TM-0728

Nested Guarded Horn Clauses: A Language Provided with a Complete Set of Unfolding Rules

by
M. Falaschi, M. Gabbrielli, G. Levi
& M. Murakami

May, 19"

Nested Guarded Horn Clauses: a language provided with a complete set of Unfolding Rules.

Moreno Falaschi, Maurizio Gabbrielli*, Giorgio Levi*, Masaki Murakami+*

*Università di Pisa, Dipartimento di Informatica
Corso Italia 40,
Pisa, Italy.

+ First research laboratory,
ICOT.
4-28, Mita 1-chome, Minato-ku,
Tokyo, 108, Japan.

Extended Abstract

Note for the program committee. In order to keep the paper in the page limits, we have moved to an appendix the definition of an alghoritm (normalization). This more technical part is not essential and can be eliminated in the final version.

1. Introduction.

In the last few years, a lot of efforts have been devoted to the definition of concurrent logic programming languages. We are mainly interested in the family of languages that includes PARLOG [CG 86, G 87], Concurrent Prolog (CP) [Sh 86] and Guarded Horn Clauses (GHC) [U 86, U 87]. We were initially motivated by the problem of defining a semantics in the case of GHC. However we think that some of our results can be applied to the other cases too, due to many basic similarities between GHC, CP and PARLOG as pointed out by Shapiro [Sh 89]. All these languages do somewhat affect the basic logic programming computational mechanism, i. e. unification, by introducing synchronization primitives which are all based on same sort of constrained unification.

As a consequence, the traditional elegant semantics of Definite Horn Clauses [vEK 76, AvE 82, Ll 87] is not suitable anymore to describe these languages, because the extralogical features that have been added. Most of the semantics developed for these languages are either in the operational style, based on transition systems [S 85], [B 86], [BK 88], [S 87] and [S 87a], or in a denotational style, based on metric spaces [BK 88], [K 88]. Both these approaches are based on the traditional techniques for imperative languages. The first attempts to define a semantics with a declarative flavour can be found in [LP 85], [LP 87] and [L 88] for variants of the original languages, namely the *deadlock*-free versions. [S 85] first proposed, in an operational semantics framework, *input/output histories* as the most adequate semantic domain for these languages. Similar ideas have recently been used in fixpoint semantics definitions [L 88, M 88, P 88, BKPR 89]. For example, in [P 88, BKPR 89] the characterization of (full) GHC and PARLOG, respectively, is based on variable annotations to represent the input-output constraints.

Levi [L 88a] proposes a new approach to the semantics to provide a "semantic scheme" for all the languages that differ by modification of the unification paradigm. This approach has already been successfully applied to the case of pure Horn clause logic [LM 88] and to the already mentioned variant of Flat GHC [L 88]. The key issue of the approach is that models should allow to derive the interesting operational properties and Herbrand models are not adequate, even in the case of pure logic programs [FLMP 88]. The right notion of model is then, informally, that of a (possibly infinite) set of unit clauses. The relevant operational properties must be observable, by executing a goal in the model. With this notion of model, a formal semantics can be based on program transformation techniques, i.e. *unfolding* [TS 84]. Unfolding is, in fact, strongly related to the operational semantics and can be used to formally derive the model, which is also a program. The unfolding semantics and the least fixpoint semantics, are strictly related, since if we have a correct notion of unfolding, it is straightforward to define the transformation T_P.

In order to give a semantics to GHC according to Levi's proposal, one major open problem was to find a complete (i.e. always applicable) set of unfolding rules for the language. There exist several attempts to define a set of unfolding rules for GHC [FOM 87, FOF 88, U 88] but none of them resulted in a complete set. We believe that it is not possible to give a complete set of rules for GHC (and for the other mentioned languages as well) because the GHC syntax is not powerful enough to express its own semantic, which has to be preserved by the unfolding process. Therefore the problem of giving a complete set of unfolding rules for GHC can be solved only by extending the language.

In this paper, starting from Flat GHC (FGHC) and following Courien' s philosophy of making syntax akin to semantics [C 86], we define a new language, *Nested Guarded Horn Clauses (NGHC)*, which is equipped with a complete set of unfolding rules. Consequently, according to the method exploited in [LM 88] we define an unfolding semantics and an equivalent fixpoint semantics for NGHC. Since it is possible to embedd FGHC in NGHC, this allows to give a semantics to FGHC as well.

The main new feature of NGHC is its concept of guard. A generic NGHC clause has n ($n \geq 1$) layers of (standard) guards. This improvement in syntax is directly related to the language semantics since it is this feature that allows to define a complete set of unfolding rules for NGHC (and therefore a fixpoint semantics). Since our models are made of NGHC unit clauses and since every reasonable model has to satisfy some minimality properties, we define a strong normal form for NGHC clauses, a related normalization procedure and a minimalization operation on NGHC programs in strong normal form. An NGHC clause is proved to be equivalent (w.r.t success set) to its normalized version, which represents a sort of canonical form. With minimalization, we can eliminate redundant clauses from a program to obtain an equivalent (w.r.t. success set) program.

We define a complete set of unfolding rules for NGHC and then, once we have formally defined our notion of *guarded interpretation* as a minimal set of NGHC clauses in strong normal form, we define an *Unfolding semantics* ([L 88a], [LM 88]), which is proved equivalent (w.r.t success set) to the operational one. We also define a fixpoint semantics in the classic logic programming style, by making use of an operator T_{UP} defined in terms of unfolding which works on (guarded) interpretations. The Fixpoint and Unfolding semantics are proved to be equivalent (w.r.t. success set).
We assume the reader to be familiar with the concepts of logic programming (see [Ll 87]).

2. The NGHC (Nested Guarded Horn Clauses) language.

In this section we present the basic definitions of NGHC. The main new feature of NGHC is its concept of guard. In fact any clause can have m layers of guards. Each guard is a pair *I | O (input constraints / output constraints)*, where the input and the output parts are separated by the *commit* (|) symbol. This kind of guard is very similar to the Ask and Tell constraints [S 88, KYKS 88]. Roughly speaking, let us consider the set of the variables of the head of a given clause as the environment of the clause. Then the input guards are expected not to instantiate variables in the environment. They express some conditions on the input values, that must be satisfied. If these input conditions are satisfied by values received from the environment, output guards compute, by means of unification atoms, values that can be exported to the environment. The unification primitive of NGHC is defined as in FGHC. Unlike the language FCP(:,?) [KYKS 88] NGHC unification is not atomic and the commit is done before the tell part of a guard. NGHC is then closer to the GHC model. Unlike GHC, FCP(:,?) and many other languages, the NGHC input constraints are not defined by a suspension rule which depends on the goals and therefore on the external environment of the clause. In fact input constraints are defined using the one-way unification primitve [G 86] as in PARLOG. This, together with the many layers guards, gives rice to nice semantic properties, namely the existence of a complete (i.e. always applicable) set of unfolding rules and, consequently, the existence of an unfolding and an (equivalent) fixpoint semantics.

