PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1988,
edited by ICOT.

Finite failures and partial computations in concurrent logic languages

Moreno Falaschi and Giorgio Levi

Dipartimento di Informatica, Università di Pisa
Corso Italia 40, 56125 Pisa, Italy

ABSTRACT

This paper investigates some semantic properties of concurrent logic languages. Concurrent logic languages with a commit operator differ from pure logic languages in the finite failure set. Since the new finite failure set depends on the computation rule, we define an operational semantics based on a parallel computation rule and a fair search rule. Failures with a parallel computation rule are shown to be strongly related to partial computed answer substitutions. The set of partial computed answer substitutions and the set of finite failures are defined as the least fixpoint of a transformation on pairs of interpretations.

1 INTRODUCTION

In this paper, we are interested in some issues concerning the commit operator and its semantics. The commit operator is a primitive which was first introduced in the relational parallel language (Clark and Gregory 1981) and is present in many concurrent languages, such as PARLOG (Clark and Gregory 1986), Concurrent Prolog (Shapiro 1983) and GHC (Ueda 1985). The mechanism of committing was explicitly designed for concurrent logic languages to allow nondeterminism control. As already noted in other papers (Saraswat 1986), the commit operator strongly affects the semantics. In particular, the notions of success and finite failure sets (Lloyd 1984) must be reconsidered in the case of logic programs with the commit operator. As noted in (Takeuchi 1987), the intersection between these two sets is in general non empty. Some previous studies on concurrent logic languages (Maher 1987, Saraswat 1986, Saraswat 1987a) aimed at the definition of variations of the basic commit operator which preserve a logical meaning to the underlying language. This was achieved by imposing strong constraints on the semantically characterized language (almost deterministic programs in (Maher 1987)), or by changing the nature of the commit operator, (as in the case of the *don't know commit* in (Saraswat 1986)). The standard commit operator was only characterized operationally. For example, Saraswat (1987a, 1987b) defines a structured operational semantics, which also takes into account the mechanisms of synchronization of Concurrent Prolog and GHC. In this paper, we try to preserve the original commit operator and to study the corresponding modified semantics. We show that failures depend on the computation rule and that the standard notion of model theoretic semantics and the negation-as-failure rule (Clark 1978) do not make sense any more. Then, we specify a parallel computation rule and characterize the "enlarged" set of finite failures due to the commit operator.

2 COMMITTED-CHOICE LOGIC LANGUAGES

Committed-choice logic languages are based on Horn Clause Logic (HCL), extended with the commit operator (Clark and Gregory 1981, Shapiro 1983, Shapiro 1986, Clark and Gregory 1986, Ueda 1985), which was introduced in concurrent logic languages. We do not consider the other relevant extension of concurrent logic languages, i.e. synchronization (such as explicit mode declarations or read-only variables). Levi and Palamidessi (1987) proposed an approach to the semantics of synchronization and Levi (1988) proposed a different approach to the synchronization mechanism of Flat GHC. We are currently investigating on the possibility to combine our approach and the one in (Levi and Palamidessi 1987) to obtain a semantics for both the synchronization and the control mechanisms.

In the following we assume, for a given program, F to be the set of functors (with definite arity), denoted by a, b, c,..., P to be the set of predicates (with definite arity), denoted by p, q, r,..., and X to be a denumerable set of variables, x, y, z, ... Constants are 0-adic functors.

The set of *terms* T is the minimal set such that

i) $X \subseteq T$

ii) $\forall\, c \in F$, c n-adic, $\forall\, t_1,...,t_n \in T$, $c(t_1,...,t_n) \in T$

A *substitution* ϑ is a mapping from X to T. An *atom* is a formula $p(t_1,...,t_n)$ where $p \in P$ and $t_1,...,t_n \in T$.

A *program* is a finite set of *clauses* of the following form

$$H \leftarrow G_1,...,G_m \mid B_1,...,B_n \quad (m, n \geq 0)$$

where H (head), the G_i's (guards) and the B_j's (body processes) are atoms. "|" is the commit operator. The predicates which appear in the guards are assumed to be defined by standard Horn clauses (without commit).

The *corresponding pure logic program* is obtained simply by replacing "|" with "," and by interpreting the G_i's and the B_j's as standard atoms.

A *goal statement* (*goal*) is a clause of the form $\leftarrow B_1,...,B_n$ $(n \geq 0)$. If n = 0, the goal is the *empty clause* (denoted by $\square$).

The operational semantics of committed-choice

languages is the same of pure logic languages, apart from the following rule (rule of *commitment* (Ueda 1987)):

> When some clause C called by a goal G succeeds in solving its guard, clause C tries to be selected for the execution of G. To be selected, C must first confirm that no other clauses have been selected for G. If this is the case, C is selected and the execution of G commits to C.

Pure commitment (without synchronization) does not affect the success set semantics (Maher 1987), provided that we assign the success set a different meaning. In fact, any answer substitution, computed in the corresponding pure HCL program, is potentially computed in the committed-choice program, where, however, we are not guaranteed that the substitution will always be computed, since the program execution could commit to a different path in the execution tree. This is the reason why the commitment does instead strongly affect the finite failure set, which can have a non-empty intersection with the success set (Takeuchi 1987). A good semantic characterization of a committed-choice program cannot be based on the success set only. In fact, a query $\leftarrow p(x)$ can terminate either with one (and only one) answer which belongs to the success set, or with the answer "$p(x)$ fails" even if the success set contains instances of $p(x)$.

Example 2.1.

1) $Mod(x,x) \leftarrow x \geq 0 \mid$ 2) $Mod(x,-x) \leftarrow x \leq 0 \mid$ ♦

This program computes (second argument of *Mod*) the absolute value of an integer (first argument of *Mod*). The choice between (1) and (2) is guided from the guards. If the computation commits to clause (1), the first argument is certainly positive and the computation terminates successfully.

Example 2.2.

1) $Mod(x,x) \leftarrow \mid x \geq 0$ 2) $Mod(x,-x) \leftarrow \mid x \leq 0$ ♦

In this case, it is possible, because of the empty guards, to commit to clause (1) even if the first argument is negative. In such a case, the computation fails and clause (2) will never be considered.

The following example shows that the set of finite failures depends upon the computation rule.

Example 2.3.

1) $p(x) \leftarrow \mid q(x,y), r(y).$ 2) $r(s(x)) \leftarrow \mid.$
3) $q(f(x),s(a)) \leftarrow \mid.$ 4) $q(f(x),g(x)) \leftarrow \mid.$ ♦

The goal $\leftarrow p(x)$, with a rightmost computation rule, (where $r(y)$ is reduced before $q(x,y)$), has one possible successful computation only. For the same goal with a leftmost computation rule (which forces to reduce $q(x,y)$ first), we have two possibilities: either clause 3) is choosen and the computation terminates successfully, or clause 4) is choosen and the computation terminates with a failure.
The following example shows that in general the finite failure of a non ground goal does not imply the failure of the goal instances.

Example 2.4.

1) $p(x) \leftarrow \mid q(x).$ 2) $q(a) \leftarrow \mid r.$
3) $q(b) \leftarrow \mid.$ ♦

$\leftarrow p(x).$ has a finitely failing derivation (obtained by committing to clauses (1) and (2)), yet $\leftarrow p(b).$ has no finitely failing derivations.

