atomic formula, $\models_I A$ if $A \in I$. ∎

Definition 9 *An interpretation I is a* model *of a program P if $\models_I C$ for every clause $C \in P$. An extended goal $\overline{G}$ is said to be a* consequence *of P, written $P \models \overline{G}$, if $\models_I \overline{G}$ for every model I of P. The* success set *of $\overline{G}$ with respect to P is the set $SS_D(\overline{G}) = \{\theta : P \models \overline{G}\theta\}$ of all substitutions θ such that $\overline{G}\theta$ is a consequence of P.* ∎

We are now in a position to define the model declarative semantics.

Definition 10 (Model declarative semantics) *Define the model declarative semantics as the following function $Decl_m : Sprog \rightarrow Segoal \rightarrow \mathcal{P}(Ssubst)$: for any $P \in Sprog$, $\overline{G} \in Segoal$, $Decl_m(P)(\overline{G}) = SS_D(\overline{G})$.* ∎

If I and J are interpretations and F is a formula, it is easy to see by induction on the structure of F that $\models_{I \cap J} F$ if and only if $\models_I F$ and $\models_J F$. If we take for F the clauses of P, we conclude that the intersection of two models of P is again a model. This statement can obviously be generalized to the intersection of an arbitrary number of models. Since EB is a model, it follows that any program has a least model.

Proposition 11 *Every program P has a least model M_P.* ∎

The importance of M_P is that it allows to simplify the definition of success set: instead of requiring that $\overline{G}\theta$ be true in all models of P it is enough that it is true in M_P. Indeed, this is a consequence of the easy fact that if I and J are interpretations such that $I \subseteq J$ then $\models_I \overline{G}$ implies $\models_J \overline{G}$.

Proposition 12 $SS_D(\overline{G}) = \{\theta : \models_{M_P} \overline{G}\theta\}$. ∎

The least model M_P can also be characterized as the least fixed point of a continuous transformation $T_P : \mathcal{P}(EB) \rightarrow \mathcal{P}(EB)$, called as usual the *immediate consequence operator*. For every interpretation I, $T_P(I)$ is the set of all ground extended goals of the form $c[A_1, \ldots, A_n] \in EB$ such that $\models_I \overline{G}$ and $\models_I c[\overline{G_1}, \ldots, \overline{G_n}]$, for an n-ary ground t-context c and a ground instance $A_1 \diamond \cdots \diamond A_n \leftarrow \overline{G} \mid \overline{G_1} \diamond \cdots \diamond \overline{G_n}$ of a clause in P^*.

Proposition 13 *The operator T_P is continuous and M_P is the least fixed point of T_P.* ∎

The *fixed-point* semantics of P associates with each $\overline{G}$ the set of all θ such that $\overline{G}\theta$ is true in the least fixed point *lfp(T_P)* of T_P.

Definition 14 (Fixed-point declarative semantics) *Define the fixed-point declarative semantics as the following function $Decl_f : Sprog \rightarrow Segoal \rightarrow \mathcal{P}(Ssubst)$: for any $P \in Sprog$, $\overline{G} \in Segoal$, $Decl_f(P)(\overline{G}) = \{\theta : \models_{lfp(T_P)} \overline{G}\theta\}$.* ∎

Proposition 13 establishes the equivalence between the declarative and the fixed-point semantics of P.

Proposition 15 $Decl_m = Decl_f$. ∎

Finally, the equivalence between the operational and the declarative semantics can be stated as follows.

Proposition 16 *For every program P and every extended goal $\overline{G}$,*

i) if $P \vdash \overline{G}$ with θ, for some substitution θ, then $P \models \overline{G_0}$ for every ground instance $\overline{G_0}$ of $\overline{G}\theta$;
ii) if $P \models \overline{G}\tau$ for some substitution τ, then $P \vdash \overline{G}$ with θ, for some substitution θ such that $\overline{G}\tau \geq \overline{G}\theta$.

In particular, let $\alpha_1 : \mathcal{P}(Ssubst) \rightarrow \mathcal{P}(Ssubst)$ be the following function: for any $\Theta \in \mathcal{P}(Ssubst)$,

$$\alpha_1(\Theta) = \{\theta\gamma_{|S} : \theta \in \Theta, \gamma \in Ssubst, dom(\theta) \subseteq S\}$$

where $\theta\gamma_{|S}$ is the restriction of $\theta\gamma$ to the variables of S and $dom(\theta)$ denotes the domain of θ. Then, the equality

$$Decl_m(P)(\overline{G}) = Decl_f(P)(\overline{G}) = \alpha_1(O_d(P)(\overline{G}))$$

holds for any $P \in Sprog$, $\overline{G} \in Segoal$. ∎

9 The denotational semantics

This section introduces our last semantics. It is defined compositionally and makes no use of transition systems as well as no reference to any declarative paradigm. It is called denotational in view of these properties.