The *one-way unification primitive* ($\leq$) was defined in PARLOG as follows.

Definiton 1. *(One-way unification)* [G 87]. Let s and t be two first order terms which do not share variables.Given the goal $s \leq t$:

1) If there exists an mgu θ of t and s such that θ does not bind variables in s, then the call to $s \leq t$ succeeds with c.a.s. θ.

2) If t and s can be unified only by binding variables in s then the call to $\leq$ suspends.

3) Otherwise the call to $\leq$ fails.

Definition 2. Let $I_1, I_2, \ldots, I_n$ be n sets of one-way unifications atoms and $O_1, O_2, \ldots, O_n$ n set of standard unification atoms. Then $S = I_1|O_1, \ldots, I_n|O_n$ is a *unification sequence*, or, for short, a *sequence*. An $I_i|O_i$ is an *element* of the sequence. The generic I_i is called *input guard* and O_i *output guard*. Empty, i.e. always satisfied guards are denoted by the atom *true*. ♦

In the following Var(I) will denote the set of variables in I, where I can be a term, a guard, a sequence or a set.

Definition 3. Let $S = I_1|O_1, \ldots, I_n|O_n$ be a sequence with $I_i = \{t_{1,i} \leq s_{1,i}, \ldots, t_{m(i),i} \leq s_{m(i),i}\}$ and let H be a set of variables. S is an *admissible sequence w.r.t.* H iff the following conditions hold:

i) $Var(\{s_{1,i}, \ldots, s_{m(i),i}\}) \cap Var(\{t_{1,i}, \ldots, t_{m(i),i}\}) = \varnothing$.

ii) $Var(\{s_{1,i}, \ldots, s_{m(i),i}\}) \cap Var(H \cup I_1 \cup O_1 \cup \ldots \cup I_{i-1} \cup O_{i-1}) = \varnothing$. ♦

Definition 4. *(NGHC language)* A NGHC program is a finite set of clauses of the form

$$H :\!- I_1|O_1, I_2|O_2, \ldots, I_n|O_n \leftarrow B_1, \ldots, B_m. \quad (n,m \geq 0)$$

where H (head), the I_i's (input guards), the O_j's (output guards) and the B_k's (body atoms) satisfy the following conditions:

i) H is the application of a predicate simbol of arity k to k distinct variables.

ii) I_i is a conjunction of one-way unification atoms and O_i is a conjunction of unification atoms, $i = 1, \ldots, n$.

iii) $I_1|O_1, I_2|O_2, \ldots, I_n|O_n$ is an admissible sequence w.r.t. Var(H).

iv) None of the B_k's is an unification or one-way unification atom.

A unit clause is a clause of the form: $H :\!- I_1|O_1, I_2|O_2, \ldots, I_n|O_n$.

A goal clause is a clause of the form: $:\!- B_1, \ldots, B_m$. $(m \geq 1)$ ♦

2.1 Operational semantics of NGHC.

The operational semantics of NGHC is similar to the one given for FGHC. The definition of computation rule can abstractly be described as in case of FGHC by considering the suspension rule defined by one-way unification instead of the one defined in FGHC. Given a program w and a goal G of the form $:\!- B_1, \ldots, B_m$., the execution of G is the (possibly parallel) reduction of G to the empty goal, using the clauses in w. Computed answer substitutions can be defined as in the case of FGHC. The elementary reduction step in NGHC is nondeterministic AND-parallel resolution and the elementary atomic action of NGHC is the binding of a variable to a term. The definition of the additional rules that must be satisfied by NGHC executions is different from the FGHC one. In fact for NGHC the following rules have to be satisfied:

Sequential evaluation of guards.

The evaluation of I_i precedes the evaluation of O_i, $1 \leq i \leq n$. The evaluation of $I_i \mid O_i$ precedes the evaluation of $I_j \mid O_j$, $1 \leq i < j \leq n$.

Rule of suspension.

The evaluation of $I_i = \{t_1 \leq s_1, \ldots, s_n \leq t_n\}$ is performed by evaluating the atom $(t_1, \ldots, t_n) \leq (s_1, \ldots, s_n)$ as specified by the definition of the primitive $\leq$. Therefore, as in the case of FGHC, if the evaluation of the guard I_i suspends it can later be resumed only when the conditions to successfully evaluate $(t_1, \ldots, t_n) \leq (s_1, \ldots, s_n)$ are satisfied.

The most important difference between NGHC and FGHC (and in general between NGHC and existing committed-choice concurrent logic languages) is in the stucture of nested guards. Because of this structure, the commitment is also quite different. In NGHC the commitment is done by levels as specified by the following rule.

Rule of commitment.

Let us consider all the clauses whose head unifies with A. All these clauses are evaluated in parallel, and are competing for the reduction of A. If $c = H :\!- I_1|O_1, I_2|O_2 \ldots I_n|O_n \leftarrow B$ is one of the competing clauses and we are evaluating I_k and the evaluation succeeds, then *c* tries to be selected for the execution of A. To be selected *c* must first confirm that no other competing clause has been selected for A. If this is the case, *c* is selected and the execution of G commits at level *k* to c, according to the following two possible cases:

i) if k = n then c is selected and the execution of G commits to c.

ii) If k < n the commitment is still not completed. But, any clause whose prefix until level K is different (a part from variable names) from $H :- I_1|O_1, \dots, I_{k-1}|O_{k-1}$ is disregarded.

Notice that the evaluation of the output guard O_i starts after the succesfull evaluation of I_i, and, if it result in a failure, the evaluation of A fails.