Our simple examples suggest the following remarks.

a) A committed computation can terminate with a failure even if the goal has instances in the success set.

b) Failures are affected by the computation rule.

c) The negation-as-failure rule is not sound for finite failure. In fact, in our example 2.3, $\leftarrow p(f(a))$ can fail, yet $\neg p(f(a))$ cannot be considered a logical consequence of (some sound logical extension) of the program, since $p(f(a))$ is a logical consequence of the corresponding pure logic program.

d) It is not possible to infer the existence of a failure for a goal G from the existence of failures for a goal G', such that G is an instance of G'.

The problem of (finite) failures, for a given computation rule, consists in characterizing the atoms for which there exists at least one (finitely) failing reduction path. This problem will be considered in the following, in the framework of a specific (parallel) computation rule.

3 OPERATIONAL SEMANTICS

We now introduce some general definitions which extend the classical corresponding notions to our framework. The computation rule is AND-parallelism, defined as any possible interleaving of the literals in the goal.

Definition 3.1 Let W be a program and let $S_i \equiv \leftarrow A_1,\ldots,A_n$ be a goal statement. If

(a) $\exists\, B \leftarrow G_1,\ldots,G_t \mid B_1,\ldots,B_k \in W$ and $\exists\, mgu\ \vartheta$ between B and A_j $(1 \leq j \leq n)$

(b) $\leftarrow (G_1,\ldots,G_t)\vartheta$ has a standard refutation computing an answer substitution ϑ'

(c) $S_{i+1} \equiv \leftarrow (A_1,\ldots,A_{j-1},B_1,\ldots,B_k,A_{j+1},\ldots,A_n)\ \vartheta\vartheta'$

then S_{i+1} is a *potential reduction* of S_i using $mgu\ \vartheta$ and the computed answer substitution ϑ' for the guard $G_1,\ldots,G_t$. The notation $S_i \mapsto^{\vartheta} S_{i+1}$ will denote that S_{i+1} is a potential reduction of S_i, by substitution ϑ. If for some $n \geq 0$ $S \equiv S_0 \mapsto^{\vartheta_1} S_1 \mapsto^{\vartheta_2} \ldots \mapsto^{\vartheta_n} S_n \equiv T$ and $\vartheta = \vartheta_1 \cdot \vartheta_2 \cdot \ldots \cdot \vartheta_n$, where $\cdot$ is the usual composition on substitutions, the relation: $S \mapsto^{\vartheta *} T$ holds. ♦

Definition 3.2 Let W be a program and $S \equiv \leftarrow A_1,\ldots,A_n$ be a goal statement. A *potential derivation* for S in W is a (finite or infinite) sequence of potential reductions starting from S and using (variants of) the clauses of W. ♦

We sometimes use a notation which points out the clauses applied in the derivation. For example $S \xrightarrow[C_1\ldots C_m]{\vartheta}{}^{*} T$ means that the potential derivation for S has length m and uses the clauses $C_1\ldots C_m$.

Definition 3.3 A goal statement $\leftarrow A_1,\ldots,A_n$ has a *potential refutation* (in W) iff there exists a derivation

$$A_1,\ldots,A_n \mapsto^{\vartheta *} \square$$

If this is the case, a potential derivation of $\leftarrow A_1,\ldots,A_n$ in W terminates successfully and ϑ is the computed answer substitution. ♦

Definition 3.4 An atom A is *not rewritable* if there is no clause in W, whose head is unifiable with A. An atom A is *not reducible* if it is not rewritable or, for every clause $H \leftarrow G \mid B \in W$ for which there exists an *mgu* ϑ between H and A, $G\vartheta$ has no refutation. ♦

Definition 3.5 Let W be a program and $S \equiv \leftarrow A_1,\ldots,A_n$ be a goal statement. A finite potential derivation for S in W can succeed (potential refutation) or fail. A *failed* derivation is one which ends in a non empty goal containing an atom which is not *rewritable*. A *guarded failed* derivation is one which ends in a non empty goal containing an atom which is not *reducible*. ♦

In our definition we do not consider infinite computations as failures.

From the above definitions it is easy to see that the only difference between the concept of *potential derivation*, *potential refutation*, etc... and the conventional SLD ones (Apt and van Emden 1982) is that the use of a clause for a reduction in a committed-choice language requires a preliminary solution of its guard. Definition 3.5 distinguishes two kinds of finite failures for potential derivations. The notion of failed derivation is similar to the standard one, while the notion of guarded failed derivation takes into account possible failures caused by the guards.

The main difference between a committed-choice program and the corresponding standard HCL program, apart from the evaluation of guards, is that every (committed) derivation is, roughly speaking, deterministic. Namely, the commit operator allows only one derivation and backtracking is inhibited. There is still a degree of uncertainty in the choice of the clause to be used for reduction when more guards are satisfied. This kind of non determinism, typical of committed-choice languages, is usually called "don't care" nondeterminism because it is based on the assumption that the specific clause chosen for reduction is not relevant to the search for a solution.

The operational semantics is defined in two parts. The first one corresponds to the success set, while the second one corresponds to the set of finite failures.

If p is a predicate, then the *success set* of p is

$S(p) = \{\ p(t_1,\ldots,t_n) \mid \exists\ t'_1,\ldots,t'_n \in T$ such that $p(t'_1,\ldots,t'_n) \mapsto^{\vartheta *} \square,\ \vartheta(t'_1) = t_1,\ldots,\ \vartheta(t'_n) = t_n\ \}$

where ϑ is the computed answer substitution.

The set of *finitely failing atoms* is

$FF_G(p) = \{\ p(t_1,\ldots,t_n) \mid p(t_1,\ldots,t_n)$ has a guarded failed derivation$\}$.

The actual difference between the standard finite failure set (Apt and van Emden 1982) for Horn clause logic and the new definition of FF_G is, intuitively, based on the use of different quantifiers: In the standard one, the goal has only failed (fair) derivations, whilst in the new one the goal has at least one failed derivation.

Our aim is finding a fixpoint characterization of $FF_G(p)$. We simulate by a bottom-up transformation the behaviour of our parallel computation rule. This relies on some notion of partial computed answer substitution, since failures can occur because of intermediate variable bindings produced by partial (not necessarily successful) derivations. This information cannot be obtained from the standard minimal Herbrand model, because

i) it does not allow to determine the actual computed answer substitutions,

ii) it does not characterize partial derivations.

A solution to the first problem can be found in a different semantics recently proposed for pure logic programs (Falaschi et al. 1988a, 1988b). In the next section we give an overview of the definitions and results which are used in our framework. Section 5 presents a new result, i.e. a solution to the problem of the characterization of the set of partial computed answer substitutions.

4 EXTENDED SEMANTICS FOR STANDARD HCL PROGRAMS

The Herbrand universe U (for a given program) is the set of equivalence classes (*quotient set*) of T with respect to the *variance* equivalence relation:

$t \simeq t'$ iff there exist two substitutions ϑ and γ such that

$$t\vartheta = t' \text{ and } t'\gamma = t\ .$$

The new Herbrand universe is different from the standard one, because terms can contain variables.

For the sake of simplicity, the elements of U will have the same representation of the elements of T. In the following, the elements of U will denote the corresponding equivalence classes; i.e. a representative whose variables are renamed, whenever it is needed, to avoid confusion with other variables. This also holds for any other structure that we will define (Base, Interpretations, etc.).
We define an ordering relation $\leq$ on U:

$t \leq t'$ iff there exists a substitution ϑ from X to U such that $t\vartheta = t'$.