Compositionality of the semantics requires to determine the semantics of a compound goal in terms of the semantics of its components. However, as pointed out in section 8, this is not straightforward to realize for ELP. The problem is essentially that the failure or the suspension of a compound goal cannot be inferred directly from the failure or the suspension of its components considered individually. One way of solving this problem consists of taking into account environments composed of concurrent atoms (if any) that would unsuspend the suspended derivations of the components. To be more specific, with respect to the one clause program $A \diamond B \leftarrow \triangle \mid \triangle \diamond \triangle$ our idea is to deliver as semantics for A not failure nor a simple suspension but a suspension mark together with the derivation obtained by assuming the presence of B in concurrence with A. Giving a similar semantics for B, it is not difficult to imagine that it is possible to combine the semantics of A and of B to obtain that of $A \parallel B$. In general, the denotational semantics, to be presented subsequently, makes hypotheses about the environment of the reduction of a goal in order to unsuspend

suspended derivations. Technically speaking, these hypotheses are inserted as members of the histories; they take the form $hyp[(A,\Sigma),(B,\Upsilon)]$ with the reading that given that A is composed of the atoms that can be reduced in the treated goal and given that Σ is composed of the associated substitutions in the derivation (representing the results computed sofar by the parallel components of the considered goal), the presence of concurrent atoms of B associated with the substitutions of Υ allows the considered suspended reduction to resume. As extended goals and the head of extended clauses may contain multiple occurrences of an atom, the A, B, Σ and Υ are designed as multi-sets.

The above example might lead to think that the presence of hypotheses in the histories suppress the grounds for existence of suspension marks. This is not true as shown by the program composed of the two clauses $A \diamond B \leftarrow \Delta \mid \Delta \diamond \Delta$ and $A \diamond B \diamond C \leftarrow \Delta \mid \Delta \diamond \Delta \diamond \Delta$. The hypothetical way of reasoning includes an hypothetical derivation assuming the existence of C in the semantics of $A \parallel B$. Nevertheless, despite it, the reduction of $A \parallel B$ does not suspend. Hence, any suspended reduction needs still to be associated with one reduction ending with one suspension mark. It takes the form $susp[\{(A_1,\sigma_1),\ldots,(A_m,\sigma_m)\},B]$ where the (A_i,σ_i) 's are the top-level atoms of the considered goal with their associated substitutions and where B is the set of the heads of the clauses that would allow the reduction to resume in case suitable atoms would be placed in concurrence with the treated goal.

A final technicality is involved in the denotational semantics. Treating in a compositional way a sequentially composed goal requires to be able to give the semantics of the second component of the goal in view of the results (i.e. substitutions) computed by the first component of the goal. Hence, the denotational semantics should deliver, for any given program and any given extended goal, not some set of histories but some function that maps any substitution to such a set. In order to ease the determination of the results, the termination mark in success is furthermore enriched by the set of substitutions computed during the considered derivation.

The following definition precises the concepts just introduced.

Definition 17

1) *An* hypothetical statement *is a construct of the form* $\mathrm{hyp}[(\mathrm{A},\Sigma),(\mathrm{B},\Upsilon)]$ *where A and B are multi-sets of atomic formulae and where* Σ *and* Υ *are multi-sets of substitutions. In the following, hypothetical statements are typically denoted by the* hh *symbol and their set is referred to as* Shyp.
2) *A* suspension statement *is a construct of the form* $\mathrm{susp}[\{(\mathrm{A}_1,\sigma_1),\ldots,(\mathrm{A}_\mathrm{m},\sigma_\mathrm{m})\},\mathrm{B}]$ *where the* A_i*'s are atomic formulae, the* σ_i*'s are substitutions and B is a set of multi-sets of atomic formulae. In the following, suspension statements are typically denoted by the* ss *symbol and their set is referred to as* Ssusp.
3) *Let* Sterm *be the set composed of the element* fail *and constructs of the form* $\mathrm{succ}(\Theta)$ *where* Θ *is a set of substitutions. The set of denotational histories,* Sdhist *is defined as the solution of the following recursive equation:*

$$Sdhist = Sterm \cup Ssusp \cup (Ssubst \times Sdhist) \cup (Shyp \times Ssubst \times Sdhist) \times Sdhist$$