2.2 The strong normal form of NGHC programs.

In this section we define a strong normal form for NGHC programs, and in section 3 we will show a normalization procedure which transforms any NGHC program to an equivalent (w.r.t. success set) one. There are two main reasons for defining a strong normal form, to have a minimal representation of NGHC clauses and, as far as possible, to delete wrong clauses, i.e. clauses whose evaluation always fails. This minimal representation is useful to define the semantics of a program. In fact, as sketched in the introduction, our notion of intepretation, and hence of model, is that of a set of NGHC unit clauses. Hence, by considering sets of NGHC clauses in strong normal form, we obtain interpretations and models which satisfy some desiderable minimality property. Let us consider a few examples.

Example 1.
c1) $p(X,Y) :- \{f(X,Y) \leq f(g(Z),b)\} \mid \{f(Z)= f(f(K))\}$, $\{Z \leq f(g(M)\} \mid \{M=b\}$
c2) $p(X,Y) :- \{X \leq g(Z), Y\leq b\} \mid \{Z= f(K)\}, \{K \leq g(M)\} \mid \{M=b\}$.
Because of NGHC operational semantics definition, c1 and c2 are equivalent. Therefore c2, which will be shown to be in strong normal form, can be considered a minimal representation of c1. Notice that each variable X of c2 occurs on the left part of at most one one-way unification or unification atom. ◆

Example 2.
c3) $p(X,Y) :-\{X \leq g(Z), Y\leq b\} \mid \{Z= f(K), W=a\}, \{X\leq g(g(M) \mid \{M=b\}$.
c4) $p(X,Y) :-\{X \leq g(Z), Y\leq b\} \mid true$, $\{K \leq f(a)\} \mid true$.
c3 and c4 are legal NGHC clauses but not in strong normal form. We can easily note that both clauses always fail. In fact, in c3 after the evaluation of the first input and output guard, X is bounded to the term g(f(Z)), while the second input guard requires $X \leq g(g(M))$. In c4, instead, the atom $K \leq f(a)$ in the second input guard will never succeed, since the variable K cannot be instantiated. Therefore in a minimal representation, considering only success cases, the two clauses can be deleted. ◆

Example 3.
c5) $p(X,Y) :- true \mid \{X=a\}, true \mid \{Y=b\}$.
c6) $p(X,Y) :- \{X\leq a\} \mid true, \{Y\leq b\} \mid true$.
Obviously clauses c5 and c6 are equivalent to clauses c7 and c8,
c7) $p(X,Y) :- true \mid \{X=a,Y=b\}$.
c8) $p(X,Y) :- \{X\leq a, Y\leq b\} \mid true$.
which are the strong normal forms of clauses c5 and c6. ◆

Definition 5. *(NGHC strong normal form).* An NGHC program in strong normal form is a finite set of NGHC clauses of the form:

$$H :- I_1|O_1, I_2|O_2, \dots, I_n|O_n \leftarrow B_1, \dots, B_m. \quad (n,m \geq 0)$$

where the I_i's , the O_i 's and the B's satisfy the following conditions:

i) The I_i's are sets of one-way unification atoms of the form $X \leq t$ with X variable and t term. The O_i's are sets of unification atoms of the form $X = t$ with X variable and t term.

ii) There are no one-way unification atoms or unification atoms in $I_1|O_1, I_2|O_2 \dots I_n|O_n$ having the same variable as left term for $i = 1, 2, \dots, n$.

iii) Let V_i be the set of variables defined as follows: $V_i = Var(H \cup I_1 \cup O_1 \cup \cup \dots \cup I_{i-1} \cup O_{i-1})$. Then the left terms of all the atoms in I_i, $i = 1, 2, \dots, n$, are variables occuring in V_i and the left terms of atoms in O_i are variables occuring in $V_i \cup Var(I_i)$.

iv) Variables which are left terms of atoms in I_i, $i = 1, 2, \dots, n$ do not occur in
$O_i, I_{i+1}|O_{i+1}, \dots, I_n|O_n, B_1, \dots, B_m$.
Variables which are left terms of atoms in O_i do not occur in
$I_{i+1}|O_{i+1}, \dots, I_n|O_n, B_1, \dots, B_m$.

v) There does not exist an element $I_i|O_i$, $i = 2, \dots, n$, such that $I_i = true$.
There does not exist an element $I_j|O_j$, $j = 1, \dots, n-1$, such that $O_j = true$.

2.3 Embedding FGHC into NGHC.

It is possible to embed FGHC in NGHC by performing the following steps:

i) Transform an FGHC program in a equivalent (FGHC) program in strong normal form as done in [L 88] (with some modifications). The FGHC strong normal form is quite similar to that one given for NGHC (in FGHC there is only one guard layer and there are no one-way unifications).

ii) Once we have an FGHC program in strong normal form, we can replace unification atoms in the guard with one-way unifications to get an FGHC program.

Here we are not interested in the details of this transformation. Let us consider just an example of such a transformation.

Example 4. Let us consider the FGHC program P:

$P\ =\ \{\ p(X,Y,Z)\ {:}{-}\ X{=}f(Y),\ /\ Z{=}g(X),\ Y{=}b\ ,\ q(Y,Z)$

$\qquad q(b,f(Z))\ {:}{-}\ true\ /\ true.\ \}$

The equivalent FGHC program in strong normal form is:

$P'\ =\ \{\ p(X,Y,Z)\ \ {:}{-}\ X{=}f(K),Y{=}K\ /\ Z{=}g(f(b),\ K{=}b\ ,\ q(b,g(f(b)))$

$\qquad q(X,Y)\ {:}{-}\ X{=}b,\ Y\ =\ f(Z),\ /\ true.\ \}$

From this program we can get the equivalent NGHC program simply by replacing = by ≤ in the input guard:

$P''\ =\ \{\ p(X,Y,Z)\ \ {:}{-}\ \{X{\leq}f(K),\ Y{\leq}\ K\}\ /\ \{Z{=}g(f(b)\}\ \ \leftarrow q(f(b),g(f(b)))$

$\qquad q(X,Y)\ {:}{-}\ \{X{\leq}b,\ Y\ \leq f(Z)\}\ /\ true.\ \}$

Notice that we can subsitute a unification atom with a one-way unification atom to get an equivalent NGHC program, only if the original FGHC program is in strong normal form. In fact, the program obtained from w by performing the replacement of =by ≤ only, is the following program w1 (the second clause of w has been normalized to have variables in the head):

$P'''\ =\ \{\ p(X,Y,Z)\ {:}{-}\ \{X{\leq}\ f(Y)\}\ /\ \{Z{=}g(X),\ Y{=}b\}\ \leftarrow q(X,Z)$

$\qquad q(X,Y)\ {:}{-}\ \{X{\leq}b,\ Y\ \leq f(Z)\}\ /\ true.\}$

This program is not a legal NGHC program. Moreover P''' is not equivalent to P since the goal p(f(b),Y,Z) succeeds in P''' and fails (because of a deadlock) in the original program P. ◆

3. The normalization of NGHC programs.

Any NGHC program can be transformed into an equivalent (w.r.t. success set) program in strong normal form. In the previous examples 1 and 2, c2 is the normalized version of c1, while the normalized version both of c3 and c4 is the empty clause. Let us now consider a non unit clause.