The *Herbrand base* B (for a given program) is the set of all the formulas (equivalence classes with respect to the induced variance relation) $p(t_1,\ldots,t_n)$, where $t_1,\ldots,t_n \in U$ and $p \in P$.

The ordering on U induces an ordering on B.
If $t_1 \leq t'_1 ,...,t_n \leq t'_n$ then $p(t_1,...,t_n) \leq p(t'_1,...,t'_n)$.

S-Herbrand interpretations are defined as subsets of B. Variables in the interpretations allow to treat universally quantified formulas in our models and transformations. Thanks to the introduction of variables in the Herbrand domain, the notion of truth (*S-truth*) can be defined as follows (Falaschi et al. 1988a, 1988b). Let I be an S-interpretation:

- a unit clause $A \leftarrow$ is S-true in I iff A belongs to I,
- a definite clause $A \leftarrow B_1,...,B_n$ is S-true in I iff for every $B'_1,...,B'_n$ belonging to I, if there exists $\vartheta = mgu((B'_1,...,B'_n), (B_1,...,B_n))$, then $A\vartheta$ belongs to I,
- an atom A (possibly not ground) is S-true in I iff $\exists A'$, such that (the equivalence class of) A' belongs to I and $A' \leq A$.

A Herbrand model (for a given program) is a Herbrand interpretation M, such that all the clauses of the program are true in M. As usual, we take the minimal model, S_W, whose existence is proved in (Falaschi et al. 1988a, 1988b), as the model-theoretic semantics of the program.

The set of Herbrand interpretations *Int* of a program W is a complete lattice with respect to set inclusion. B *and* $\varnothing$ are, respectively, the top and the bottom element. The fixpoint semantics is based on the following transformation.

Definition 4.1. Let W be a program. The mapping T_W on the set of S-Herbrand interpretations, associated to W, is defined as follows

$$T_W(I) = \{A' \in B \mid \exists A \leftarrow B_1,...,B_n \text{ in } W, \exists B'_1,...,B'_n \in I, \exists \vartheta = mgu((B'_1,...,B'_n), (B_1,...,B_n)), \text{and } A' = A\vartheta \}$$ ♦

Falaschi et al. (1988a, 1988b) show that

- T_W is continuous. Hence $lfp(T_W) = lub_{n \geq 0} T_W{}^n(\varnothing)$ where $lfp(T_W)$ is the fixpoint semantics of T_W, and $lfp(T_W)$ is defined as the (least) fixpoint semantics of the program W.
- The set of models is the same as the set of interpretations closed with respect to T_W (where an S-interpretation I is closed with respect to T_W iff $T_W(I) \subseteq I$).
- The fixpoint and model-theoretic semantics are equivalent, i.e. $lfp(T_W) = S_W$.
- Each computed (possibly non ground) answer substitution for a goal G can be obtained by unifying the goal G with the atoms contained in S_W.

The (non ground) minimal model S_W has stronger properties than the standard (ground) minimal model. The following two theorems establish that if a (possibly non ground) goal has a refutation, then the computed answer substitutions can be characterized by the most general unifiers of the atoms in the goal with the atoms in the minimal model S_W. These theorems are generalizations of the corresponding ones in the ground case (see theorems 7.1, 7.4 and corollary 7.3 in (Lloyd 1984)).

Given a goal G and a substitution ϑ, $\vartheta|_G$ denotes the restriction of ϑ to the variables of G. $G \vdash\!\!\!\!-\vartheta\!\!\rightarrow^* \square$ specifies that G has an SLD-refutation with computed answer substitution ϑ.

Theorem 4.1. (Strong Soundness) Let W be a program, let G be a goal $\leftarrow A_1,...,A_n$ and assume $G \vdash\!\!\!\!-\vartheta\!\!\rightarrow^* \square$. Then $\exists A'_1,...,A'_n \in S_W$ and $\exists \vartheta' = mgu((A_1,...,A_n), (A'_1,...,A'_n))$ such that $\vartheta'|_G = \vartheta|_G$.

Theorem 4.2. (Strong Completeness) Let W be a program, let G be a goal $\leftarrow A_1,...,A_n$ and assume $\exists A'_1,...,A'_n \in S_W$ and $\exists \vartheta' = mgu((A_1,...,A_n), (A'_1,...,A'_n))$ then $\exists \vartheta$ $G \vdash\!\!\!\!-\vartheta\!\!\rightarrow^* \square$ and $\vartheta'|_G = \vartheta|_G$.

5 PARTIAL ANSWER SUBSTITUTIONS

In (Lloyd 1984) the notion of computed answer substitution depends on the notion of refutation. ϑ is a *computed answer substitution* for the program W and the goal G iff $\exists \vartheta'$ $G \vdash\!\!\!\!-\vartheta'\!\!\rightarrow^* \square$ and $\vartheta = \vartheta'|_G$.The next definition introduces a notion of computed answer substitution that is more general than the one in (Lloyd 1984). This generalization is necessary to give the semantics that will be defined in section 6.

Definition 5.1. Let W be a HCL program and G be a goal. ϑ is a *partial computed answer substitution* for the program $W \cup G$ iff $\exists \vartheta'$ $G \vdash\!\!\!\!-\vartheta'\!\!\rightarrow^* G'$ and $\vartheta = \vartheta'|_G$. ♦

A partial computed answer substitution is given by restricting the composition of the substitutions used in a derivation to the variables of the initial goal. Obviously, a computed answer substitution is a partial computed substitution where the derivation is a refutation.

We propose a method to characterize the set of partial (computed) answer substitutions in a declarative way; analogous to the characterization of the computed answer substitutions given by S_W. This is possible by defining a simple transformation on programs.

Definition 5.2. Let W be a HCL program.

$$Pr(W) = \{P(x) \leftarrow . \mid P \text{ is a n-ary predicate in } W \text{ and } x \text{ is a n-tuple of distinct variables}\}$$

$$Part(W) = W \cup Pr(W)$$ ♦

$Pr(W)$ represents a new set of facts corresponding to predicates appearing in W. $Part(W)$ is the set given by adding the new clauses of $Pr(W)$ to the program W.

Part(W), as it is shown by the following theorems, has the nice property to characterize exactly the set of partial computed answer substitutions of W. Informally, the clauses in Pr(W) allow to transform any (partial) derivation into a refutation. Let us now prove that the set of partial computed answer substitutions for W and Part(W) is the same. We prove this result by showing that, given a goal G, for any derivation in the program W, there exists a corresponding derivation in Part(W) computing the same answer substitution, and vice versa.

Theorem 5.1. Let W be a HCL program and G be a goal. ϑ is a partial computed answer substitution for the program $W \cup G$ iff ϑ is a partial computed answer substitution for the program Part(W)$\cup G$.

Proof. Let us consider first a derivation for G in the program W. The same derivation is also possible in Part(W), being $W \subseteq$ Part(W).

