TR-345

A New Declarative Semantics of Flat Guarded Horn Clauses

by

Giorgio Levi

February, 1988

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

A new declarative semantics of Flat Guarded Horn Clauses

Giorgio Levi[*]

Dipartimento di Informatica
Università di Pisa
Corso Italia 40, 56100 Pisa, Italy

January 29, 1988

Abstract

This paper defines a new semantics for Flat GHC programs, where a model is a set of FGHC *unit clauses*. The semantics is first informally defined in operational terms. Then the model-theoretic and the fixpoint semantics are given. Their definition relies on the definition of *most general guarded unifiers*. The two semantics are proven to be equivalent and the operational semantics is proved to be sound. Completeness, on the contrary, requires an operational semantics, which avoids failures caused by deadlocks. Finally, a new *unfolding* transformation is defined and proved correct with respect to the model-theoretic semantics.

[*]Work done while the author was visiting ICOT, Tokyo, Japan

1 The language FGHC (Flat Guarded Horn Clauses)

An FGHC program [Ueda 86], [Ueda 87] is a finite set of *clauses* of the form

$$H : -G_1, \ldots, G_m | B_1, \ldots, B_n . (m, n \geq 0)$$

where H (*head*), the G_i's (*guard atoms*) and the B_j's (*body atoms*) are first order logic atomic formulas.

The language has one primitive predicate ($=$), which stands for unification. *Unification atoms* can occur both in the guard and in the body. Guard atoms can only be unification atoms. Empty, i.e. always satisfied, guards and bodies are denoted by the atom *true*.

In the following we assume, for a given FGHC program, D to be the (finite) set of *functors* (with definite arity), denoted by $a, b, c, \ldots$, P to be the (finite) set of *predicates* (with definite arity), denoted by $p, q, r, \ldots$, and V to be a denumerable set of *variables*, denoted by $X, Y, Z, \ldots$

A *goal clause* is a clause of the form

$$? - B_1, \ldots, B_m . (m \geq 1)$$

The FGHC *computation rule* can abstractly be described as a modification of the pure Horn Clause Logic (HCL) rule. Given a program p and a goal

$$G :? - B_1, \ldots, B_n .$$

the execution of G is the (possibly parallel) reduction of G to the empty goal *true*, using the clauses in p.

If λ is the composition of the most general unifiers generated in the derivation, then the substitution $\theta = \lambda|_G$ (the restriction of λ to the variables occurring in G), is the *computed answer substitution*.

The elementary reduction step is nondeterministic AND-parallel resolution, with the following additional rules [Ueda 87]:

Rules of suspension
While trying the application of a clause $H : -G|B$ to the goal A

i) the unification betwen A and G and the evaluation of G are not allowed to instantiate variables in A,

ii) the evaluation of B is not allowed to instantiate variables in G or A, until the clause is *selected for commitment* (see the rule of commitment below).

Unification atoms are reduced using the clause $X = X$ and their reduction is subject to conditions i) and ii).

A unification step which violates conditions i) or ii) is *suspended* and can later be resumed only if the conditions are satisfied.

Rule of commitment

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

A formal declarative semantics of FGHC programs could be obtained as a special case of the construction given in [Levi 87], for a more general class of logic concurrent languages. We will describe here a different approach, directly related to the specific FGHC features. The resulting notion of model is probably easier to understand, even if almost equivalent to the previous one. However, most of the definitions and proofs turn out to be much simpler.

2 Constrained unification and commitment

The nature of the models which are needed to characterize FGHC programs can better be understood by comparing a FGHC program p to the corresponding pure Horn Clause Logic (HCL) program p', obtained by interpreting the *commit operator* (|) as conjunction and by ignoring the difference between guard and body atoms.

As already noted, FGHC has two distinguishing features with respect to HCL, i.e. a set of constraints on the admissibility of substitutions (*constrained unification* as defined by the suspension rules) and a nondeterministic choice mechanism associated to the commit operator (as defined by the rule of commitment). Both mechanisms are associated to the guard concept, yet they can be considered separately.

Let us first consider constrained unification only. This feature strongly affects the semantics, i.e. the evaluation of a goal G in the FGHC program p can fail, even if the same goal evaluated in the HCL program p' succeeds. Moreover, when both p and p' succeed, the set of answer substitutions computed by p can be "smaller" than the set computed by p' (even without commitment).

Example 1
$p(X,Y) : -X = a|Y = b.$
$p(X,Y) : -Y = c|X = a.$

The goal $? - p(X,Y)$. fails in FGHC and succeeds in HCL with the computed answer substitutions $(X = a, Y = b)$ and $(X = a, Y = c)$. The goal $? - p(a, Z)$. succeeds in FGHC with one computed answer substitution only, $(Z = b)$, even without considering the commitment. The same goal in HCL succeeds with two computed answer substitutions, $(Z = b)$ and $(Z = c)$.

There exists one important kind of failure, caused by constrained unification, i.e. *deadlock*. A deadlock occurs when a conjunctive goal fails because it contains a set of suspended goals, each waiting for a partial data structure to be computed by another goal in the set. For example, the goal $? - p(X,Y), q(Y,X)$. is a deadlock, when executed in the program

$p(X,Y) : -X = a|Y = b.$
$q(X,Y) : -Y = b|X = a.$

Let us now consider the commitment rule. Pure commitment (without constrained unification) does not affect the success set semantics [Maher 87], [Takeuchi 87], [Falaschi 88b], provided that we assign it a different meaning. In fact, any answer substitution, computed in the pure HCL program p', is *potentially computed* in the FGHC program p. In FGHC, 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 87], [Falaschi 88b].

Example 2
$p(X,Y) : -true|q(X,Y).$
$p(X,Y) : -X = a|Y = b.$
$p(X,Y) : -X = a|Y = c.$

The goal $? - p(a, Z)$. succeeds in HCL, computing the answer substitutions $(Z = b)$ and $(Z = c)$. The same goal in FGHC (without constrained unification) can either fail or succeed. In the last case, it can compute either $(Z = b)$ or $(Z = c)$. The set of potential computed answer substitutions is therefore the same as the success set in the HCL case.

In conclusion, since we are only interested here in the definition of the success set semantics, we can ignore the commitment and take into consideration constrained unification only.