Example 5.

c1) $\quad p(X,Y)\ {:}{-}\ \{f(X,Y)\ \leq f(g(Z),b)\}\ \ /\ \{f(Z){=}\ f(f(K))\}\ \leftarrow\ \ q(X,Y).$

Clearly when we evaluate the body of the clause, X is bound to g(f(K)) and Y to b. We can then consider the following clause c2 which is in strong normal form and which is equivalent (w.r.t. success set) to c1, as the normalized version of c1

c2) $\quad p(X,Y)\ {:}{-}\ \{X\ \leq g(Z),\ Y{\leq}\ b\}\ /\ \{Z{=}\ f(K)\}\ \ \leftarrow\ \ q(g(f(K)),b).$ ◆

Let us formally define a normalized sequence as follows.

Definition 6. Let S be a sequence and let H = $\{X_1, \dots ,X_h\}$ be a set of variables. S is ***normalized*** w.r.t. H iff $p(X_1, \dots ,X_h)$:- S is an NGHC unit clause in strong normal form, where p is any predicate symbol. ◆

In the appendix we define a NORM(S,H) procedure which derives from a sequence S admissible w.r.t. the set of variables H, an equivalent sequence S_1 which is normalized w.r.t. H. Using the procedure NORM(S,H) it is straightforward to define an algorithm which transforms an NGHC program to an NGHC program in strong normal form.

Definition 7. *(Normalization of an NGHC program).* Let c be an NGHC clause

$c = p(X_1, X_2, \dots, X_r)$:- $I_1|O_1, I_2|O_2, \dots, I_n|O_n \leftarrow B_1, \dots, B_m$.

We define the clause c_Nor resulting from the *normalization of the clause* c as follows.

i) If NORM($(I_1|O_1, I_2|O_2, \dots, I_n|O_n), \{X_1, \dots, X_n\}$) = *fail* then c_Nor = *true* (empty clause).

ii) If NORM($(I_1|O_1, I_2|O_2, \dots, I_n|O_n), \{X_1, \dots, X_n\}$) = $I\_Nor_1|O\_Nor_1, \dots, I\_Nor_{n(i)}|O\_Nor_{n(i)} \neq$ *fail* then

c_Nor = $p(X_1, X_2, \dots, X_r)$:- $I\_Nor_1|O\_Nor_1, \dots, I\_Nor_h|O\_Nor_h \leftarrow B'_1, \dots, B'_m$

where $B'_j = B_j\,\theta$, $j = 1, \dots, m$, with $\theta = \alpha_1\beta_1\alpha_2\beta_2 \dots \alpha_h\,\beta_h$ and $\alpha_i = \{X/t \mid X \leq t \in I\_Nor_i\}$, $\beta_i = \{X/t \mid X \leq t \in O\_Nor_i\}$, $i=1,2,\dots,h$.

If *w* is an NGHC program, we define the normalization of*w* as the program resulting from the normalization of every clause in w. ♦

It is possible to show that a program and its normalized version are equivalent w.r.t. success set. In fact the following theorem holds.

Theorem 1. Let P be an NGHC program and let P' be the result of the normalization of P. Then the evaluation of G in P succeeds with c.a.s. θ iff the evaluation of G in P' succeeds with c.a.s. θ', where θ' is a variant of θ. ♦

P' is not equivalent to P as far as failures are considered. In fact a goal G can fail in P and not in P' (see example 2). If we are interested in the success set semantics only, deleting failures does not matter. Therefore we can use normalization in the definition of the unfolding rules for NGHC, and then we can safely use unfolding for giving a succes set semantics to NGHC programs. If we were interested in failures too, we could keep the always failing clauses detected at normalization time in a separate set, and from this get the necessary information about failures.

4. A further reduction for NGHC programs in strong normal form.

We have shown how to obtain the minimal version of an NGHC clause by means of normalization. If we consider programs instead of single clauses, it is possible to define a further reduction by eliminating clauses which are redundant, to obtain a minimal version of an NGHC program. Let us consider an example.

Example 6.

c1) *p(X,Z) :- {X≤f(Y) Z≤f(b)}| {Y=a}.*

c2) *p(X,Z) :- {Z≤f(K)} | {K=b, X=f(a)}.*

The program w = {c1,c2} and the program w' = {c2} are equivalent w.r.t. success set. In fact for any goal, since clause c2 is "less constrained" than c1, if the input guard of c1 is satisfied the input guard of c2 is satisfied too. Moreover the substitution computed by c1 is the same as the one computed by c2. Therefore we can delete clause c1 in program w obtaining an equivalent (w.r.t. success set) program. ♦

In the following we define a relation of subsumption between NGHC clauses in strong normal form, which is an extension of the usual subsumption for first order formulas. By means of subsumption, we define an equivalence w.r.t. success set among NGHC programs in strong normal form. By means of this equivalence relation we can reduce, as in the previous example, the number of clauses of an NGHC program. This is particularly useful to control the explosion in the number of clauses, when unfolding a program.

Definition 8. Let s and t be first order terms. *t subsumes* s* and ($t \supseteq_* s$) iff there exists an idempotent substitution θ such that $t\theta = s$ and θ does not bind variables in s. ♦

Notice that this definition is a special case of definition a1.

Definition 9. Let c_i, c_j be two NGHC clauses in strong normal form :

$c_i = p(X_1, \dots X_m)$:- $I_{i,1}|O_{i,1}, \dots I_{i,n(i)} \mid O_{i,n(i)} \leftarrow Body_i$

$c_j = p(X_1, \dots X_m)$:- $I_{j,1}|O_{j,1}, \dots I_{j,n(j)} \mid O_{j,n(j)} \leftarrow Body_j$.

which share variables $X_1, \dots, X_m$ only. c_i *subsumes* c_j ($c_i \supseteq_{clause} c_j$) iff the following algorithm ends successfully.