Let us now consider a derivation for G in Part(W). The proof is by induction on the number n of clauses of Pr(W) used in the derivation.

n=0) In this case the derivation uses only clauses which are in W too.

n>0) Assume our derivation is

$$G \vdash\!\!\!\rightarrow^{\vartheta'*} G' \vdash\!\!\!\rightarrow_{C_l}^{\vartheta''} G'' \vdash\!\!\!\rightarrow^{\vartheta'''*} G''' \ .$$

where C_l is the first clause of Pr(W) used in the derivation. Let us now consider the reduction $G' \vdash\!\!\!\rightarrow_{C_l}^{\vartheta''} G''$ and the structure of the substitution ϑ'':

Clearly $\vartheta'' = mgu(P(t),P(x))$ for some predicate $P(t)$ in G' and the corresponding predicate $P(x)$ in Pr(W). By definition of Pr(W), $P(x)\vartheta'' = P(t)$. Thus ϑ'' modifies the values of some new variables introduced in this step of derivation, and these variables do not affect the composition of substitutions. Therefore

$$\vartheta'\vartheta''\vartheta'''|_G = \vartheta'\vartheta'''|_G \qquad (1)$$

Let us now simply consider the derivation $G \vdash\!\!\!\rightarrow^{\vartheta'*} G' \vdash\!\!\!\rightarrow^{\vartheta'''*} G''''$ where $P(t)$ is not rewritten, and therefore $G'''' = G'''$ plus the atom $P(t)$ (possibly more instantiated). This derivation uses n-1 clauses of Pr(W). Hence, by the inductive hypothesis, there exists a derivation for G in W which gives the same answer substitution $\vartheta'\vartheta'''|_G$, and, by (1),

$$\vartheta'\vartheta''\vartheta'''|_G = \vartheta'\vartheta'''|_G. \qquad \blacklozenge$$

In a similar manner we can prove that any partial computed answer substitution in Part(W) is also a computed answer substitution in Part(W).

Theorem 5.2. Let W be a HCL program and G be a goal. ϑ is a partial computed answer substitution for the program $W \cup G$ iff ϑ is a computed answer substitution for the program Part(W)$\cup G$.

Proof. By theorem 5.1 the set of computed answer substitutions of Part(W) is equal to its set of partial answer substitutions. Therefore, it is sufficient to prove that ϑ is a partial computed answer substitution for Part(W) $\cup G$ iff it is a computed answer substitution. (Sketch) Let us consider a derivation $G \vdash\!\!\!\rightarrow^{\vartheta'*} G'$ for G in Part(W). Let $\vartheta'|_G = \vartheta$ and consider a refutation of G' using clauses of Pr(W). $G \vdash\!\!\!\rightarrow^{\vartheta'*} G' \vdash\!\!\!\rightarrow^{\vartheta''*} \square$. Clearly, $\vartheta''|_{G'} = \varepsilon$. Therefore, this refutation computes the answer substitution $\vartheta'\vartheta''|_G = \vartheta'|_G = \vartheta$. $\blacklozenge$

We can now characterize the set of partial computed answer substitution of a program W, simply considering the minimal model of the augmented program $S_{\text{Part}(W)}$. In fact, by theorems 4.1 and 4.2 $S_{\text{Part}(W)}$ characterizes the set of computed answer substitutions of Part(W), and, by theorem 5.2, the set of computed answer substitutions of Part(W) is equal to the set of partial answer substitutions of W.

We are interested, in this paper, to potential derivations in committed-choice programs. The notion of partial computed answer substitution obviously extends to the notion of partial potential computed answer substitution, by simply substituting the definition of potential derivation (definition 3.2) to the standard notion of derivation. Because of our assumption on the guards (i.e. guards are defined by standard Horn clauses), the set of partial potential computed answer substitutions can be generated using the following construction.

Definition 5.3. Let W be a committed-choice program.

$\text{Pr}_G(W) = \{P(x) \leftarrow . \mid$ P is a n-ary predicate appearing in the body or the head of some clause in W and x is a n-tuple of distinct variables$\}$

$\text{Part}_G(W) = W \cup \text{Pr}_G(W)$ $\blacklozenge$

Theorem 5.3. Let W be a committed-choice program and G be a goal. ϑ is a partial potential computed answer substitution for the program $W \cup G$ iff ϑ is a computed answer substitution for the program $\text{Part}_G(W) \cup G$.

Proof. Similar to the case of partial computed answer substitutions for HCL programs. $\blacklozenge$

Thus, given a committed-choice program W, the set of partial potential computed answer substitutions is represented by $S_{\text{Part}_G(W)}$. Moreover, the subset of the (non ground) minimal model corresponding to the predicates appearing in the guards does not change when we augment the clauses to get the partial computed answer substitutions. This depends on the choice to use a different mechanism to evaluate guards, that is to have standard HCL programs for guards. Refutations are the only allowed partial computations for guards.

Example 5.1: Let us consider the program W:

1) $p(x) \leftarrow q(x) \mid r(x)$. 2) $p(b) \leftarrow \mid .$

3) $q(a) \leftarrow$. (Standard HCL clause).

4) $q(c) \leftarrow \mid .$ 5) $r(a) \leftarrow \mid .$

6) $r(b) \leftarrow \mid .$ $\blacklozenge$

The standard (ground) minimal model would be the set { $q(a), q(c), p(a), p(b), r(a), r(b)$ }. The new program for potential partial computations can easily be constructed:

$$Pr(W) = \{(7)\ p(x) \leftarrow !,\ (8)\ r(x) \leftarrow !\}.$$

Thus $Part(W) = W \cup Pr(W)$

The new minimal augmented model is the following:

$T_{Part_G(W)}{}^{1}(\varnothing) = \{p(x), r(x), p(a), p(b), q(a), q(c), r(a), r(b)\}$, by applying clauses (2), (3), (4), (5), (6), (7) and (8).

$T_{Part_G(W)}{}^{2}(\varnothing) = T_{Part_G(W)}(\{p(x), r(x), p(a), p(b), q(a), q(c), r(a), r(b)\}) = \{p(c), p(x), r(x), p(a), p(b), q(a), q(c), r(a), r(b)\}$, where $p(c)$ is added by applying $q(c)$ and $r(x)$ to the clause (1)

$T_{Part_G(W)}{}^{3}(\varnothing) = T_{Part_G(W)}{}^{2}(\varnothing)$

Thus $T_{Part_G(W)}{}^{\omega}(\varnothing) = S_{Part_G(W)} = \{p(c), p(x), r(x), p(a), p(b), q(a), r(a), r(b)\}$. The elements of $S_{Part_G(W)}$ represent the corresponding equivalence classes. Therefore variable names are not meaningful. Note, for example, that $p(c)$ does not belong to the standard minimal model. We can note that, starting from the goal $\leftarrow p(x)$, clause (1) can be applied and the guard has a refutation computing the partial potential answer substitution $\{x=c\}$. The same substitution can be computed by the unification of an atom in $S_{Part_G(W)}$ and the atoms in the query itself. $\{x=c\}$ is the *mgu* of the atom $p(x)$ in the query, and the atom $p(c)$ in $S_{Part_G(W)}$. $S_{Part_G(W)}$ allows to characterize the standard computed answer substitutions for $Part(W)$. For example, by the unification of $p(x)$ and atoms in $S_{Part_G(W)}$, it can be shown that $\leftarrow p(x)$ has four possible computed answer subsitutions, i.e. the empty one, and the substitutions $\{x=a\}$, $\{x=b\}$, $\{x=c\}$.