(see [6] or [1] for the resolution of this equation). Histories are thus streams written as $(e_1,(e_2,(e_3,\cdots)))$ *thanks to the cartesian products. They are often rewritten in the simpler form* $e_1.e_2.e_3.\cdots$ *to avoid the intricate use of brackets. However, the structure of hypotheses followed by substitutions followed by guard evaluations will be conserved and written as triplets.*
Histories are typically denoted by the h *letter. Histories of the form* $< hh,\theta,g > .h$ *with* $hh \in Shyp$, $\theta \in Ssubst$ *and* $g,h \in Shist$ *are subsequently called* hypothetical histories. *Histories containing no hypothetical statement are called real histories. Their set is referred to as* Srhist. *Histories containing no suspension statements are called* unsuspended histories. *Their set is referred to as* Suhist.
4) *A set of histories S is coherent iff for any suspended history of S of the form*

$$hp.susp[\{(A_1,\sigma_1),\cdots,(A_m,\sigma_m)\},C]$$

there is in S an history of the form $hp. < hyp[(A,\Sigma),(B,\Gamma)],\theta,g > .hs$ *such that* $A = \{A_{\nu_1},\cdots,A_{\nu_p}\}$ *and* $\Sigma = \{\sigma_{\nu_1},\cdots,\sigma_{\nu_p}\}$*, for some subsequence* $(\nu_1,\cdots,\nu_p)$ *of* $(1,\cdots,m)$.
5) *The semantic domain* Sem *is defined as the (complete metric) space* $\mathcal{P}_{nccl}(Sdhist)$ *of non-empty, coherent and closed subsets of Sdhist.* ∎

Semantic counterparts for the operators " ; " and " | " can be defined quite directly. The recursive nature of streams might suggest recursive definitions. However, their possible infinite nature makes direct definitions incorrectly stated. This problem is circumvented by using a higher-order function Ψ_{seq} of the same recursive nature but that turns out to be a well-defined contraction.

Definition 18 *Define the operator* $\Psi_{seq} : [(Sdhist \times (Ssubst \rightarrow Sem)) \rightarrow Sem] \rightarrow [(Sdhist \times (Ssubst \rightarrow Sem)) \rightarrow Sem]$ *as follows : for any* $F \in [(Sdhist \times (Ssubst \rightarrow Sem)) \rightarrow Sem]$, $f \in (Ssubst \rightarrow Sem)$, $\Theta \subseteq Ssubst$, $ss \in Ssusp$, $e \in \mathcal{M}(Ssubst) \cup Shyp$, $h \in Shist$,

i) $\Psi_{seq}(F)(fail, f) = \{fail\}$

ii) $\Psi_{seq}(F)(succ(\Theta), f) = \begin{cases} f(\theta) & \text{if } \Theta \text{ is reconcilable with reconciliation } \theta \\ \{fail\} & \text{otherwise} \end{cases}$

iii) $\Psi_{seq}(F)(ss, f) = \{ss\}$

iv) $\Psi_{seq}(F)(e.h, f) = \{e.h^* : h^* \in F(h, f)\}$. ∎

Proposition 19 *The function* Ψ_{seq} *is well-defined and is a contraction.* ∎

Definition 20

1) Define the operator $\widehat{;}_{str} : (Sdhist \times (Ssubst \rightarrow Sem)) \rightarrow Sem$ *as the fixed point of* Ψ_{seq}.

2) Define the operator $\widetilde{;} : ((Ssubst \rightarrow Sem) \times (Ssubst \rightarrow Sem)) \rightarrow (Ssubst \rightarrow Sem)$ *as follows: for any* $f_1 : (Ssubst \rightarrow Sem)$, $f_2 : (Ssubst \rightarrow Sem)$ *and for any* $\sigma \in Ssubst$,

$$(f_1 \widetilde{;} f_2)(\sigma) = \{h : h_1 \in f_1(\sigma), h \in h_1 \widehat{;}_{str} f_2\}$$ ∎

Definition 21 *The counterpart of the operator "*$\widehat{;}_{str}$*" on* $Sdhist \times Sdhist \rightarrow Sdhist$ *is defined similarly to definitions 18 and 20 and is denoted by "*$\odot$*"* ∎

The operator " | " has the sequential nature of the operator " ; " but further constraints its left-hand side argument to be evaluated on its own. This latter feature is semantically modeled by preventing the argument evaluation to make hypotheses about its (non-existing) environment and by prohibiting suspended executions (of this evaluation) to be resumed thanks to the environment. Technically speaking, these two points are respectively achieved by eliminating hypothetical histories from the denotational semantics of the argument and by emptying the second argument of suspension marks of histories of this semantics. These are essentially the goals attached to the following function *guard*. For ease of subsequent use, it is defined at the level of functions of $Ssubst \rightarrow \mathcal{P}(Sdhist)$ rather than at the level of sets of $\mathcal{P}(Sdhist)$.