Let $\alpha_{i,h} = \{X/t \mid X \leq t \in I_{i,h}\}$, $\beta_{i,h} = \{X/t \mid X \leq t \in O_{i,h}\}$, $h = 1,2, \dots, n(i)$ and

$\alpha_{j,h} = \{X/t \mid X \leq t \in I_{j,h}\}$, $\beta_{j,h} = \{X/t \mid X \leq t \in O_{j,h}\}$.

a) For $s = 1,2, \dots, \max(n(i),n(j))$

i) $(X_1, \ldots, X_m)\,\alpha_{i,1}\,\beta_{i,1} \cdots \alpha_{i,s-1}\,\beta_{i,s-1}\,\alpha_{i,s} \supseteq^* (X_1, \ldots, X_m)\alpha_{j,1}\,\beta_{j,1} \cdots \alpha_{j,s-1}\,\beta_{j,s-1}\,\alpha_{j,s}$.

ii) $(X_1, \ldots, X_m)\alpha_{j,1}\,\beta_{j,1} \cdots \alpha_{j,s}\,\beta_{j,s} \supseteq^* (X_1, \ldots, X_m)\,\alpha_{i,1}\,\beta_{i,1} \cdots \alpha_{i,s}\,\beta_{i,s}$.

b) $(X_1, \ldots, X_m)\,\alpha_{i,1}\,\beta_{i,1} \cdots \alpha_{i,n(i)}\,\beta_{i,n(i)}\,\rho = (X_1, \ldots, X_m)\,\alpha_{j,1}\,\beta_{j,1} \cdots \alpha_{j,n(j)}\,\beta_{j,n(j)}$
where ρ is a variables renaming.

c) $(\text{Body}_i)\,\rho\delta = (\text{Body}_j)$ where ρ is the same of step b) and δ is a renaming such that
$\text{Dom}(\delta) \cap \text{Var}(H \cup I_{i,1} \cup O_{i,1} \cup \ldots \cup I_{i,h} \cup O_{i,h} \cup \text{Cod}(\rho)) = \varnothing$ and
$\text{Cod}(\delta) \cap \text{Var}(H \cup I_{i,1} \cup O_{i,1} \cup \ldots \cup I_{i,h} \cup O_{i,h} \cup \text{Cod}(\rho)) = \varnothing$ where
$\text{Dom}(\{X_1/t_1, \ldots, X_n/t_n\}) = \{X_1, \ldots, X_n\}$ and $\text{Cod}(\{X_1/t_1, \ldots, X_n/t_n\}) = \text{Var}(\{t_1, t_2, \ldots, t_n\})$. ♦

The following theorem shows that we can safely eliminate a clause c_j from a program P if there exists another clause c_i in P such that $c_i \supseteq_{\text{clause}} c_j$.

Theorem 2. Let c_i and c_j be clauses of the program w and let $c_i \supseteq_{\text{clause}} c_j$. Then if the program w' is w / $\{c_j\}$, w' and w are equivalent w.r.t. success set. ♦

Definition 10. *(Minimalization).* Let P be a set of NGHC clauses in strong normal form. We define MIN(P) as the set $\text{MIN}(P) = \{c \mid c \in P$ and there does not exist $c' \in P$ such that $c' \supseteq_{\text{clause}} c\}$. ♦

Minimalization will be used in the definition of unfolding. The correctness of such transformation follows from the previous theorem, which can obviously be extended to show that a program P and MIN(P) are equivalent w.r.t. success set.

5. Unfolding rules for NGHC programs.

In this section we define the unfolding rules for NGHC programs. Because of the definition of NGHC, namely because it has several layers of guards, this definition is quite easy and natural. Our set of rules is complete, i.e. our unfolding rules are always applicable, and it is possible to prove that the unfolding of a NGHC program P is an NGHC program P' which is equivalent w.r.t. success set to P. Since it is possible to embed FGHC in NGHC, our rules can be used for FGHC also (by first transforming an FGHC program into an NGHC program). In the literature there are several attempts to define a set of unfolding rules for GHC like languages[FOF 87, FOM 87, U 88], but none of the attempts results in a complete set of rules. We believe that the GHC syntax is not powerful enough to express its own semantics, which has to be preserved in the unfolding process. Namely, the FGHC operational semantics defines, for synchronization reasons, an order in the evaluation of guards of different clauses, and sometimes it is not possible to reduce this sequence of guards to a unique guard. Therefore, ithe only way to express synchronization in these cases is to keep separate clauses. Let us consider an example from [L 88a]

Example 7. Let w be an FGHC program containing the following clauses:

c1) *s(X,Y) :- true / X=a, t(a,Y)*
c2) *t(X,Y) :- Y=b / true*

The existing unfolding rules do not allow the unfolding of c1 using c2. In fact, the clause c3

c3) *s(X,Y) :- Y=b /X=a*

which could be the result of the unfolding of c1, is equivalent to c1 only in the deadlock-free version of FGHC [L 88]. However, in the real FGHC case, c1 and c3 are equivalent only if we consider goals consisting of a single call to s. In fact, if s occurs in conjunction with other goals, the order of evaluation of the unification atoms X=a and Y=b is relevant to synchronization. For example, in the program

P = { c1) s(X,Y) :- true / X=a, t(a,Y).
c2) t(X,Y) :- Y=b / true.
c4) p(X,Y) :- X=a / Y=b.}

the goal s(X,Y),p(X,Y) succeeds, while it fails (because of a deadlock) in the program obtained from P by replacing clause c1 with c3. The correct unfolding of c1 should be the (non FGHC) clause

c5) *s(X,Y) :- true / X = a , Y = b / true*

which tells us that in the evaluation of s(X,Y) <u>first</u> the (output) unification atom X = a is evaluated and <u>then</u> the (input) atom Y=b. If we consider ≤ instead of = in the guards, P becomes an equivalent NGHC program P' and in P', the unfolding of c1 is really the NGHC clause:

c6) *s(X,Y) :- true | {X = a}, {Y ≤ b} | true.* ♦

Notice that it is not possible to solve this problem by simply defining an extension of FGHC which allows several layers of guards (as in NGHC) and keeps the suspension rule of FGHC. In fact such a rule depends on the goals (guard unification cannot bind variables which appear in the goal) which are not known at unfolding time. Let us now consider the standard definition of interleaving, an then the one of unfolding.