6 CHARACTERIZATION OF THE FAILING ATOMS

The problem consists in characterizing the set FF_G. Our characterization of FF_G takes into account the notion of partial computed (non ground) answer substitutions, and the related properties. The definitions of the transformations, which are given in the rest of this section, are always related to the non standard definition of Herbrand universe and Herbrand Base, whose terms are possibly non ground. Throughout this section we will refer to the lattice *Int* of S-Herbrand interpretations under set inclusion, which was introduced in section 4. In our mappings on Herbrand interpretations, variables of clauses are implicitely renamed when an *mgu* with some atom is computed. This is possible because we can choose an appropriate representative of the equivalence class with respect to the variance relation and avoid the technical complications due to the possible collisions of variable names. Given a program W, $S_{Part_G(W)}$ will always denote the (non ground) minimal model for partial computations introduced in section 5.

$[C]\lambda : H' \leftarrow B_1', \ldots, B_n'$ denotes the instance $H' \leftarrow B_1', \ldots, B_n'$ of the clause C under the substitution λ.

Let us introduce the relation Ξ. Given a set of atoms $B_1, \ldots, B_k$ and a Herbrand interpretation L, the relation Ξ on $B_1, \ldots, B_k$ and L holds iff there exist (variants of) $B_1', \ldots, B_k' \in L$ such that $(B_1, \ldots, B_k)$ and $(B_1', \ldots, B_k')$ are unifiable with *mgu* λ. $(B_1, \ldots, B_k)\lambda\ \Xi\ L$ denotes that the relation Ξ holds under the substitution λ.

As a first step towards a complete characterization of the atoms for which there exists a finitely failing derivation, let us define a useful transformation which maps Herbrand interpretations to Herbrand interpretations.

Definition 6.1 Let I be a Herbrand interpretation, W be a program, and $S_{Part_G(W)}$ be the minimal model for partial computations.

$T_F(I) = \{H :$

$\neg\exists C \in W$ such that $\exists mgu\ (H, head(C))$

or

$\exists C \in W$ such that $\exists mgu\ (H, head(C)) = \lambda$

and $[C]\lambda : H' \leftarrow B_1', \ldots, B_m' \,|\, B_{m+1}', \ldots, B_n'$

such that $\exists k\ \ m+1 \le k \le n\ B_k'\lambda' \in I$,

and $(B_1', \ldots, B_{k-1}', B_{k+1}', \ldots, B_n')\lambda'\ \Xi\ S_{Part_G(W)}\}$ ♦

Roughly speaking, the condition

$(B_1', \ldots, B_{k-1}', B_{k+1}', \ldots, B_n')\lambda'\ \Xi\ S_{Part_G(W)}$

in the definition of T_F shows that $B_1', \ldots, B_{k-1}', B_{k+1}', \ldots, B_n'$ are involved in a partial computation, and that they are partially reduced computing the binding λ'. Moreover, the condition $m+1 \le k$ corresponds to requiring a refutation of the guards. We are able to handle the bindings computed from partial computations thanks to the minimal model $S_{Part_G(W)}$ we have introduced in section 5.

Proposition 6.1 T_F is continuous

Proof Standard. ♦

Corollary 6.1 $T_F{}^{\omega}(\varnothing) = \bigcup_{n=0}^{\infty} T_F{}^{n}(\varnothing) =$ least fixed point of T_F

Proof Immediate from proposition 6.1. ♦

We now show that the set $T_F{}^{\omega}(\varnothing)$ characterizes the set of finitely failing atoms. We give the characterization for programs W which satisfy the following quite reasonable condition: if an atom A is unifiable with the head of some clauses $H_{i_1} \leftarrow G_{i_1} \,|\, B_{i_1}, \ldots, H_{i_k} \leftarrow G_{i_k} \,|\, B_{i_k}$ in W, then at least one of the corresponding guards $G_{i_1}, \ldots, G_{i_k}$ is satisfiable. Thus we cannot have a failure caused by the unsatisfiability of guards and, consequently, the set of *guarded failed* derivations is equal to the set of *failed* derivations.

Given a goal G, an atom A in G, and a derivation, a successor of A is A or any of the successors of the atoms introduced from the reduction of G in the derivation itself.

Lemma 6.1 Let $G \equiv \leftarrow A_1,\ldots,A_n$ be a goal. Assume G has a derivation

$G \vdash_{C_1} \vartheta_1 \rightarrow G_1 \vdash_{C_2} \vartheta_2 \rightarrow G_2 \vdash_{C_3} \vartheta_3 \rightarrow \ldots \vdash_{C_m} \vartheta_m \rightarrow G_m$

Let $C(A_i)$ denote the clauses in $C_1,\ldots,C_m$ (in the same order) that correspond to the reduction of a successor of A_i. Let $\underline{C}(A_i)$ be the remaining clauses of $C_1,\ldots,C_m$ (in the same order).

Then $G \vdash_{\underline{C}(A_i)} \vartheta' \longrightarrow^* G' \vdash_{C(A_i)} \vartheta'' \longrightarrow^* G'_m$, where G'_m is a variant of G_m.

Proof. It is an easy generalization of the Switching lemma in (Lloyd 1984). ♦

Theorem 6.1 $FF_G = T_F^{\omega}(\varnothing)$.

Proof.

a) Let us prove $FF_G \subseteq T_F^{\omega}(\varnothing)$ first.

(By induction on the number t of clauses applied in the failing path). Let $A \in FF_G$; we prove that $A \in T_F^{\omega}(\varnothing)$.

Assume first that t=0. If $\leftarrow A$ immediately fails, A is clearly not rewritable. By definition of T_F, any atom which is not rewritable, i.e. any atom H for which $\neg\exists C \in W$ such that $\exists mgu(H, head(C))$, belongs to $T_F(I)$ for any I. Thus A belongs to $T_F(T_F^{\omega}(\varnothing)) = T_F^{\omega}(\varnothing)$.

Now assume that the result holds for derivations of length $\leq$ t-1. Assume $H \leftarrow B_1,\ldots,B_m | B_{m+1},\ldots,B_n$ is the clause C_1 applied in the first derivation step and ϑ_1 be the *mgu* of H and A (hence the substitution ϑ computed from the potential reduction is $\vartheta_1\vartheta_2$, i.e. the composition of ϑ_1 and the substitution ϑ_2 computed from the refutation of the guard, see definition 3.1).

Let $\leftarrow A \vdash_{C_1} \vartheta \rightarrow \leftarrow (B_{m+1}\ldots B_k\ldots B_n)\,\vartheta \vdash_{C_2\ldots C_t} \gamma \rightarrow$ $\leftarrow \ldots B\ldots$ be a finitely failing derivation of length t, and B be the atom which is not rewritable. Assume B is a successor of B_k in the derivation. By lemma 6.1,

$\leftarrow (B_{m+1}\ldots B_k\ldots B_n)\vartheta \vdash_{\underline{C}(B_k)} \vartheta' \rightarrow G' \vdash_{C(B_k)} \vartheta'' \rightarrow$ $G'' \equiv \leftarrow \ldots B'\ldots$ where B' is a variant of B (and therefore it is not rewritable).The derivation that goes from $\leftarrow (B_{m+1}\ldots B_k\ldots B_n)\vartheta \equiv G$ to G' is a partial potential one, and it does not involve any successor of B_i. Thus there exists, by theorem 5.3, an mgu σ (that for the guards computes the binding ϑ_2 given by the refutation) between the (n-1)-tuple $(B_1,\ldots,B_{k-1},B_{k+1},\ldots,B_n)\vartheta_1$ and a corresponding (n-1)-tuple in $S_{\mathrm{Part}_G(W)}$,

i.e. $(B_1\vartheta_1,\ldots,B_{k-1}\vartheta_1,B_{k+1}\vartheta_1,\ldots,B_n\vartheta_1)\sigma \Xi S_{\mathrm{Part}_G(W)}$, such that $\sigma_{|G} = \vartheta_2\vartheta'_{|G}$. Hence $(B_{m+1}\ldots B_k\ldots B_n)\,\vartheta\vartheta' = (B_{m+1}\ldots B_k\ldots B_n)\,\vartheta_1\vartheta_2\vartheta'_{|G} = (B_{m+1}\ldots B_k\ldots B_n)\,\vartheta_1\sigma_{|G}$,

and therefore, $B_k\vartheta\vartheta' = B_k\vartheta_1\sigma$ (1)