3 The strong normal form

We will define the language declarative semantics for *FGHC programs in strong normal form.*

Definition 1 *A clause in strong normal form is a clause of the form*

$$A : -I|O \leftarrow B.$$

where A (the head), I (the input guard), O (the output guard) and B (the body) satisfy the following conditions:

i) A is the application of a predicate symbol of arity k to k distinct variables,

ii) I and O are conjunctions of unification atoms of the form variable=term,

iii) there are no unification atoms having the same variable as left term, *iv) the left terms of all the unification atoms in I are variables occurring in the head,*

v) the left terms of all the unification atoms in O are variables occurring in I or in A,

vi) variables which are left terms of unification atoms in I cannot occur in O and in B,

vii) variables which are left terms of unification atoms in O cannot occur in B.

Definition 2 *A FGHC unit clause is a clause in strong normal form whose body is empty. Unit clauses are represented by formulas having the form*

$$A :- I|O.$$

Any "reasonable" FGHC program can be transformed into an equivalent program in strong normal form. Condition i) defines *FGHC programs in normal form* [Furukawa 87]. Any FGHC clause

$$p(t_1, \ldots, t_k) :- G_1, \ldots, G_m | B_1, \ldots, B_n.$$

is transformed into the clause in normal form

$$p(X_1, \ldots, X_k) :- X_1 = t_1, \ldots, X_k = t_k, G_1, \ldots, G_m | B_1, \ldots, B_n.,$$

where $X_1, \ldots, X_k$ are distinct variables not occurring in the original clause.

Programs satisfying conditions ii) to vii) can be obtained by the *normalization procedure*, which performs the following transformations:

a) If there exist atoms of the form $t_1 = t_2$, where t_1 is not a variable, perform the unification of t_1 and t_2, which, if succeeding, results in a new set of unification atoms, which replace the original one.

b) If the guard (the body) contains two atoms $X = t_1$ and $X = t_2$, one of the atoms is replaced, if it is possible, by the result of the unification of t_1 and t_2.

c) If there are two atoms $X = t_1$ and $X = t_2$, such that the first occurs in the guard and the second occurs in the body, $X = t_2$ is replaced, if it is possible, by the result of the unification of t_1 and t_2.

d) If the guard contains an atom $X = t_1$, such that X does not occur in the head, and there exists at least another atom $Y = t_2$ in the guard, such that X occurs in t_2, then delete the atom $X = t_1$ and replace all the occurrences of X by t_1.

e) If the body contains an atom $X = t$, such that X does not occur neither in the head nor in the guard, then delete the atom $X = t$ and replace all the occurrences of X by t.

f) If a non unification atom b in the body contains an occurrence of a variable X and there exists an atom $X = t$ either in the guard or in the body, the occurrences of X in b are replaced by t.

The transformation steps are executed in the specified order. When no transformations are applicable, we obtain a clause in strong normal form

$$A :- I | O \leftarrow B.,$$

where I is the guard, O is the conjunction of the unification atoms in the body and B is the conjunction of the remaining atoms in the body.

Clauses which cannot be transformed in strong normal form (because of a unification failure in steps a), b) or c) or because step d) cannot be performed) are clauses in which either the guard or the body would always fail. These clauses can be disregarded when defining the success set semantics.

If we do not take the commitment rule into account, pure HCL programs can be viewed as a special class of FGHC programs, namely those having empty input guards. This corresponds to the fact that HCL has no constraints on the unification, and, therefore, no causality relation between input and output unification.

4 Towards a notion of model for FGHC programs

The first relevant step in the definition of the semantics is the choice of a suitable *interpretation structure*. In the case of HCL, the semantics is usually based on Herbrand interpretations [vanEmden 76], [Lloyd 84]. We will now show, by means of some examples, why this choice is not satisfactory in the case of FGHC programs.

Let us first consider the program in Example 1 again, which is in strong normal form. Its standard Minimal Herbrand Model semantics is

$$M = \{p(a,b), p(a,c)\}.$$

M seems to model correctly the behaviour of ground FGHC goals. In fact, both the goals $? - p(a,b).$ and $? - p(a,c).$ are refutable in FGHC. However, M does not allow us to predict the behaviour of non-ground FGHC goals. For example, the goal $? - p(X,Y).$ has two instances in M, yet it is not refutable. This example shows that, because of constrained unification, non-ground completeness gets lost, i.e. it is not true any more that if a non-ground atom F has an instance F_σ in the minimal model, then there exists a refutation for $? - F.$, which computes an answer substitution ϑ, such that $\sigma = \lambda * \vartheta$.

It is worth noting that, in general, FGHC is not even ground complete, if body atoms contain local variables, i.e. variables which do not occur in the clause head or in the input guard.

Example 3
$p(X) : -X = a|q(Y).$
$q(Y) : -Y = b|true.$

The minimal Herbrand model is now $\{p(a), q(b)\}$. However, the ground goal $? - p(a).$ is not refutable in FGHC.

We can now note that, in example 1, the incompleteness comes from the fact that suitable values for either the first argument or the second argument of p must be provided as inputs, in order to satisfy the suspension rules. This extra-information must be represented in the model. In [Levi 85], [Levi 87] this information was represented by *functor annotations*. We will use here the notion of *guarded atoms*, which are represented by an FGHC syntax. Our first definition of guarded atom is the following. A guarded atom is a formula

$$A : -I.$$

where A is a possibly non-ground atom and I (the *input guard*) is a conjunction of unification atoms, such that each variable occurring in A is bound to a ground term by one unification atom in I.

A model of the program in Example 1 can now be defined by the following set of guarded atoms:

$$\{p(X, b) : -X = a, p(a, Y) : -Y = c\}$$

This model allows us to predict that the goal $? - p(X, Y)$. will fail (since none of the input guards is satisfied) and that the goal $? - p(a, Y)$. has one potential computed answer substitution (since the input guard of one atom in the model is satisfied).

The present notion of guarded atom does not allow to model FGHC programs which receive a partially determined data structure and compute some data structure component, as shown by the following program.

Example 4
$q(Y) : -Y = f(X)|X = a.$

A model like $\{q(Y) : -Y = f(a)\}$ would not in fact be correct, since it would have a too strong input guard.