Definition 11. Let $S_1, S_2, \dots, S_n$ be n sequences with $S_j = I^j_1|O^j_1, \dots, I^j_{n(j)}|O^j_{n(j)}$. The *interleaving* $S_1 \parallel S_2 \parallel \dots \parallel S_n$ is the set of sequences:

INT = { S_{int} | *i)* S_{int} is a sequence made with all the elements of all the sequences $S_1, \dots, S_n$,
ii) the element $I^j_h|O^j_h$ precedes $I^j_k|O^j_k$ for $1 \le h < k \le n(j)$, $1 \le j \le n$,
iii) $I^j_h|O^j_h$ precedes $I^i_k|O^i_k$ for $1 \le h \le n(j)$, $1 \le k \le n(i)$, $1 \le i < j \le n$.} ♦

Definition 12. *(Unfolding Rules).* Let c be a clause in the NGHC program w

$c = p(Y_1, \dots Y_k) \text{:-}\ I_1|O_1, I_2|O_2, \dots, I_n|O_n \leftarrow M_1, M_2 \dots M_z$.

Let H be the set $H = \{Y_1, \dots Y_k\}$. The *unfolding* of *c* w.r.t. w is the set Unf(c,w) of clauses defined as follows:

i) If z = 0 (c is a unit clause) then Unf(c,w) = {c}

ii) If z > 0 let ALL-INT(c) be the set of clauses defined as follows:

ALL-INT (c) = $\{p(Y_1, \dots Y_k) \text{:-}\ I_1|O_1, I_2|O_2, \dots, I_n|O_n, S\_int \leftarrow$
$G^1_1 \dots G^1_{s(1)}, G^2_1, \dots, G^2_{s(2)}, \dots, G^m_1, \dots, G^m_{s(m)}, B_1, \dots, B_s$

such that

i) $\{p_1(t^1_1 \dots t^1_{s(1)}), \dots, p_m(t^m_1 \dots t^m_{s(m)})\}$ is a subset of $\{M_1, M_2 \dots M_z\}$

ii) $C = (c_1, \dots, c_m)$ is a m-tuple of clauses in w where

$c_j = p_j(X^j_1 \dots X^j_{s(j)}) \text{:-}\ I^j_1|O^j_1, \dots, I^j_{n(j)}|O^j_{n(j)} \leftarrow G^j_1 \dots G^j_{s(j)}$

iii) $\{B_1, \dots B_s\} = \{M_1, M_2 \dots M_z\} / \{p_1(t^1_1 \dots t^1_{s(1)}), \dots, p_m(t^m_1 \dots t^m_{s(m)})\}$

iv) S_int is a sequence of the interleaving:

$(\text{true} \mid O_{call\_1}, I^1_1|O^1_1, \dots, I^1_{n(1)}|O^1_{n(1)}) \parallel$
$(\text{true} \mid O_{call\_2}, I^2_1|O^2_1, \dots, I^2_{n(2)}|O^2_{n(2)}) \parallel$
:
$(\text{true} \mid O_{call\_m}, I^m_1|O^m_1, \dots, I^m_{n(m)}|O^m_{n(m)})$

where $O_{call\_i} = \{X^i_1 = t^i_1, \dots, X^i_{s(i)} = t^i_{s(i)}\}$ }.

We define Unf(c,w) as the set of clauses resulting from the minimalization and normalization of the clauses of ALL-INT(c_i) for every clause c_i, i. e.

Unf(c,w) = MIN{c_nor | c_nor is the normalization of c_int, with c_int∈ $\cup_{cl \in w}$ ALL-INT(cl) }. ♦

It is straightforward to note that every clause in ALL-INT(c) is an NGHC clause. We can then consider the normalization of such a clause and Unfolding is well defined.

Definition 13. Let P be the NGHC program $\{c_1, \dots c_n\}$. Then its *unfolding*, denoted by *Unf*(P), is the collection of clauses *Unf(P)* = $\cup_{i=1 \dots n}$ Unf(c_i,P). ♦

Example 8. Let us consider the NGHC program

P = { t(X,Y) :- {Y ≤ b} | true.
r(X,Y) :- {X ≤ a} | {Y=b}
s(X,Y) :- true | {X=a} ← t(a,Y).
q(X,Y) :- true | true ← r(X,Y), s(X,Y). }

The unfolding of P is the following program

P' = { t(X,Y) :- {Y ≤ b} | true.
r(X,Y) :- {X ≤ a} | {Y=b}.

s(X,Y) :- true / {X=a}, {Y ≤ b} / true.
q(X,Y) :- {X ≤ a} / {Y=b} ← s(a,b)
q(X,Y) :- true / {X=a , Y=b} ← t(a,b)
q(X,Y) :- true / {X=a} ← r(a,Y) t(a,Y). } ♦

The definition of unfolding guarantees that the unfolded version Unf(P) of an NGHC program P, is an NGHC program in strong normal form. It is possible to show that P and Unf(P) are equivalent w.r.t. the success set. In fact, the following theorem holds.

Theorem 3. Let P be an NGHC program and let Unf(P) be the result of the unfolding of P. Then the evaluation of G in P succeeds with c.a.s. θ iff the evaluation of G in Unf(P) succeeds with c.a.s. θ' such that θ' is a variant of θ. ♦

6. Unfolding and Fixpoint semantics.

As previously sketched, we are interested to unfolding more for semantics concerns rather than as a program transformation technique. As pointed out in [L 88a, LM 88], if the unfolding is well defined, i.e. if the result of unfolding a logic program written in the language L is an equivalent logic program in the language L, and if the unfolding rules can always be applied, we can talk about the infinite unfolding of a logic program and we can use this to characterize the meaning of a program in an alternative way. In fact, if we collect the unit clauses that are derived at each unfolding step of an infinite unfolding process, we obtain a set which defines the unfolding semantics of the program, and which in [LM 88], for pure HCL programs, is shown to be equal to the minimal S_Model (minimal Herbrand model with variables, see [FLMP 88]).