Moreover, by lemma 6.1,

$$B_k\vartheta\vartheta' \vdash_{C(B_k)} \vartheta''' \longrightarrow G''' \equiv \leftarrow \ldots B'\ldots$$

(G''' is in general different from G''), i.e. G''' contains an atom which is not rewritable. Thus $B_k\vartheta\vartheta'$ is the root of a failing path of length $\leq$t-1, and therefore, by the inductive hypothesis,

$$B_k\vartheta\vartheta' \in T_F^{\omega}(\varnothing) \qquad (2)$$

Let $\lambda = \vartheta_1$, and $[C]_{\vartheta_1}: H\vartheta_1 \leftarrow (B_1,\ldots,B_n)\vartheta_1$.

$(B_1\vartheta_1,\ldots,B_{k-1}\vartheta_1,B_{k+1}\vartheta_1,\ldots,B_n\vartheta_1)\sigma \Xi S_{\mathrm{Part}_G(W)}$,

m+1$\leq$k, and, by (1) and (2) $B_k\vartheta_1\sigma = B_k\vartheta\vartheta' \in T_F^{\omega}(\varnothing)$. Thus, by definition of T_F, any atom whose *mgu* with H is ϑ_1 belongs to $T_F(T_F^{\omega}(\varnothing))$. Hence

$$A \in T_F(T_F^{\omega}(\varnothing)) = T_F^{\omega}(\varnothing).$$

b) Let us prove now that $T_F^{\omega}(\varnothing) \subseteq FF_G$.

Assume that A belongs to $T_F^{\omega}(\varnothing)$. Then A belongs to $T_F^{t}(\varnothing)$, for some t $\in$ N. We prove the assertion by induction on t:

Assume first that t=1. Then $A \in T_F^{1}(\varnothing)$ means that $\neg\exists C \in W$ such that $\exists mgu\,(A, head(C))$. Hence A has a finitely failing derivation.

Now assume that the result holds for t-1. Assume $A \in T_F^{t}(\varnothing)$. Then

(a) $\neg\exists C \in W$ such that $\exists mgu(A, head(C))$

or

(b) $\exists C \in W$ such that $\exists mgu\,(A, head(C)) = \lambda$ and

$[C]_\lambda: H' \leftarrow B_1',\ldots,B_m' | B_{m+1}',\ldots,B_n'$ such that

$\exists$k m+1$\leq$k$\leq$n $B_k'\lambda' \in T_F^{t-1}(\varnothing)$ and

$(B_1',\ldots,B_{k-1}',B_{k+1}',\ldots,B_n')\lambda' \Xi S_{\mathrm{Part}_G(W)}$.

Case (a) is obvious. Let us consider the other case. $\exists C \in W$ such that $\exists mgu\,(A, H') = \lambda$ means that a first reduction could be applied to A, under the condition that its guard has a refutation. Now, according to the definition of relation Ξ and by theorem 5.3, if

$$(B_1',\ldots,B_{k-1}',B_{k+1}',\ldots,B_n')\lambda' \Xi S_{\mathrm{Part}_G(W)}$$

then there exists a partial derivation for

$$\leftarrow B_1',\ldots,B_{k-1}',B_{k+1}',\ldots,B_n',$$

which corresponds to a refutation of the atoms in the guard, and that, if restricted to the variables of $(B_1',\ldots,B_{k-1}',B_{k+1}',\ldots,B_n')$, computes the substitution λ'. Thus starting from A, by means of a derivation, we reach a goal containing the atom $B_k'\lambda' \in T_F^{t-1}(\varnothing)$ (m+1$\leq$k$\leq$n), which, by the inductive hypothesis, has a finitely failing derivation ♦

$T_F^{\omega}(\varnothing)$ can also be used to characterize finite failures of goal statements, as it is shown by the next theorem.

Theorem 6.2 Let W be a program and $G \equiv \leftarrow A_1,\ldots,A_n$ be a goal statement. Then G finitely fails iff $\exists$k $1 \leq k \leq n$ such that $(A_1,\ldots,A_{k-1},A_{k+1},\ldots,A_n)\lambda' \Xi S_{\mathrm{Part}_G(W)}$ and $A_k\lambda' \in T_F^{\omega}(\varnothing)$.

Proof. By definition of FF_G, an atom A can finitely fail iff $A \in FF_G$. Therefore a goal statement G can finitely fail iff, possibly after a partial computation, one of the atoms in the (derived) goal statement belongs to FF_G. Thus, by theorem 5.3, and by lemma 6.1 with arguments similar to

the proof of theorem 6.1, $\leftarrow A_1,\ldots,A_n$ can finitely fail iff $\exists k\ 1\leq k\leq n$ such that $(A_1,..,A_{k-1},A_{k+1},..,A_n)\lambda' \Xi S_{\mathrm{Part_G}(W)}$ and $A_k\lambda' \in FF_G$. Hence, by theorem 6.1, $A_k\lambda' \in FF_G$ iff $A_k\lambda' \in T_F{}^{\omega}(\varnothing)$. ♦

The problem with the transformation T_F is that it assumes to start with the set $S_{\mathrm{Part_G}(W)}$.

However we can use a general property of lattices (see theorem 6.1 in (Stoy 1977)) which allows to construct new lattices in a natural way, by simply making cartesian products of them and inheriting the corresponding partial orderings. Namely, given the complete lattices $(L_1,\leq_1)$, $(L_2, \leq_2),\ldots,(L_n, \leq_n)$, the cartesian product $L_1 \times\ldots\times L_n$ with the induced partial order on its elements $(x_1,\ldots,x_n) \leq (y_1,\ldots,y_n)$ iff $x_i \leq_i y_i\ \forall$ i (i=1,...,n) is a complete lattice too and is usually called *product*. It is clear that we can construct the *product* by making the cartesian product of the same lattice. We use this construction with the lattice *Int*, and define a transformation which maps pairs of Herbrand interpretations to pairs of Herbrand interpretations which allows to effectively compute the set $T_F{}^{\omega}(\varnothing)$.