We need therefore a more general notion of guarded atom, allowing *output guards* as well. For the sake of simplicity, output guards will be used to represent any data structure computation. A guarded atom is now a formula

$$A : -I|O$$

where A is the application of a predicate symbol of arity k to k distinct variables, I and O are conjunctions of unifications atoms, satisfying the following condition: each variable occurring in A is either bound to a ground term by a unification atom in I or in O, or is bound in I to a non-ground term t, such that all the variables in t are bound to ground terms by unification atoms in O.

According to the new definition, the model of the program in Example 4 is now $\{q(Y) : -Y = f(X)|X = a\}$.

The current definition of guarded atom interpretations (*guarded interpretations*) is the exact counterpart of the standard notion of Herbrand interpretations in the HCL case. Models defined as Herbrand interpretations do not allow a precise characterization of the set of answer substitutions computed by a non-ground goal computation. The same problem arises in FGHC, as shown by the following example.

Example 5
$p(X, Y) : -X = a|true.$
$p(X, Y) : -X = b|Y = b.$

The guarded atom model (*guarded model*) $M1$ is

$\{p(X, Y) : -X = a|Y = a,$
$p(X, Y) : -X = a|Y = b,$
$p(X, Y) : -X = b|Y = b\}.$

This model could suggest that the goal $? - p(a, X)$. has two potential computed answer substitutions, i.e. $(X = a)$ and $(X = b)$, while this is not the case, since the refutation does not compute any answer substitution.

As already noted, the same problem arises in HCL. This problem was solved in HCL, by defining interpretations as sets of non-ground atoms [Falaschi 88a]. The meaning of an atom like $p(X, a)$ in the model is the following. Declaratively, $\forall X.p(X, a)$ is valid. Operationally, the goal $? - p(X, Y)$ has a refutation computing the answer substitution $(Y = a)$.

The same solution can be adopted in our FGHC interpretations, leading to the final definition of *guarded atoms as FGHC unit clauses*, as defined in the previous Section.

The new guarded model of the program in Example 5 is now

$$M2 = \{p(X, Y) : -X = a, p(X, Y) : -X = b|Y = b\}$$

From this model we can infer that the goal $? - p(a, X)$. will succeed without computing any answer substitution and that the goal $? - p(b. X)$. will succeed computing the answer substitution $(X = b)$.

It is worth noting that $M1$ is not actually a model of the program, because the model-theoretic semantics will be defined so as to provide interpretations which contain the "least instantiated information" derivable from program executions.

Let us consider some more examples.

Example 6
$p(X,Y):-Y=X|X=a.$

This is an example of a non-left-linear program. These programs were not modeled correctly by the semantics given in [Levi 87]. The guarded model is now

$$\{p(X,Y):-Y=X|X=a\},$$

which shows that the goals $?-p(X,Y).$, $?-p(a,Y).$ and $?-p(X,a).$ will fail, while the goal $?-p(a,a).$ will succeed.

It is worth noting that, in all the previous examples, the model is exactly the same as the original FGHC program. This corresponds to the fact that the programs considered so far are all unit clauses. We will now consider non-unit clauses.

Example 7
$p(X):-X=f(Y)|Y=a.$
$q(X):-true|X=f(Z).$
$r(X,Y):-X=a|Y=b.$
$s(X,Y,Z):-true|true \leftarrow p(X),q(X),r(Y,Z).$

The first three clauses are unit clauses, hence their denotation in the model is the same as the program. The denotation of s is the guarded atom $s(X,Y,Z):-Y=a|X=f(a),Z=b.$

Example 8
$plus(X,Y,Z):-X=0|Z=Y.$
$plus(X,Y,Z):-X=X1+1|Z=Z1+1 \leftarrow plus(X1,Y,Z1).$

This is an example of recursive program and the model contains infinite guarded atoms.

$\{plus(X,Y,Z):-X=0|Z=Y,$
$plus(X,Y,Z):-X=1|Z=Y+1,$
$plus(X,Y,Z):-X=2|Z=Y+2,\ldots\}$

Example 9.1
$p(X,Y):-X=[X1|Y1]|true \leftarrow q(X1,Y1,Y).$
$q(X,Y,Z):-true|Z=[X|X1] \leftarrow r(Y,X1).$
$r(X,Y):-X=[X1|Y1]|Y=[X1].$
$s(X,Y):-X=[X1|Y1]|Y=[b|Y2].$
$t(X,Y):-true|true \leftarrow p([a|X],Y),s(Y,X).$

This program is taken from [Furukawa 87], where it is used to discuss the correctness of unfolding transformations.

The denotation of q, p and t, which are defined by non-unit clauses are

$q(X,Y,Z):-Y=[X1|Y1]|Z=[X|X1],$
$p(X,Y):-X=[X1,X2|Y1]|Y=[X1|X2],$
$t(X,Y):-true|Y=[a|b],X=[b|Y1],$

respectively.

Example 9.2
It is the same as the program in Example 9.1, apart from the clause for q, which is replaced by the following clause:

$q(X,Y,Z):-Y=[X1|Y1]|Z=[X|X1].$

The denotation of p and q is the same we had in Example 9.1. The new denotation of t is now

$\{t(X,Y):-X=[X1|Y1]|Y=[a|b],X1=b,$
$t(X,Y):-Y=[X1|X2]|X1=a,X2=b,X=[b|Y1]\}$

The semantics of t is different, since the goal $?-t(X,Y).$, with no input values, fails because of a deadlock. It is worth noting that the denotation of predicates p and s is the same in the two programs. However, t, which is defined in terms of p and s, has a different semantics. This means that the operational semantics is not *compositional*, i.e. the semantics of a predicate cannot always be obtained as composition of the semantics of the predicates which occur in the bodies of the clauses which define it. These examples, related to the deadlock, will be reconsidered later, when discussing the completeness of the operational semantics and the correctness of unfolding.

It is worth noting that the models we have defined are (possibly infinite) sets of unit clauses, as is the case of models introduced in [Falaschi 88a] for pure HCL programs. Due to the presence of variables in the interpretations, nonrecursive programs have always a finite model, even if the interpretation domain is infinite (i.e. if the program contains at least one functor whose arity is ≥ 1). Models of HCL programs can also be viewed as (possibly infinite) sets of FGHC unit clauses with empty input guards.