This method for giving a semantics of logic programs starting from purely operational argoments, is particularly useful when we are faced with extended logic languages. In fact, most of them exhibit a well understood operational behaviour, whereas it is often difficult to find a declarative counterpart. By the definition of an unfolding semantics we can subsequently derive a fixpoint semantics, defining a suitable notion of model or interpretation. In particular, as shown in [L 88], the right notion of interpretation should be one which allows to observe the interesting operational properties. In this sense, standard Herbrand interpretations are not adequate [FLMP 88]. Therefore we consider interpretations which are sets of unit clauses. This allows to obtain a fixpoint semantics which is equivalent to the unfolding semantics, and, therefore, to the operational semantics.

Example 9. If we consider programs P and P' of example 8, we note that further unfoldings of P' derive the programs P'' and P''':

P''= { t(X,Y) :- {Y ≤ b} / true.
r(X,Y) :- {X ≤ a} / {Y=b}.
s(X,Y) :- true / {X=a}, {Y ≤ b} / true.
q(X,Y) :- -true / {X=a, Y=b}.
q(X,Y) :- -true / {X=a, Y=b} ← t(a,b).
q(X,Y) :- -true / {X=a}, {Y ≤ b} / true← r(a,b).
P'''= { t(X,Y) :- {Y ≤ b} / true.
r(X,Y) :- {X ≤ a} / {Y=b}.
s(X,Y) :- true / {X=a}, {Y ≤ b} / true.
q(X,Y) :- -true / {X=a , Y=b}. }

P''' is a model of the program P. In fact, P''' shows, for example, that the goal ?-s(X,Y). fails, that the goal ?-s(X,b). succeeds computing the substitution (X=a), while the goal ?-q(X,Y). succeeds computing the substitution (X=a, Y=b). In other words P''' is the unfolding semantics of program P. ♦

Example 10. Consider the NGHC program W:

W = { r(X,Y) :- {X ≤ a} / {Y=b}.
s(X,Y) :- {Y≤ b} / {X=a}.
q(X,Y) :- true / true ← r(X,Y), s(X,Y). }

Two successive unfoldings of W produce the following program:

W''= {r(X,Y) :- {X ≤ a} / {Y=b}.
s(X,Y) :- {Y≤ b} / {X=a}.
q(X,Y) :- {X ≤ a} / {Y=b}.
q(X,Y) :- {Y≤ b} / {X=a}. }

which is the unfolding model of W and which shows, for example, that the goal ? :- s(X,Y). and the goal ? :- q(X,Y) fails, while the goal ? :- q(X,b). succeeds computing the substitution (X=a). ♦

As already mentioned, the right notion of interpretation is that of a (possibly infinite) set of unit clauses. Let us now formally define our notion of interpretation. Afterwards we will define the unfolding semantics based on this notion.

Definition 14. Let A_P be the set of all the NGHC unit clauses in strong normal form, defined by the function and predicate symbols occurring in the program P. Let $\sim^*$ be the equivalence relation defined as follows: if $c1,c2 \in P$ then $c1 \sim^* c2$ iff $c1 \sqsupseteq_{clause} c2$ and $c2 \sqsupseteq_{clause} c1$. The *interpretation base* B_P is defined as $A_{/\sim^*}$, i.e. the quotient set of A with respect to the equivalence relation $\sim^*$. ♦

Definition 15. A *guarded interpretation* for a program P is MIN(I) where I is any subset of B_P. The set of all the guarded interpretations of P is denoted by I_P. ♦

Lemma 1. I_P is a *complete lattice* with respect to the partial order $\leq^*$ defined as follows:
if $X,Y \in I_P$, $X \leq^* Y$ iff for each $c_X \in X$ there exists $c_Y \in Y$ such that $c_Y \sqsupseteq_{clause} c_X$.
The bottom element of I_P is $\varnothing$, the top element is $MIN(B_P)$, lub and glb are defined as $lub(X) = MIN(\cup(X))$ and $glb(X) = MIN(\cap(X))$ where $X \subseteq I_P$. ♦

Definition 16. Let P be an NGHC program. The following is an inductive definition of a collection of programs which are equivalent :
$P_0 = P$
$P_{i+1} = \mathit{Unf}(P_i)$.
Moreover, given a program P_i in the previous collection, let us denote by U_i the (guarded) interpretation:
$U_i = \{A :- I_1 | O_1, I_2 | O_2 \dots I_n | O_n \mid A :- I_1 | O_1, I_2 | O_2 \dots I_n | O_n$ is a unit clause of $P_i\}$. ♦

Definition 17. *(Unfolding semantics).* Let P be a NGHC program. Then the *unfolding semantics* of P, denoted by *U*(P), is the set $U(P) = \cup_{i \in \omega} U_i$. ♦

Definition 18. Let P be an NGHC program. The *mapping* T_{UP} on the set of guarded interpretations of P is defined as follow:
$T_{UP}(J) = \{A :- I_1 | O_1, I_2 | O_2 \dots I_n | O_n \in H$, such that $\exists$ a clause $c = A :- \Gamma'_1 | O'_1, \dots, \Gamma'_m | O'_m \leftarrow B$ in p, such that $(A :- I_1 | O_1, I_2 | O_2 \dots I_n | O_n) \in Unf(c,J)\}$. ♦

The following are standard results which allow to define a fixpoint semantics based on T_{UP}.

Theorem 4. T_{UP} is monotonic and continuous. ♦

Theorem 5. There exists the least fixpoint of T_{UP} $\mathit{lfp}(T_{UP}) = \cup_{n \in \omega} T_{UP}^n(\varnothing) = T_{UP} \uparrow \omega$ ♦

Definition 19. *(Fixpoint semantics).* The *fixpoint semantics* Fix(P) of an NGHC program P, is the least fixpoint of the transformation T_{UP} associated to the program P. ♦

The following theorems prove the equivalence (w.r.t. the success set) of the operational, the Unfolding and the Fixpoint semantics . Before discussing them, let us recall which is the meaning of a guarded interpretation. It should be clear, from our discussion and from the examples we have shown that, given an NGHC goal, we can "execute" it in a guarded intepretation since a guarded interpretation is an NGHC program. For instance, the execution of the goal q(a,Y) (example 10) in the model succeeds with c.a.s. {Y=b} exactly as the evaluation of q(a,Y) in the program P does. Therefore our equivalence results show that if the refutation of a goal G in the program P derives the c.a.s. θ, (a variant of) such c.a.s. can be obtained by executing G in the model. The following theorem holds for atomic goals, but can easily be extended to generic goals.