Definition 6.2 Let (I, J) be a pair of Herbrand interpretations belonging to the lattice *product Int* $\times$ *Int*, let W be a program

$T_G(I, J)=$
({H:
$\neg\exists C \in W$ such that $\exists mgu\ (H, head(C))$
or
$\exists C \in W$ such that $\exists mgu\ (H, head(C)) = \lambda$
and $[C]\lambda$: $H' \leftarrow B_1',\ldots,B_m' \| B_{m+1}',\ldots,B_n'$
such that $\exists k\ B_k'\lambda' \in I$, m+1≤k≤n
and $(B_1',\ldots,B_{k-1}',B_{k+1}',\ldots,B_n')\lambda' \Xi J$ },
{A':
$\exists A \leftarrow B_1,\ldots,B_n$ in $\mathrm{Part_G}(W)$,
$\exists B'_1,\ldots,B'_n \in J$,
$\exists \vartheta = mgu((B'_1,\ldots,B'_n), (B_1,\ldots,B_n))$,
and $A' = A\vartheta$ }) ♦

We can note that the definition of the first component is similar to that for T_F. The only difference consists in the interpretation J (the second argument of the function), which replaces the set $S_{\mathrm{Part_G}(W)}$. The definition of the second component is the same given for the transformation $T_{\mathrm{Part_G}(W)}$ in section 5. It only depends on the second argument of T_G.

Proposition 6.2. T_G is a continuous mapping.

Proof The proof of continuity can be given by simply proving the continuity of T_G with respect to its components separately. This is, in fact, a general property of the *product* of lattices with the naturally induced order (see theorem 6.2 in (Stoy 1977)).

As we have already noted, the definition of the second component does not depend on the first argument of T_G. Thus it is equivalent to the definition of the function $T_{\mathrm{Part_G}(W)}$ that is continuous.

With a given J the continuity of the first component is analogous to the one for proposition 6.1. ♦

Let $(X, Y)|_i$ indicate the i-th element of the ordered pair (X, Y), i=1, 2. Thus $(X, Y)|_1 = X$ and $(X, Y)|_2 = Y$.

Proposition 6.3. The least fixed point of T_G is

$$T_G{}^{\omega}(\varnothing, \varnothing) = \cup_{k\in N}(T_G{}^k(\varnothing, \varnothing)|_1, T_G{}^k(\varnothing, \varnothing)|_2)$$

Proof The proof follows from the fact that the *product* *Int* × *Int* of subsets of the Herbrand base with set inclusion is a complete lattice and from proposition 6.2. ♦

We now show that the function T_G allows to compute both the minimal model $S_{\mathrm{Part_G}(W)}$ and the set of finitely failing atoms FF_G.

Theorem 6.3. Let W be a committed-choice program, then

$$T_G{}^{\omega}(\varnothing, \varnothing) = (T_F{}^{\omega}(\varnothing), T_{\mathrm{Part_G}(W)}{}^{\omega}(\varnothing))$$

Proof The equality $T_G{}^{\omega}(\varnothing, \varnothing)|_2 = T_{\mathrm{Part_G}(W)}{}^{\omega}(\varnothing)$ is an immediate consequence of the equivalence of the definition of the second component of T_G and the definition of the function $T_{\mathrm{Part_G}(W)}$. We also have the stronger property $\forall n \in N\ T_{\mathrm{Part_G}(W)}{}^n(\varnothing) = T_G{}^n(\varnothing, \varnothing)|_2$.

For the second part we must prove that $T_G{}^{\omega}(\varnothing, \varnothing)|_1 \subseteq T_F{}^{\omega}(\varnothing)$ and that $T_F{}^{\omega}(\varnothing) \subseteq T_G{}^{\omega}(\varnothing, \varnothing)|_1$. The first inclusion $T_G{}^{\omega}(\varnothing, \varnothing)|_1 \subseteq T_F{}^{\omega}(\varnothing)$ is obvious. Therefore, it is sufficient to prove that

$$T_F{}^{\omega}(\varnothing) \subseteq T_G{}^{\omega}(\varnothing, \varnothing)|_1.$$

This corresponds to proving that, for any m,

$$T_F{}^m(\varnothing) \subseteq T_G{}^{\omega}(\varnothing, \varnothing)|_1 \text{ (by induction on m).}$$

Assume first that m=1. By definition of T_F, $H \in T_F{}^1(\varnothing)$ iff $\neg\exists C \in W$ such that $\exists mgu\ (H, head(C))$. Then, by definition of T_G, and by proposition 6.3,

$$H \in T_G{}^1(\varnothing, \varnothing)|_1 \subseteq T_G{}^{\omega}(\varnothing, \varnothing)|_1.$$

Now assume that the result holds for m-1. Assume $H \in T_F{}^m(\varnothing)$. If $\neg\exists C \in W$ such that $\exists mgu\ (H, head(C))$ the case is similar to the previous one. Otherwise, there exists a clause $C \in W$ such that $\exists\ mgu\ (H, head(C)) = \lambda$ and $[C]\lambda: H' \leftarrow B_1',\ldots,B_m' \| B_{m+1}',\ldots,B_n'$ such that $(B_1',\ldots,B_{k-1}',B_{k+1}',\ldots,B_n')\lambda' \Xi S_{\mathrm{Part_G}(W)}$ m+1≤k and $B_k'\lambda' \in T_F{}^{m-1}(\varnothing)$.

If $(B_1',\ldots,B_{k-1}',B_{k+1}',\ldots,B_n')\lambda' \Xi S_{\mathrm{Part_G}(W)}$ then there exists t' such that

$$(B_1',\ldots,B_{k-1}',B_{k+1}',\ldots,B_n')\lambda' \Xi T_{\mathrm{Part_G}(W)}{}^{t'}(\varnothing), \text{ with}$$

$$T_{\mathrm{Part_G}(W)}{}^{t'}(\varnothing) = T_G{}^{t'}(\varnothing, \varnothing)|_2.$$

Moreover, by the inductive hypothesis,

$$T_F{}^{m-1}(\varnothing) \subseteq T_G{}^{\omega}(\varnothing, \varnothing)|_1.$$

Thus, by proposition 6.3, there exists t'' such that $B_k'\lambda' \in T_G{}^{t''}(\varnothing, \varnothing)|_1$. By monotonicity of T_G, and

assuming $t = max(t',t'')$, we obtain that

$$H \in T_G(T_G{}^t(\varnothing, \varnothing)|_1, T_G{}^t(\varnothing, \varnothing)|_2)|_1.$$

From $T_G{}^{t+1}(\varnothing, \varnothing) \subseteq T_G{}^{\omega}(\varnothing, \varnothing)$ we derive the theorem. ♦

Corollary 6.2. $FF_G = T_F{}^{\omega}(\varnothing) = T_G{}^{\omega}(\varnothing, \varnothing)|_1$ and $S_{Part_G(W)} = T_{Part_G(W)}{}^{\omega}(\varnothing) = T_G{}^{\omega}(\varnothing, \varnothing)|_2$.

Proof The first equivalences derive from theorems 6.1 and 6.3. The second equivalences derive from the theorems summarized in section 5 and theorem 6.3. ♦

We can see how the transformation T_G works on a simple example

Example 6.1.

1) $p(x) \leftarrow | q(x,y), r(y).$ 2) $q(a,b) \leftarrow |.$
3) $q(x,a) \leftarrow |.$ 4) $r(b) \leftarrow |.$ ♦

We will only comment the component of failures, while for the other component the considerations of section 5 are still valid.

$T_G{}^1(\varnothing, \varnothing) = (\ \{r(a), q(b,b)\},\ \{q(x,a), q(a,b), r(b), q(x,y), r(x), p(x)\ }\ )$, since $r(a)$ and $q(b,b)$ are the only atoms which are not rewritable.