5 The model theoretic semantics

As already mentioned, the interpretation domain is a set of possibly non-ground atoms. If the language is defined by the triple $< D, V, P >$ (functors, variables and predicates), $T_{D(V)}$ (the set of *terms*) is the free D-algebra on V.

$\leq$ is the preorder on terms, defined by $t_1 \leq t_2$ iff $\exists \vartheta . t_{1_\vartheta} = t_2$, where ϑ is a substitution (i.e. a mapping from variables to terms). The symmetric closure of $\leq$ is an equivalence relation on terms, called *variance* ($\sim$).

Definition 3 *Our interpretation domain U is defined as $T_{D(V)/\sim}$, i.e. the quotient set of $T_{D(V)}$ with respect to the variance relation.*

The standard Herbrand Universe is the subset of U containing ground terms only. The preorder $\leq$ on $T_{D(V)}$ induces a partial ordering relation on $T_{D(V)/\sim}$ (and therefore on U). For the sake of simplicity, the elements of U will have the same representation of the elements of $T_{D(V)}$ (the meaning of $f(X, g(Y)) \in U$ is that the equivalence class of $f(X, g(Y))$ belongs to U). The partial order on U will also be denoted by $\leq$.

Definition 4 *The interpretation base H is the set of all the guarded atoms in strong normal form*

$$p(X_1, \ldots, X_n) : -I|O.,$$

such that $p \in P$, p has arity n, and I and O are conjunctions of unification atoms of the form $v_i = t_i$, satisfying the conditions given in the previous Section, where $v_i \in V$, $t_i \in U$.

Definition 5 *A guarded interpretation is any subset of H.*

The set of all the guarded interpretations $\{J\}$ is partially ordered by set inclusion ($\subseteq$). ($\{J\}, \subseteq$) is a complete lattice, i.e. every set of interpretations has a greatest lower bound and a least upper bound. Φ (the empty interpretation) and H are the bottom and top element of the lattice.

Definition 6 *A guarded model of an FGHC program w is a guarded interpretation J, such that all the clauses in w are true in J.*

The definition of truth in a guarded interpretation is based on the notion of most general guarded unifier (mggu). We will first define the extension of the standard *most general unifier* (mgu).

Definition 7 *(Most General Unifier) A unit goal*

$$g : p(t_1, \ldots, t_n)$$

and a guarded atom

$$a : p(X_1, \ldots, X_n) : -I|O$$

are unifiable, iff the following conditions are satisfied. Let λ be the substitution $(X_1 = t_1, \ldots, X_n = t_n)$.

i) $I_\lambda = \mu$ is a set of unification atoms which can be evaluated to true without instantiating any variable in g,

ii) the normalization of $O_{\lambda,\mu}$ is a substitution ϑ.

If g and a are unifiable, their most general unifier (mgu) is the substitution ϑ.

For example,

- the goal $q(f(W))$ and the guarded atom

$$q(Y) :- Y = f(X) | X = a$$

are unifiable and their mgu is $(W = a)$.

- the goal $p(a, W)$ and the (non left-linear) guarded atom

$$p(X, Y) :- Y = X | X = a$$

are not unifiable (since the evaluation of $I_\lambda = (W, a)$ would bind a variable in g).

The above definition of unification (and mgu) reflects FGHC constrained unification. It is worth noting that, if the guarded atom has an empty input guard (i.e. it is a HCL atom), the new definition reduces to the standard one.

Definition 8 *(Most General Guarded Unifier) A unit goal*

$$g : p(t_1, \ldots, t_n)$$

and a guarded atom

$$a : p(X_1, \ldots, X_n) :- I|O$$

are guard-unifiable, iff the following conditions are satisfied. Let λ be the substitution $(X_1 = t_1, \ldots, X_n = t_n)$.

i) the normalization of I_λ can be decomposed in a set of unification atoms μ, which can be evaluated to true, without instantiating any variable in g and a set of unification atoms (substitution) μ_1,

ii) the normalization of $O_{\lambda.\mu.\mu_1}$ is a substitution ϑ.

If g and a are guard-unifiable, their most general guarded unifier *(mggu) is the pair of substitutions (μ_1, ϑ).*

It is worth noting that if g and a are unifiable (with mgu ϑ), then they are also guard-unifiable (with mggu (Φ, ϑ)).

For example,

- the goal $p(a, W)$ and the guarded atom

$$p(X, Y) : -Y = X | X = a$$

are guard-unifiable and their mggu is the pair

$$((W = a), \Phi).$$

- the goal $p(b, W)$ and the guarded atom

$$p(X, Y) : -Y = X | X = a$$

are not guard-unifiable (since $b = a$, in the output guard, fails).

The above definitions can be extended to the guard-unification of tuples.

Definition 9 *(Most General Synchronous Unifier on tuples) An n-tuple of goals*

$$(g_1, \ldots, g_n)$$

and an n-tuple of guarded atoms

$$(a_1, \ldots, a_n)$$

are s-unifiable iff the following conditions are satisfied.

Let

$$\lambda_1 = (\mu_1, \vartheta_1), \ldots, \lambda_n = (\mu_n, \vartheta_n)$$

be the mggu's computed for each (g_i, a_i).

If for each equational atom $X = t_i$, which occurs in a μ_i, there exists an equational atom $X = t_j$, occurring in a ϑ_j $(i \neq j)$, then the unification atom $X = t_i$ is removed from μ_i and the atom $t_i = t_j$ is composed with ϑ_j.

The most general synchronous unifier (mgsu) is the substitution obtained by composing the resulting ϑ_i's.

For example, the pair of goals $(p(W), q(W))$ and the pair of guarded atoms

$$(p(X1) :- X1 = f(X2) | X2 = a, q(Y1) :- true | Y1 = f(Y2))$$

are guard-unifiable. The separate mggu's are

$((W = f(X2)), (X2 = a))$ and
$(\Phi, W = f(Y2))$.

Hence the composed mgsu is $(W = f(a))$.