Definition 22 *Define the function guard* $: [Ssubst \rightarrow \mathcal{P}(Sdhist)] \rightarrow [Ssubst \rightarrow \mathcal{P}(Sdhist)]$ *as follows: for any* $f \in [Ssubst \rightarrow \mathcal{P}(Sdhist)]$, *any* $\sigma \in Ssubst$,

$$guard(f)(\sigma) = (f(\sigma) \cap Srhist \cap Suhist) \cup \{h.susp(\mathbf{A}, \emptyset) : h.susp(\mathbf{A}, B) \in f(\sigma)\}.$$ ∎

It is possible to prove that the function guard conserves the non-empty, closed and coherent features of the elements of Sem. The operator " $\widetilde{|}$ " can thus be defined in terms of the operator " $\widetilde{;}$ " and the function guard.

Definition 23 *Define* $\widetilde{|} : ((Ssubst \rightarrow Sem) \times (Ssubst \rightarrow Sem)) \rightarrow (Ssubst \rightarrow Sem)$ *as follows: for any* $f_1, f_2 \in (Ssubst \rightarrow Sem)$, $f_1 \widetilde{|} f_2 = guard(f_1) \widetilde{;} f_2$. ∎

The construction of hypothetical histories requires an operator like the " | " operator but that conserves the marks of the guards. It is defined as the following operator " $\Rightarrow$ ".

Definition 24 *Define the function* $\Rightarrow : (Shyp \times Ssubst \times (Ssubst \rightarrow Sem) \times (Ssubst \rightarrow Sem)) \rightarrow (Ssubst \rightarrow Sem)$ *as follows: for any* $hh \in Shyp$, $f_1, f_2 : (Ssubst \rightarrow Sem)$, $\sigma, \theta \in Ssubst$,

$$\begin{aligned}(f_1 \Rightarrow_{hh,\theta} f_2)(\sigma) = \{ & < hh, \theta, g > .fail : g \in guard(f_1)(\sigma), g \text{ is infinite } \} \\ & \cup \{< hh, \theta, g.ss > .fail : g.ss \in guard(f_1)(\sigma)\} \\ & \cup \{< hh, \theta, g.fail > .fail : g.fail \in guard(f_1)(\sigma)\} \\ & \cup \{< hh, \theta, g.succ(\Omega) > .h : g.succ(\Omega) \in guard(f_1)(\sigma), \\ & \qquad \Omega \text{ is reconcilable with reconciliation } \omega, h \in f_2(\omega)\} \\ & \cup \{< hh, \theta, g.succ(\Omega) > .fail : g.succ(\Omega) \in guard(f_1)(\sigma), \\ & \qquad \Omega \text{ is not reconcilable } \}\end{aligned}$$ ∎

In order to define the semantic counterpart " $\widetilde{\|}$ " of the parallel composition operator, let us first introduce two auxiliary operators. The first one, the operator " $\widehat{\|}_{susp}$ " determines how the ending suspension marks of derivations should be combined in parallel.

Definition 25 *Define the auxiliary operator* $\widehat{\|}_{susp}$ *on suspension statements as follows.*

$$susp[T_1, S_1] \ \widehat{\|}_{susp} \ susp[T_2, S_2] = \begin{cases} \emptyset & \text{if } C \text{ holds} \\ \{susp[T_1 \cup T_2, S_1 \cup S_2]\} & \text{otherwise} \end{cases}$$

where C stands for the following condition : there is $\{(A_1,\sigma_1),\ldots,(A_m,\sigma_m)\} \subseteq T_1$, $m > 0$, $\{(B_1,\tau_1),\ldots,(B_n,\tau_n)\} \subseteq T_2$, $n > 0$ and $\{H_1,\ldots,H_{m+n}\} \in S_1 \cap S_2$ such that $\{\sigma_1,\ldots,\sigma_m,\tau_1,\ldots,\tau_n\}$ is reconcilable, say with reconciliation θ, and such that $< A_1,\ldots,A_m,B_1,\ldots,B_n >\theta$ and $< H_1,\ldots,H_{m+n} >$ are unifiable ∎

The other auxiliary operator " $\widehat{\|}_{syhyp}$ " specifies how hypothetical histories should be combined in a synchronized parallel fashion.

Definition 26 *Define the operator* $\widehat{\|}_{syhyp}$ *on hypothetical histories and functions of* $(Sdhist \times Sdhist \rightarrow Sem)$ *as follows:*

$$< hyp[(A,\Sigma),(B,\Gamma)],\theta_1,g_1 > .h_1 \ \widehat{\|}^F_{syhyp} \ < hyp[(C,\Upsilon),(D,\Psi)],\theta_2,g_2 > .h_2$$
$$= \begin{cases} \{< hyp[(V,\Omega),(W,\Phi)],\theta_1,g_1 > .h : Desc\} & \text{if } Cond_1 \text{ and } Cond_2 \text{ holds} \\ \{\theta_1.g_1 \odot h\} & \text{if } Cond_1 \text{ holds and } Cond_2 \text{ does not hold} \\ \emptyset & \text{otherwise} \end{cases}$$

where $Cond_1$, $Cond_2$ and $Desc$ stand for the following conditions:

- *$Cond_1$:* *i)* $\theta_1 = \theta_2$ *ii)* $g_1 = g_2$ *iii)* $A \subseteq D$ *iv)* $\Sigma \subseteq \Psi$ *v)* $C \subseteq B$ *vi)* $\Upsilon \subseteq \Gamma$ *vii)* $A \cup B = C \cup D$ *viii)* $\Sigma \cup \Gamma = \Upsilon \cup \Psi$
- *$Cond_2$:* *i)* $B \backslash C \neq \emptyset$ *ii)* $\Gamma \backslash \Upsilon \neq \emptyset$
- *Desc:* *i)* $V = A \cup C$ *ii)* $\Omega = \Sigma \cup \Upsilon$ *iii)* $W = B \backslash C$ *iv)* $\Phi = \Gamma \backslash \Upsilon$ *v)* $h \in F(h_1,h_2)$ ∎

We are now in a position to define the semantic counterpart " $\widetilde{\|}$ " of the operator " $\|$ ". As before, a suitable operator is used to provide a correct recursive definition. It is defined on histories rather than on sets of histories for the ease of the presentation.

Definition 27 *Define the function* $\Psi_{para} : (Sdhist \times Sdhist \rightarrow \mathcal{P}_{cl}(Sdhist)) \rightarrow (Sdhist \times Sdhist \rightarrow \mathcal{P}_{cl}(Sdhist))$ *as follows : for any* $F \in (Sdhist \times Sdhist \rightarrow \mathcal{P}_{cl}(Sdhist))$, *for any* $g, g_1, g_2, h, h_1, h_2 \in Sdhist$, $ss, ss_1, ss_2 \in Ssusp$, $hh, hh_1, hh_2 \in Shyp$, $\theta, \theta_1, \theta_2 \in Ssubst$, $\Theta, \Theta_1, \Theta_2 \subseteq Ssubst$,

i) $\Psi_{para}(F)(fail,h) = \Psi_{para}(F)(h,fail) = \{fail\} \cup \{\theta.h^* : \theta.h_r \in h, h^* \in F(fail,h_r)\}$
$\cup \{< hh,\theta,g > .h^* :< hh,\theta,g > .h_r \in h, h^* \in F(fail,h_r)\}$

ii) $\Psi_{para}(F)(succ(\Theta_1),succ(\Theta_2)) = \{succ(\Theta_1 \cup \Theta_2)\}$

iii) $\Psi_{para}(F)(succ(\Theta),ss) = \Psi_{para}(F)(ss,succ(\Theta)) = \{ss\}$

iv) $\Psi_{para}(F)(succ(\Theta_1),\theta_2.h) = \Psi_{para}(F)(\theta_2.h,succ(\Theta_1)) = \{\theta_2.h^* : h^* \in F(succ(\Theta_1),h)\}$

v) $\Psi_{para}(F)(succ(\Theta),< hh,\theta,g > .h) = \Psi_{para}(F)(< hh,\theta,g > .h,succ(\Theta))$
$= \{< hh,\theta,g > .h^* : h^* \in F(succ(\Theta),h)\}$

vi) $\Psi_{para}(F)(ss_1,ss_2) = ss_1 \ \widehat{\|}_{susp} \ ss_2$

vii) $\Psi_{para}(F)(ss,\theta.h) = \Psi_{para}(F)(\theta.h,ss) = \{\theta.h^* : h^* \in F(ss,h)\}$

viii) $\Psi_{para}(F)(ss,< hh,\theta,g > .h) = \Psi_{para}(F)(< hh,\theta,g > .h,ss) = \{< hh,\theta,g > .h^* : h^* \in F(ss,h)\}$

ix) $\Psi_{para}(F)(< hh_1,\theta_1,g_1 > .h_1,< hh_2,\theta_2,g_2 > .h_2) =< hh_1,\theta_1,g_1 > .h_1 \ \widehat{\|}^F_{syhyp} \ < hh_2,\theta_2,g_2 > .h_2$
$\cup \{< hh_1,\theta_1,g_1 > .h^* : h^* \in F(h_1,< hh_2,\theta_2,g_2 > .h_2\}$
$\cup \{< hh_2,\theta_2,g_2 > .h^* : h^* \in F(< hh_1,\theta_1,g_1 > .h_1,h_2\}$

x) $\Psi_{para}(F)(< hh_1,\theta_1,g_1 > .h_1,\theta_2.h_2) = \Psi_{para}(F)(\theta_2.h_2,< hh_1,\theta_1,g_1 > .h_1)$
$= \{\theta_2.h : h \in F(< hh_1,\theta_1,g_1 > .h_1,h_2)\} \cup \{< hh_1,\theta_1,g_1 > .h : h \in F(h_1,\theta_2.h_2)\}$

xi) $\Psi_{para}(F)(\theta_1.h_1,\theta_2.h_2) = \{\theta_1.h : h \in F(h_1,\theta_2.h_2)\} \cup \{\theta_2.h : h \in F(\theta_1.h_1,h_2)\}$ ∎