$T_G{}^2(\varnothing, \varnothing) = (\{r(a),\ p(x), p(a), p(b), q(b,b)\},\ \{q(x,a), q(a,b), r(b), q(x,y), r(x), p(a), p(x)\})$. In fact, in this step, clause (1) can be applied with the atom $p(a)$ obtaining

$[\text{Clause (1)}]_{\lambda}: p(a) \leftarrow q(a,y), r(y)$ where $\lambda = \{\ x = a\ \}$

and by $(\ q(a,y))\lambda' \ \Xi\ \{q(x,a), q(a,b), r(b), q(x,y), r(x), p(a), p(x)\ }$ we obtain, unifying $q(a,y)$ and $q(x,a)$ the substitution $\lambda' = \{\ y = a, x = a\ \}$ and then $r(y)\ \lambda' = r(a) \in T_G{}^1(\varnothing,\varnothing)|_1$. Thus $p(a) \in T_G{}^2(\varnothing, \varnothing)|_2$.
Similarly we can prove that $p(x)$ and $p(b)$ belong to $T_G{}^2(\varnothing, \varnothing)|_2$.
The contribution to the set of failures coming from the second component is, therefore, quite important in this example.

$T_G{}^3(\varnothing, \varnothing) = T_G{}^2(\varnothing, \varnothing)$. Hence $T_G{}^2(\varnothing, \varnothing)$ is the least fixed point of T_G. It is easy to verify that the set of finitely failing atoms is in this example exactly $\{r(a),\ p(x), p(a), p(b)\}$. For example, we have a finitely failing derivation for $p(x)$ using clause (1) first and then clause (3). Similarly for $p(a)$ and $p(b)$.

We can justify our condition on guard satisfiability, by reconsidering example 5.1.
Respect to the program W in example 5.1, the goal $\leftarrow p(b)$ can be reduced either by clause (1) or by clause (2). If the interpreter tries clause (1), then the goal $\leftarrow q(b)$, corresponding to the guard, must have a refutation before applying clause (1). $\leftarrow q(b)$ fails and, therefore, clause (1) cannot be applied. Clause (2) immediately gives a success. Thus $\leftarrow p(b)$ cannot fail. The mapping T_G works correctly. However, let us consider a slight variation of example 5.1, which does not contain clause (2), and, therefore, does not satisfy our condition on guards.

Example 6.2.

1) $p(x) \leftarrow q(x) | r(x).$
2) $q(a) \leftarrow .$ (Standard HCL clause).
3) $r(a) \leftarrow |.$ 4) $r(b) \leftarrow |.$ ♦

Let us consider the same goal $\leftarrow p(b)$. It can be reduced by clause (1) only. Hence the goal $\leftarrow q(b)$, corresponding to the guard, should have a refutation before applying clause (1). $\leftarrow q(b)$ fails and, therefore, clause (1) cannot be applied and $\leftarrow p(b)$ fails. T_G does not work correctly. In fact, $q(b)$ is not generated.

In example 5.1, the guard of clause (1) fails, but there exists an alternative (clause (2)) which can be applied. In example 6.2, instead, there are no alternatives to the application of clause (1) and, therefore, the failure of the guard causes the failure of the whole computation. Thus, a failure in a guard brings to a failure for the whole computation only if there are no alternatives to reduce the atom by some other clause. Finite failure becomes a global property of the clauses whose heads are unifiable with the atom to be reduced.

7 CONCLUSIONS

In this paper we have shown how finite failures can be defined and generated by a bottom-up construction in the case of committed-choice programs with guards satisfying a condition on satisfiability. In the most general case, the generation of finite failures with a construction similar to that shown in this paper becomes more complex. We should be able to state something about not only positive information (i.e. refutations) on the HCL program corresponding to the atoms in the guards, but also about the set of (non ground) finite failures for this program, i.e. a generalization of the concept of finite failures in the case of standard HCL programs (see (Apt and van Emden 1982, Lloyd 1984)). On the other hand, usually, in real logical concurrent languages, guards satisfy severe constraints. Therefore we believe that knowing something about the negative information of the programs corresponding to guards can be feasible, and we are currently investigating on the concept of (non ground) finite failures for standard HCL programs.

REFERENCES

K.R. Apt and M.H. van Emden, Contributions to the theory of logic programming, *J.ACM 29* (1982), 841-862.

K.L. Clark, Negation as Failure, in *Logic and Databases*, H. Gallaire and J.Minker, Eds. (Plenum Press, 1978), 293-322.

K.L. Clark and S. Gregory, A relational language for parallel programming, Proc. ACM Conf. on Functional Programming Languages and Computer Architecture (1981), 171-178.

K.L. Clark and S. Gregory, PARLOG: Parallel programming in logic, *ACM Trans. on Progr. Lang. and Syst. 8* (1986), 1-49.

M. Falaschi, G. Levi, M. Martelli and C. Palamidessi, A new Declarative Semantics for Logic Languages, Proceedings Fifth International Conference on Logic Programming (MIT Press series in Logic Programming, 1988), 993-1005.

M. Falaschi, G. Levi, M. Martelli, C. Palamidessi, Declarative modeling of the operational behaviour of logic languages, Techn. Report TR-10/88, Dipartimento di Informatica, Università di Pisa (1988).

G. Levi and C. Palamidessi, The declarative semantics of logical read-only variables, Proc. 1985 Symp. on Logic Programming (IEEE Comp. Society Press, 1985), 128-137.

G. Levi and C. Palamidessi, An approach to the declarative semantics of synchronization in logic languages, Proc. of the Fourth Int. Conference on Logic Programming (MIT Press Series in Logic Programming, 1987), 877-893.

G. Levi, A new declarative semantics of Flat Guarded Horn Clauses, Technical report, ICOT (January, 1988).

J.W. Lloyd, *Foundations of logic programming* (Springer-Verlag, 1984).

M. J. Maher, Logic semantics for a class of committed-choice programs, Proc. of the Fourth Int. Conference on Logic Programming (MIT Press Series in Logic Programming, 1987), 858-876.

V.A. Saraswat, Problems with Concurrent Prolog, Technical Report 86-100, CMU (January 1986). Revision of unpublished note, May 1985.

V.A. Saraswat, The concurrent logic programming CP: definition and operational semantics, Proceedings of the SIGACT-SIPLAN Symposium on Principles of Programming Languages (ACM, January 1987).

V.A. Saraswat, GHC: Operational semantics, problems and relationship with CP(↓, |), Proceedings of 87 Int. Symposium on Logic Programming (IEEE, August 1987), 347-358.

E.Y. Shapiro, A subset of Concurrent Prolog and its interpreter, Techn. Rep. TR-003, ICOT (1983).

E.Y. Shapiro, Concurrent Prolog: A progress report, *Fundamentals of Artificial Intelligence*, W. Bibel and Ph. Jorrand, Eds., LNCS 232 (Springer-Verlag, 1986), 277-313.

I.E. Stoy, *Denotational Semantics: The Scott-Strachey approach to Programming Languages Theory* (The MIT Press, 1977).

A. Takeuchi, A Semantic Model of Guarded Horn Clauses, unpublished note (1987).

K. Ueda, Guarded Horn clauses, Techn. Rep. TR-103, ICOT (1985). Also in *Logic Programming*, E.Wada, Ed., LNCS 221 (Springer-Verlag, 1986), 168-179.

K. Ueda, Guarded Horn clauses: A parallel logic programming language with the concept of a guard, ICOT Techn. Rep. TR-208 (1986). Also in *Programming of Future Generation Computers*, M.Nivat and K.Fuchi, Eds. (North-Holland, 1988).