Definition 10 *(Most General Guarded Unifier on tuples) An n-tuple of goals*

$$(g_1, \ldots, g_n)$$

and an n-tuple of guarded atoms

$$(a_1, \ldots, a_n)$$

are guard-unifiable iff the following algorithm teminates successfully.

1. Compute the composition λ of the mgu's of all the pairs which are unifiable and apply λ to the remaining goals.

2. Compute the composition μ of the mgsu's of all the pairs of tuples which are s-unifiable and apply it to the remaining goals.

3. For each permutation p of the remaining goals and guarded atoms

$$(g_1, \ldots, g_k),$$

$$(a_1, \ldots, a_k)$$

let $(\vartheta_i^p, \sigma_i^p)$ be the mggu of g_i and a_i, obtained by applying the mggu's of the previous pairs in the permutation and let (ϑ^p, σ^p) be the normalization of the compositions of the single mggu's.

The most general guarded unifier (mggu) is the set of pairs of substitutions, obtained by normalizing $(\vartheta^p, \lambda.\mu.\sigma^p)$ for each permutation p.

For example, the pair of goals $(q(X), r(X))$ and the pair of guarded atoms

$$(r(Y1) : -Y1 = f(g(Y2))|Y2 = a,$$
$$q(Z1) : -Z1 = f(Z2)|Z2 = g(a))$$

are guard-unifiable and have two mggu's:

$((X = f(Z)), (Z = g(a)))$ and $((X = f(g(Y))), (Y = a))$.

We can now give the definition of *truth of a guarded clause in a guarded interpretation.*

Definition 11 *(Truth of a guarded clause in a guarded interpretation)*
Let c be a guarded clause and J be a guarded interpretation.
i) if c is a unit clause, then c is true in J iff $c \in J$.
ii) if c has the form

$$A : -I|O \leftarrow B_1, \dots, B_n,$$

c is true in J iff
for every n-tuple $(A'_1, \dots, A'_n)$ of guarded atoms in J, such that $(B_1, \dots, B_n)$ and $(A'_1, \dots, A'_n))$ are guard unifiable, then for each

$$(\vartheta_1, \vartheta_2) = mggu((B_1, \dots, B_n), (A'_1, \dots, A'_n))$$

the normalization of

$$A : -I, \vartheta_1|O, \vartheta_2$$

belongs to J.

We will now give a set of theorems on guarded models, whose proofs can easily be derived from the proofs of the corresponding theorems in [Falaschi 88a].

Theorem 1 *(Model intersection property) If L is a non-empty set of guarded models of an FGHC program w, then $\bigcap L$ is a guarded model of w.*

Theorem 2 *The class of guarded models is a complete lattice.*

Theorem 3 *(Existence of the minimal model) For every FGHC program w, there exists a minimal model.*

Definition 12 *(Model-theoretic semantics) The model-theoretic semantics $M_m(w)$ of an FGHC program w is its minimal model.*

It is easy to show that the models of the previous Section, defined by means of operational arguments, for the programs in Examples 5 to 9.1, are indeed minimal guarded models. Unfortunately, the semantics given to the program in Example 9.2 (where some goal execution results in a dead-lock) is not the minimal model, which is instead identical to the minimal model of the program in Example 9.1. The model-theoretic semantics is, in fact, compositional and is forced to provide the same semantics to the two programs. It is worth noting that the "operational model" of the program in Example 9.2 is not even a model, since it does not contain the minimal model.

6 Fixpoint semantics

In this Section we define a continuous transformation on guarded interpretations, whose least fixpoint is shown to be equivalent to the model-theoretic semantics.

Definition 13 *(Transformation on guarded interpretations) Let w be an FGHC program. The mapping T on the set of guarded interpretations of w is defined as follows.*

$T(J) = \{A : -I', O' \in H,$ *such that*
$\exists A : -I|O \leftarrow B_1, \ldots, B_n$ *in* w,
$\exists A'_1, \ldots, A'_n \in J$,
$\exists(\vartheta_1, \vartheta_2) = mggu((B_1, \ldots, B_n), (A'_1, \ldots, A'_n))$,
I' *is the normalization of* $I.\vartheta_1$,
O' *is the normalization of* $O.\vartheta_2\}$.

The following theorems state relevant properties of T.

Theorem 4 *T is monotonic and continuous.*

Theorem 5 *There exists the least fixpoint of T,*

$$lfp(T) = \bigcup_{n \in \omega} T^n(\Phi) = T \uparrow \omega$$

Definition 14 *(Fixpoint semantics) The fixpoint semantics $M_f(w)$ of an FGHC program w is the least fixpoint of the transformation T associated to w.*

The following theorems state the equivalence between the model-theoretic and the fixpoint semantics.

Theorem 6 *A guarded interpretation J is a guarded model iff $T(J) \subseteq J$.*

Theorem 7 *(Equivalence of model-theoretic and fixpoint semantics) For every FGHC program w, $M_m(w) = M_f(w)$.*

The above theorems allow us to incrementally compute the minimal model by a bottom-up program execution. Let us consider a few examples.

Example 7
$p(X) : -X = f(Y)|Y = a.$
$q(X) : -true|X = f(Z).$
$r(X,Y) : -X = a|Y = b.$
$s(X,Y,Z) : -true|true \leftarrow p(X), q(X), r(Y,Z).$

$T^1(\Phi) =$
$\{p(X) : -X = f(Y)|Y = a,$
$q(X) : -true|X = f(Z),$
$r(X,Y) : -X = a|Y = b\}.$

$T^2(\Phi) =$
$\{p(X) : -X = f(Y)|Y = a,$
$q(X) : -true|X = f(Z),$
$r(X,Y) : -X = a|Y = b,$

$s(X,Y,Z) :- Y = a | X = f(a), Z = b\}.$

$T^3(\Phi) = T^2(\Phi) = M_f.$

Example 9.1
$p(X,Y) :- X = [X1|Y1] | true \leftarrow q(X1, Y1, Y).$
$q(X,Y,Z) :- true | Z = [X|X1] \leftarrow r(Y, X1).$
$r(X,Y) :- X = [X1|Y1] | Y = [X1].$
$s(X,Y) :- X = [X1|Y1] | Y = [b|Y2].$
$t(X,Y) :- true | true \leftarrow p([a|X], Y), s(Y, X).$