Proposition 28 *The function Ψ_{para} is a contraction.* ∎

Definition 29

1) *Define the operator $\widehat{\|}_{str} : Sdhist \times Sdhist \to \mathcal{P}_{cl}(Sdhist)$ as the unique fixed point of the contraction Ψ_{para}*
2) *Define the operator $\widetilde{\|} : ((Ssubst \to Sem) \times (Ssubst \to Sem)) \to (Ssubst \to Sem)$ as follows: for any $f_1, f_2 \in (Ssubst \to Sem)$ and any $\sigma \in Ssubst$,*

$$(f_1 \,\widetilde{\|}\, f_2)(\sigma) = \bigcup\{h_1 \,\widehat{\|}_{str}\, h_2 : h_1 \in f_1(\sigma), h_2 \in f_2(\sigma)\}$$

∎

Given the semantical counterparts "$\widetilde{\|}$", "$\widetilde{;}$" and "$\widetilde{|}$" of the operators "$\|$", "$;$" and "$|$", defining the denotational semantics essentially consists of defining the semantics for the basic constructs, namely the empty goal and the extended goals composed of one atom. The semantics of the former goal is quite obvious: success is returned together with the empty substitution ϵ. The semantics of a goal of the latter form, say A placed in the context of the substitution σ, is of a fourthfold nature: it contains

i) derivations started by any clause $(H \leftarrow \overline{G} \mid \overline{B})$ that unifies with $A\sigma$; the corresponding histories are composed of the mgu θ of the corresponding unification (precisely, the set formed of this mgu) followed by the histories of the semantics of $\overline{G} \mid \overline{B}$ in the state $\sigma\theta$;
ii) hypothetical histories for any extended Horn clause C and any multiset of atoms and substitutions that put in concurrence with the goal would allow C to be used; they are composed of the corresponding hypothetical statement followed by the mgu θ corresponding to the unification with the treated clause, an history of the guard evaluation and one execution of the corresponding body part of C
iii) a suspension mark for such extended Horn clauses, if any;
iv) a fail mark in case none of the previous histories can be delivered in the semantics

As before, a suitable higher-order contraction is used to tackle recursivity adequately.

Definition 30 *Define the operator $\Psi_{den} : [Sprog \to Segoal \to Ssubst \to Sem] \to [Sprog \to Segoal \to Ssubst \to Sem]$ as follows : for any $F : [Sprog \to Segoal \to Ssubst \to Sem]$, any $P \in Sprog$, any $\sigma \in Ssubst$, any atom A, any $\overline{G}_1, \overline{G}_2 \in Segoal$,*

i) $\Psi_{den}(F)(P)(A)(\sigma) =$
$\{\, \theta.F(P)(\overline{G} \mid \overline{B})(\sigma\theta) : (H \leftarrow \overline{G} \mid \overline{B}) \in P, A\sigma \text{ and } H \text{ unify with mgu } \theta \,\}$
$\cup \bigcup\{[\, \Psi_{den}(F)(P)(\overline{G}) \Rightarrow_{hh,\tau\theta} F(P)(\overline{G}_1)](\tau\theta) : \ Cond_h \,\}$
$\cup \{\, susp[\{(A,\sigma)\}, C] : \ Cond_s \,\}$
$\cup \{\, fail : \ Cond_f \,\}$

ii) $\Psi_{den}(F)(P)(\Delta)(\sigma) = \{succ(\{\sigma\})\}$

iii) $\Psi_{den}(F)(P)(\overline{G}_1 \parallel \overline{G}_2)(\sigma) = [\, \Psi_{den}(F)(P)(\overline{G}_1) \,\widetilde{\|}\, \Psi_{den}(F)(P)(\overline{G}_2)](\sigma)$

iv) $\Psi_{den}(F)(P)(\overline{G}_1 \,;\, \overline{G}_2)(\sigma) = [\, \Psi_{den}(F)(P)(\overline{G}_1) \,\widetilde{;}\, F(P)(\overline{G}_2)](\sigma)$

v) $\Psi_{den}(F)(P)(\overline{G}_1 \mid \overline{G}_2)(\sigma) = [\, \Psi_{den}(F)(P)(\overline{G}_1) \,\widetilde{|}\, F(P)(\overline{G}_2)](\sigma)$

where $Cond_h$, $Cond_s$ and $Cond_f$ stand for the following conditions[7]