$T^1(\Phi) =$
$\{r(X,Y) :- X = [X1|Y1] | Y = [X1],$
$s(X,Y) :- X = [X1|Y1] | Y = [b|Y2]\}.$

$T^2(\Phi) =$
$\{r(X,Y) :- X = [X1|Y1] | Y = [X1],$
$s(X,Y) :- X = [X1|Y1] | Y = [b|Y2],$
$q(X,Y,Z) :- Y = [X1|Y1] | Z = [X|X1]\}.$

$T^3(\Phi) =$
$\{r(X,Y) :- X = [X1|Y1] | Y = [X1],$
$s(X,Y) :- X = [X1|Y1] | Y = [b|Y2],$
$q(X,Y,Z) :- Y = [X1|Y1] | Z = [X|X1],$
$p(X,Y) :- X = [X1, X2|Y1] | Y = [X1|X2]\}.$

$T^4(\Phi) =$
$\{r(X,Y) :- X = [X1|Y1] | Y = [X1],$
$s(X,Y) :- X = [X1|Y1] | Y = [b|Y2],$
$q(X,Y,Z) :- Y = [X1|Y1] | Z = [X|X1],$
$p(X,Y) :- X = [X1, X2|Y1] | Y = [X1|X2],$
$t(X,Y) :- true | Y = [a|b], X = [b|Y1]\}.$

$T^5(\Phi) = T^4(\Phi) = M_f.$

As already noted, the program in Example 9.2 has exactly the same model.

Example 10
$p(X, Y) :- true|Y = h(X) \leftarrow q(X), r(X).$
$q(X) :- X = f(Y)|true \leftarrow t(Y).$
$r(X) :- X = f(g(Y))|true \leftarrow s(Y).$
$t(Y) :- true|Y = g(a).$
$s(Y) :- true|Y = a.$

$T^1(\Phi) = \{$
$t(Y) :- true|Y = g(a),$
$s(Y) :- true|Y = a\}.$

$T^2(\Phi) = \{$
$t(Y) :- true|Y = g(a),$
$s(Y) :- true|Y = a,$
$r(X) :- X = f(g(Y))|Y = a,$
$q(X) :- X = f(Y)|Y = g(a)\}.$

$T^3(\Phi) = \{$
$t(Y) :- true|Y = g(a),$
$s(Y) :- true|Y = a,$
$r(X) :- X = f(g(Y))|Y = a,$
$q(X) :- X = f(Y)|Y = g(a),$
$p(X, Y) :- X = f(Z)|Z = g(a), Y = h(f(g(a))),$
$p(X, Y) :- X = f(g(Z))|Z = a, Y = h(f(g(a)))\}.$

$T^4(\Phi) = T^3(\Phi) = M_f.$

It is worth noting that, because of step 3 in the definition of guarded unification, the predicate p has two atoms in the minimal model even if each predicate is defined by a single clause.

7 Soundness and completeness properties

The following theorem shows that the operational semantics of FGHC is sound with respect to the Model-theoretic semantics.

Theorem 8 *(Soundness) Let w be an FGHC program, let G be a unit goal and assume that $? - G$. has a refutation with computed answer substitution ϑ. Then there exist a guarded atom*

$$A : -I|O$$

in $M_m(w)$ and a substitution

$$\lambda = mgu(G, A : -I|O),$$

such that $\lambda|_G = \vartheta|_G$.

As already noted, the operational semantics is not complete, as shown by the deadlock example. The incompleteness is, however, related to deadlocks only. In fact, if the operational semantics is changed so as to avoid suspensions that would cause deadlocks, the following theorem holds.

Theorem 9 *(Completeness) Let w be an FGHC program, let G be a unit goal and assume there exist a guarded atom*

$$A : -I|O$$

in $M_m(w)$ and a substitution

$$\lambda = mgu(G, A : -I|O),$$

then $? - G$. has a refutation with computed answer substitution ϑ, such that $\lambda|_G = \vartheta|_G$.

The complete operational semantics makes the implementation more complex, since it requires a run-time deadlock detection. However, there are several arguments to support it.

The first argument is related to the declarative semantics. It is possible to give a declarative semantics modeling the failure of deadlocks, following the construction sketched in [Levi 85]. However, the resulting semantics is very complex and does not have any longer the structure of the standard semantics of logic programs.

A second argument comes from the language features point of view, since deadlock solution can be viewed as a synchronous communication among processes.

The strongest argument, however, is related to program transformation, and, in particular, to the unfolding rules. This issue will be discussed in the next Section.

8 Unfolding

Unfolding is an elementary program transformation rule, which plays a relevant role in any program transformation system. It corresponds to the *copy rule*, which is sometimes used to define the operational semantics of procedures in any programming language. According to the copy rule, each occurrence of a procedure call can be replaced by an instance of the procedure body, with the suitable parameter bindings.

Any language should then be equipped with a safe unfolding rule, i.e. a rule which preserves the semantics.

A set of unfolding rules for FGHC was given in [Furukawa 87]. These rules are quite similar to the rules we give in the following. The relation between the two sets of rules will be discussed later in this Section. Our rules are based on the concepts of mgu, msgu and mggu.

Definition 15 *(Unfolding rules) Given an FGHC non unit clause*

$$c : A : -I|O \leftarrow G_1, \ldots, G_n.$$

and an FGHC program w, the unfolding of c in w is a set of clauses generated according to the following rules.

Rule 1 (unifiable goals). If w contains the clauses

$$A_i : -I_i|O_i \leftarrow B_i.(i \leq n)$$

such that G_j is unifiable with $A_i : -I_i|O_i$ with mgu ϑ_i, then generate all the clauses which are the normalizations of

$$A : -I|O, \vartheta_i \leftarrow G_1, \ldots, G_{j-1}, B_i, G_{j+1}, \ldots, G_n.$$

Rule 2 (s-unifiable tuples of goals). For the sake of simplicity, we assume that the goals in the body of c are possibly reordered so as to make the rule applicable to the first m goals. For each tuple of clauses in w

$$\begin{aligned}&(A_1 : -I_1|O_1 \leftarrow B_1., \ldots,\\ &\ldots, A_m : -I_m|O_m \leftarrow B_m.), (2 \leq m \leq n)\end{aligned}$$

such that the tuple of goals $(G_1, \ldots, G_m)$ is s-unifiable with

$$(A_1 : -I_1|O_1, \ldots, A_m : -I_m|O_m)$$

with mgsu ϑ, then generate all the clauses which are the normalizations of

$$A : -I|O, \vartheta \leftarrow B_1, \ldots, B_m, G_{m+1}, \ldots, G_n.$$

Rule 3 (guard-unifiable tuples of goals) The rule is applicable only when neither Rule 1 nor Rule 2 are applicable. For each tuple of clauses in w,

$$\begin{aligned}&(A_1 : -I_1|O_1 \leftarrow B_1., \ldots,\\ &\ldots, A_n : -I_n|O_n \leftarrow B_n.),\end{aligned}$$

such that the tuple of goals $(G_1, \ldots, G_n)$ is guard-unifiable with

$$(A_1 : -I_1|O_1, \ldots, A_n : -I_n|O_n),$$

let (μ_i, ϑ_i) be the mgsu of G_i and $A_i : -I_i|O_i$. Generate all the clauses which are the normalizations of

$$A : -I, \mu_i|O, \vartheta_i \leftarrow G_1, \ldots, G_{i-1}, B_i, G_{i+1}, \ldots, G_n.$$

Rules 1, 2 and 3 clearly reflect the definition of mggu used in the declarative semantics. The differences with the rules in [Furukawa 87] are the following:

i) Our rule 2, which reflects synchronous unification of tuples, is not used.

ii) Our rule 3 can only be applied if the output guard of the clause is empty.

Before discussing the rationale for our extensions, let us give a few examples.

Example 9.1 (partial)
$q(X,Y,Z) : -true|Z = [X|X1] \leftarrow r(Y,X1).$
$r(X,Y) : -X = [X1|Y1]|Y = [X1].$
$s(X,Y) : -X = [X1|Y1]|Y = [b|Y2].$
$t(X,Y) : -true|true \leftarrow q(a,X,Y), s(Y,X).$

The clause for t can be unfolded first at the call of q (by rule 1), generating the clause

$t(X,Y) : -true|Y = [a : R4] \leftarrow r(X,R4), s([a|R4],X).,$

which can now be unfolded at the call of s (by rule 1), generating the clause

$t(X,Y) : -true|Y = [a : R4], X = [b|Z5] \leftarrow r([b|Z5],R4).,$

which can now be unfolded at the call of r (by rule 1), generating the unit clause

$t(X,Y) : -true|Y = [a|b], X = [b|Z5].$

Note that if we start by unfolding the clause defining q, we obtain (by rule 3) the unit clause

$q(X, Y, Z) : -Y = [X1|Y1]|Z = [X|X1].$

The first unfolding (by rule 1) of the clause defining t cannot any longer be executed. However, we can use rule 2 to unfold concurrently the calls of q and s, obtaining the same unit clause we obtained for t in the previous unfoldings sequence. It is worth noting that the second unfoldings sequence would not be possible with the rules in [Furukawa 87], since the clause for q would not be unfoldable.

Example 10 (partial)
$p(X, Y) : -true|Y = h(X) \leftarrow q(X), r(X).$
$q(X) : -X = f(Y)|true \leftarrow t(Y).$
$r(X) : -X = f(g(Y))|true \leftarrow s(Y).$

The clause for p cannot be unfolded by rules 1 and 2. We can then apply rule 3, which generates the clauses

$p(X, Y) : -X = f(Z)|Y = h(f(Z)) \leftarrow r(f(Z)).$ and
$p(X, Y) : -X = f(g(Z))|Y = h(f(g(Z))) \leftarrow q(f(g(Z))).$

Again, this unfolding would not be possible with the rules in [Furukawa 87], since the clause for p contains a non-empty output guard.

It is worth noting that if the FGHC program contains clauses, whose input guards are always empty (i.e. it is a HCL program), rule 1 is always applicable, and we obtain the standard HCL rule.

The unfolding of the program in Example 9.1 shows that our unfolding rules are compositional, i.e. we can safely use unfolded clauses in the unfolding of other clauses. This property holds for any FGHC program, as shown by the following theorem.

Theorem 10 *(Compositionality of unfolding) Let w be any FGHC program and let $c, c_1, \ldots, c_m (m \geq 1)$ be clauses in w, such that one of the unfolding rules allows to unfold c using the clauses $c_1, \ldots, c_m$, computing the set of clauses $\{c^1, \ldots, c^k\}$.*

Then if any clause c_i is unfolded to $\{c_i^1, \ldots, c_i^{k_i}\}$, then c can be unfolded with the clauses

$$c_1, \ldots, c_{i-1}, c_i^1, \ldots, c_i^{k_i}, c_{i+1}, \ldots, c_m,$$

obtaining exactly the clauses $\{c^1, \ldots, c^k\}$.

Our unfolding rules can always be applied to any non unit clause. In fact, if no rules are applicable, the body of the clause will always fail and the clause can be deleted. The (possibly infinite) application of unfolding generates a (possibly infinite) set of unit clauses. This allows us to define the semantics of any FGHC programs in terms of unfolding, as shown by the following definition.

Definition 16 *(Unfolding semantics) The unfolding semantics $M_u(w)$ of an FGHC program w is the set of guarded atoms (unit clauses), which can be obtained by unfolding the clauses in w.*

It is easy to show that the bottom-up repeated application of the transformation T is exactly an unfolding sequence. Hence the following theorem holds.

Theorem 11 *(Equivalence of fixpoint and unfolding semantics) For any FGHC program w,*

$$M_u(w) = M_f(w).$$

The correctness and completeness of unfolding are a straigthforward consequence of the above theorem. The rules in [Furukawa 87] are therefore also correct, while they are certainly not complete, since they cannot always be applied to obtain unit clauses only.

Let us now consider the problem of deadlocks again. As shown by our Example 9.1, our unfolding rules can transform a deadlock-free program into a program which causes a deadlock. The two programs are equivalent from our semantics point of view, even if they are different from the standard FGHC operational semantics viewpoint.