- $Cond_h$:
  i) $\{\sigma\} \cup \Theta$ *reconcile with reconciliation* τ,
  ii) $(H_1 \diamond \cdots \diamond H_{n+1} \leftarrow \overline{G} \mid \overline{G}_1 \diamond \cdots \diamond \overline{G}_{n+1}) \in P^*$,
  iii) $< A, B_1, \cdots, B_n > \tau$ *and* $< H_1, \cdots, H_{n+1} >$ *unify with mgu* θ
  iv) $n \geq 1$
  v) $hh = hyp[(\{A\}, \{\sigma\}), (\{B_1, \ldots, B_n\}, \Theta)]$
- $Cond_s$:
  i) there are $B_1, \ldots, B_n$ *with* $n \geq 1$ *and a clause* $(H_1 \diamond \cdots \diamond H_{n+1} \leftarrow \overline{G} \mid \overline{G}_1 \diamond \cdots \diamond \overline{G}_{n+1}) \in P^*$, *such that* $< A, B_1, \cdots, B_n > \sigma$ *and* $< H_1, \cdots, H_{n+1} >$ *unify*
  ii) C is the set of the multisets of the atoms of the heads of all such general Horn clauses
  iii) $A\sigma$ *unifies with no clause of* P^*
- $Cond_f$: *For any clause* $(H_1 \diamond \cdots \diamond H_{n+1} \leftarrow \overline{G} \mid \overline{G}_1 \diamond \cdots \diamond \overline{G}_{n+1}) \in P^*$, *and any* $B_1, \ldots, B_n$, $n \geq 0$, $< A, B_1, \cdots, B_n > \sigma$ *and* $< H_1, \cdots, H_{n+1} >$ *do not unify.*

∎

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

Proposition 31 *The function Ψ_{den} is well-defined and is a contraction.* ∎

Definition 32 *Define the denotational semantics $Den : [Sprog \to Segoal \to Ssubst \to Sem]$ as the unique fixed point of Ψ_{den}* ∎

We conclude this section by relating the semantics Den and O_d. Obviously, Den contains more information than O_d so that relating them consists of finding a function α_2 such that $\alpha_2 \circ Den = O_d$. The appropriate function α_2 operates essentially by retaining the non-hypothetical and successful histories from Den and by delivering, for each of them, the substitution computed by it, if any. Its precise definition is as follows.

Definition 33 *Define the function $\alpha_2 : \mathcal{P}(Sdhist) \to \mathcal{P}(Sdhist)$ as follows: for any $S \in \mathcal{P}(Sdhist)$,*

$$\alpha_2(S) = \{\rho(\Theta) : h.succ(\Theta) \in S \cap Srhist, \Theta \text{ is reconcilable}\}$$ ∎

Proposition 34 *The function $\alpha_2 \hat{\circ} Den : Sprog \to Segoal \to \mathcal{P}(Ssubst)$ defined as*

$$(\alpha_2 \hat{\circ} Den)(P)(\overline{G}) = \alpha[Den(P)(\overline{G})(\epsilon)],$$

for any program P and extended goal $\overline{G}$, equals O_d. ∎

10 Conclusion

The paper has presented an extension of the Horn clause framework as well as four semantics for the extended framework, ranging in the operational, declarative and denotational types. Three of these semantics are inspired by the traditional logic programming paradigm. They consist of the operational semantics O_d, based on the derivation relation $\vdash$, and of the declarative semantics $Decl_m$ and $Decl_f$, based on model theory and fixed-point theory, respectively. The other semantics, namely the denotational semantics Den, is issued from the imperative tradition, and, more particularly, from its metric semantic branch ([6], [5], [14], ...). It describes computations, in a compositional way, via histories, possibly including hypotheses.

All these semantics have been related throughout the paper, thanks to propositions, 15, 16 and 34. The minimal relations have only been stated. From them, it is possible to deduce other relations, for instance to connect Den with $Decl_m$ and $Decl_f$. It is furthermore impossible to add nonredundant relations. For instance, it is impossible to guess the infinite derivations contained in Den in view of the only computed substitutions of O_d. It is also impossible to guess the substitutions computed in O_d from all substitutions pointed out declaratively in $Decl_m$ or $Decl_f$. However, it is worth noting that although they are associated with different semantics, it is possible to connect the derivation relation $\vdash$ and the model theory, as established by proposition 16.

The ELP language introduced in this paper provides a suitable mechanism to introduce synchronicity in concurrent logic programming and to combine, to some extend, logic programming and object-oriented programming. Our future research, under development, will be concerned with more elaborated versions, including, for instance, more object-oriented constructs. Also, we are trying to develop semantics closer to real computations in treating and-parallelism in a non-interleaving way and or-parallelism not just as non-deterministic choice.