As we have noted several times, a good semantics (and a good unfolding rule) should always be compositional (and the unfolding rule should be complete). Hence programs such as those in Examples 9.1 and 9.2 cannot be given a different semantics. Unfolding, however, allows us to look for an alternative to the modification of the FGHC semantics. In fact, we could look for models, where the program in Example 9.1 has the semantics of the program in Example 9.2. The program in Example 9.1 is a deadlock reducible program, according to the following definition.

Definition 17 *(Deadlock reducible programs) An FGHC program w is deadlock reducible iff there exists a sequence of unfoldings (using rules 1 and 3 only), which reduces w to a program which causes deadlocks.*

If deadlock reducible programs could statically be detected and ruled out, we could then provide a different declarative semantics (and unfolding rule), which could make complete the standard operational semantics of FGHC. This problem has still to be worked out. It seems, however, that a very simple modification of the definition of mggu (inhibiting synchronous unifications) is all we need. As a consequence, the unfolding rule 2 would not be needed any longer.

Let us finally note that our semantics does not model intermediate values computed by partial computations. Therefore, programs which have the same input-output behaviour and compute different intermediate values have exactly the same semantics. As a matter of fact, this is the real difference between the programs in the Examples 9.1 and 9.2. Our completeness results show that, from the operational semantics viewpoint, intermediate partial values are only relevant to the case of deadlocks and deadlock reducible programs.

9 Open problems

One relevant open problem is, of course, finding a solution to the deadlock problem and fixing the incompleteness associated to deadlocks. There are, however, other problems which must be solved in order to obtain a satisfactory semantic characterization of FGHC.

The first problem, as already mentioned, is the formal characterization of the set of finite failures. This would allow to model the commitment and could be useful in validating program transformation rules, which should preserve both the success and the finite failure set. A solution can be found by generalizing the construction in [Falaschi 88b] to the case of AND-parallel computation rules.

The second problem is related to the semantics of *logical perpetual processes* [Lloyd 84]. In fact, most interesting FGHC programs define nonterminating processes which produce and/or consume infinite data structures (streams). Because of non-termination, the standard denotation of these programs is empty.

The semantics of perpetual processes was studied in the case of pure HCL. There exists an elegant operational semantics, based on the notion of atoms being computable at infinity [Lloyd 84]. There are also some proposals for the declarative semantics [Lloyd 84], [Levi 86], which, however, make the operational semantics either not complete or not sound.

The last interesting problem, not strictly related to FGHC, is trying to apply guarded models to define the semantics of other concurrent logic languages, such as PARLOG [Clark 86] and Concurrent Prolog [Shapiro 86]. If this will result to be possible, as we strongly conjecture, the semantics of all the major concurrent logic languages would be expressed in terms of sets of FGHC unit clauses. This could lead us to argue that the synchronization mechanism of FGHC is, on one side, flexible and easy to implement, and, on the other side, more primitive and simple than those proposed for the other languages.

10 Acknowledgements

I would like to thank Koichi Furukawa and the researchers of the First Laboratory of ICOT. Special thanks are due to Masaki Murakami and Kazunori Ueda for their helpful discussions.

References

[Clark 86] K.L. Clark and S. Gregory. PARLOG: Parallel programming in logic. *A.C.M. Trans. on Programming Languages and Systems 8*(1986), pp. 1-49.

[vanEmden 76] M. vanEmden and R.A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the A.C.M. 23*(1976), pp. 733-742.

[Falaschi 88a] M. Falaschi, G. Levi, M. Martelli and C. Palamidessi. A more general declarative semantics for logic programming languages. Dipartimento di Informatica, Università di Pisa, Italy, Techn. Report, January 1988.

[Falaschi 88b] M. Falaschi and G. Levi. Operational and fixpoint semantics of a class of committed-choice languages. Dipartimento di Informatica, Università di Pisa, Italy, Techn. Report, January 1988.

[Furukawa 87] K. Furukawa, A. Okumura and M. Murakami. Unfolding rules for GHC programs. In D. Bjorner, A.P. Ershov and N.D. Jones, editors, *Workshop on Partial Evaluation and Mixed Computation*, Gl. Avernaes, Denmark, October 1987. To appear in *New Generation Computing*.

[Levi 85] G. Levi and C. Palamidessi. The declarative semantics of logical read-only variables. *Proc. 1985 Symp. on Logic Programming*. IEEE Comp. Society Press, 1985, pp. 128-137.

[Levi 86] G. Levi and C. Palamidessi. Contributions to the theory of logical perpetual processes. Dipartimento di Informatica, Università di Pisa, Italy, Techn. Report, March 1986. To appear in *Acta Informatica*.

[Levi 87] G. Levi and C. Palamidessi. An approach to the declarative semantics of synchronization in logic languages. *Proc. Fourth Int'l Conf. on Logic Programming*. The MIT Press, Series in Logic Programming, 1987, pp. 877-893.

[Lloyd 84] J.W. Lloyd. *Foundations of logic programming*. Springer-Verlag, 1984.

[Maher 87] M.J. Maher. Logic semantics for a class of committed-choice logic languages. *Proc. Fourth Int'l Conf. on Logic Programming.* The MIT Press, Series in Logic Programming, 1987, pp. 858-876.

[Shapiro 86] E.Y. Shapiro. Concurrent Prolog: A progress report. In W. Bibel and P. Jorrand, editors, *Fundamentals of Artificial Intelligence.* Lecture Notes in Computer Science 232, Springer-Verlag, 1986, pp. 277-313.

[Takeuchi 87] A. Takeuchi. A semantic model of Guarded Horn Clauses. Unpublished Note, 1987.

[Ueda 86] K. Ueda. Guarded Horn Clauses. In E. Wada, editor, *Proc. Logic Programming '85.* Lecture Notes in Computer Science 22, Springer-Verlag 1986, pp. 168-179. To appear in E.Y. Shapiro, editor, *Concurrent Prolog: Collected Papers.* The MIT Press, 1987.

[Ueda 87] K. Ueda. Guarded Horn Clauses: A parallel logic programming language with the concept of a guard. ICOT Techn. Rept. TR-208. To appear in M. Nivat and K. Fuchi, editors, *Programming of Future Generation Computers.* North-Holland, 1987.