Acknowledgments

The research reported herein has been partially supported by Esprit BRA 3020 (Integration). The first author likes to thank the members of the C.W.I. concurrency group, J.W. de Bakker, F. de Boer, F. van Breugel, A. de Bruin, E. Horita, P. Knijnenburg, J. Kok, E. Marchiori, C. Palamidessi, J. Rutten, D. Turi, E. de Vink and J. Warmerdam, for their weekly intensive discussions. The second author wish to thank the Instituto Nacional de Investigação Científica and the Junta Nacional de Investigação Científica e Tecnológica for partial support.

References

[1] P. America and J.J.M.M. Rutten. Solving reflexive domain equations in a category of complete metric spaces. *Journal of Computer and System Sciences*, 39(3):343–375, 1989.

[2] J.-M. Andreoli and R. Pareschi. Linear Objects: Logical Processes with Built-in Inheritance. In D.H.D. Warren and P. Szeredi, editors, *Proc. 7th Int. Conf. on Logic Programming*, pages 495–510, Jerusalem, Israel, 1990. The MIT Press.

[3] A. Brogi. And-Parallelism without Shared Variables. In D.H.D. Warren and P. Szeredi, editors, *Proc. 7th Int. Conf. on Logic Programming*, pages 306–321, Jerusalem, Israel, 1990. The MIT Press.

[4] J.S. Conery. Logical Objects. In R.A. Kowalski and K.A. Bowen, editors, *Proc. 5th Int. Conf. and Symp. on Logic Programming*, pages 420–434, Seattle, USA, 1988. The MIT Press.

[5] J.W. de Bakker. Comparative Semantics for Flow of Control in Logic Programming without Logic. Technical Report CS-R8840, Centre for Mathematics and Computer Science (CWI), Amsterdam, The Netherlands, 1988.

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

[7] F.S. de Boer and C. Palamidessi. On the Asynchronous Nature of Communication in Concurrent Logic Languages: a Fully Abstract Model based on Sequences. In J.C.M. Baeten and J.W. Klop, editors, *Proc. of Concur 90*, volume 458 of *Lecture Notes in Computer Science*, pages 99–114, Amsterdam, The Netherlands, 1990. Springer-Verlag.

[8] P. Degano and S. Diomedi. A First Order Semantics of a Connective Suitable to Express Concurrency. In *Proc. 2nd Workshop on Logic Programming*, pages 506–517, Albufeira, Portugal, 1983.

[9] R. Engelking. *General Topology*. Heldermann Verlag, 1989.

[10] M. Falaschi, G. Levi, and C. Palamidessi. A Synchronization Logic: Axiomatics and Formal Semantics of Generalized Horn Clauses. *Information and Control*, 60:36–69, 1984.

[11] J.Y. Girard. Linear Logic. *Theoretical Computer Science*, 50:1–102, 1987.

[12] S. Gregory. *Design, Application and Implementation of a Parallel Logic Programming Language*. PhD thesis, Department of Computing, Imperial College, London, Great-Britain, 1985.

[13] J.-M. Jacquet. *Conclog: a Methodological Approach to Concurrent Logic Programming*. PhD thesis, Facultés Universitaires Notre-Dame de la Paix, University of Namur, Belgium, 1989. to appear as Lecture Notes in Computer Science, Springer-Verlag.

[14] J.N. Kok and J.J.M.M. Rutten. Contractions in Comparing Concurrency Semantics. *Theoretical Computer Science*, 76:179–222, 1990.

[15] J.W. Lloyd. *Foundations of logic programming*. Springer-Verlag, second edition, 1987.

[16] L. Monteiro. An Extension to Horn Clause Logic allowing the Definition of Concurrent Processes. In *Proc. Formalization of Programming Concepts*, volume 107 of *Lecture Notes in Computer Science*, pages 401–407. Springer-Verlag, 1981.

[17] L. Monteiro. A Horn Clause-like Logic for Specifying Concurrency. In *Proc. 1st Int. Conf. on Logic Programming*, pages 1–8, 1982.

[18] C. Palamidessi. Algebraic Properties of Idempotent Substitutions. In M.S. Paterson, editor, *Proc. of the 17th International Colloquium on Automata, Languages and Programming*, volume 443 of *Lecture Notes in Computer Science*, pages 386–399, Warwick, England, 1990. Springer-Verlag.

[19] V.A. Saraswat. *Concurrent Constraint Programming Languages*. PhD thesis, Carnegie-Mellon University, 1989. To be published by The MIT Press.

[20] E.Y. Shapiro. A Subset of Concurrent Prolog and its Interpreter. Technical Report TR-003, Institute for New Generation Computer Technology (ICOT), Tokyo, 1983.

[21] K. Ueda. *Guarded Horn Clauses*. PhD thesis, Faculty of Engineering, University of Tokyo, Tokyo, Japan, 1986.