Theorem 6. *(Equivalence between the Operational and the Unfolding semantics)* Let P be an NGHC program and $A = a(t_1,...t_n)$ be an atomic goal for P. Then G has a refutation with c.a.s. θ iff there exists a (unit) clause $c1 = a(X_1,...X_n) :- I_1 \mid O_1 , ..., I_m \mid O_m$ belonging to *U(P)* such that the (sequential) evaluation of the goal A in the program {c1} succeeds with c.a.s. θ', such that θ' is a variant of θ. ◊

The following theorem shows the equivalence between the unfolding and the fixpoint semantics.

Theorem 7. *(Equivalence between Fixpoint and Unfolding semantics).* Let be P a NGHC program. Then U(P) = Fix(P). ◊

References.

[B 86] L. Beckman. Towards a Formal Semantics for Concurrent Logic Programming Languages, *Prooc. of the Third Int. Conference on Logic Programming,* Lectures Notes in Computer Science 225, Springer Verlag, (1986), 335-349.

[BK 88] J.W. de Bakker and J.N. Kok. Uniform Abstraction, Atomicity and Contractions in the Comparative Semantics of Concurrent Prolog. *Prooc. of the Int. Conference on Fifth Generation Computer Systems,* Tokyo (1988), 347-355.

[BKPR 89] F.S. de Boer, J.N. Kok, C. Palamidessi and J.M.M. Rutten. Semantic models for a version of PARLOG. To appear in *Proc. of the Sixth Int. Conf. on Logic Programming,* Lisboa, Portugal, 1989.

[C 86] P.L. Curien. Categorical Combinators, Sequential Algorithms and Functional Programming, Research Notes in Theoretical Computer Science, Pitman, London, 1986.

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

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

[FLMP 88] M. Falaschi, G. Levi, M. Martelli and C. Palamidessi. A new declarative semantics for logic languages. *Proc. Fifth Int'l Conf. on Logic Programming.* The MIT Press, Series in Logic Programming, (1988), 993-1005.

[FOF 88] H. Fujita, A. Okumura and K. Furukawa. Partial evaluation of GHC programs based on the UR-Set with constraints. *Proc. Fifth Int'l Conf. on Logic Programming.* The MIT Press, Series in Logic Programming 1988, 924-941.

[FOM 87] K. Furukawa, A. Okumura and M. Murakami. Unfolding rules for GHC programs. In D. Bjorner, A.P. Ershow and N.D. Jones, editors, *Workshop on Partial Evaluation and Mixed Computation,* Gl. Avernaes, Denmark, October 1987. In *New Generation Computing,* 1988, 143-158.

[G 87] S. Gregory. *Parallel Logic Programming in PARLOG: the Language and its Implementation,* Addison-Wesley, Reading, MA, 1987.

[K] J. N. Kok. A compositional semantics for Concurrent Prolog. *Prooc. of the Symp. on Theoretical Aspects of Computer Science,* 1988 (r. Cori ed.), Lectures Notes in Computer Science 294, Springer Verlag 1988, 373-388.

[KYKS 88] S. Kliger, E. Yardeni, K. Kahn, E. Shapiro. The language FCP(:,?), *Prooc. of the Int. Conference on Fifth Generation Computer Systems,* Tokyo 1988, 763-773.

[L 88] G. Levi. A new declarative semantics of Flat Guarded Horn Clauses. Technical Report, ICOT, January 1988.

[L 88a] G. Levi. Models, Unfolding Rules and Fixpoint Semantics. *Proc. Fifth Int'l Conf. on Logic Programming.* The MIT Press, Series in Logic Programming 1988, 1649-1665.

[Ll 87] J.W. Lloyd, *Foundations of logic programming* , second edition Springer-Verlag, 1987.

[LM 88] G. Levi and P. Mancarella. The unfolding semantics of logic programs. Dipartimento di Informatica, Università di Pisa, Techn. Report, TR-13/88, June 1988.

[LP 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, 128-137.

[LP 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, 877-893.

[M 88] M. Murakami. A declarative Semantics of Parallel Logic Programs with Perpetual Processes. *Prooc. of the Int. Conference on Fifth Generation Computer Systems.* Tokyo 1988, 374-381.

[P 88] C. Palamidessi. A Fixpoint Semantics for Guarded Horn Clauses. Technical Report CS-R8833, Centre for Mathematics and Computer Science, Amsterdam, 1988.

[PM 88] D.S. Parker and R.R. Muntz. A theory of directed logic programs and streams. *Proc. Fifth Int'l Conf. on Logic Programming*. The MIT Press, Series in Logic Programming, 1988, 620-650.

[S 85] V.A. Saraswat. Partial Correctness Semantics for CP(∅, |, &), in *Prooc. of the Conf. on Foundations of Software Computing and Theoretical Computer Science*, LNCS, 206, 1985, 347-368.

[S 87] V.A. Saraswat. GHC: Operational semantics, problems and relationship with CP(↓,|) in IEEE Int. Symp. on Logic Programming, San Francisco 1987, pp347-358.

[S 87a] V.A. Saraswat: The concurrent logic programming language CP: definition and operational semantics, in *Conference Record of the Fourteenth Annual ACM Symposium on Principles of Programming Languages*, Munich, West Germany, 1987, 49-62.

[S 88] V. A, Saraswat, A somewhat logical formulation of CLP synchronization primitives, *Proc. Fifth Int'l Conf. on Logic Programming*. The MIT Press, Series in Logic Programming, 1988, 1298-13314.

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

[Sh 88] 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, 277-313.

[Sh 89] E. Y. Shapiro. The Family of Concurrent Logic Programming Languages. Techn. Rep., The Weizmann Institute of Science, 1989.

[TS 84] H. Tamaki and T. Sato. Unfold/Fold transformation of logic programs. *Proc. 2nd Int'l Logic Programming Conference*. Uppsala, Sweden, 1984, 127-138.

[U 87] K. Ueda. Guarded Horn Clauses. In E. Wada, editor, *Proc. Logic Programming '85*. Lecture Notes in Computer Science 22, Springer-Verlag 1986, 168-179. Also in E. Y. Shapiro, editor, *Concurrent Prolog: Collected Papers*, chapter 4, the MIT Press, 1987.

[U 87a] K. Ueda. Guarded Horn Clauses: A parallel logic programming language with the concept of a guard. ICOT Techn. Rept. TR-208, 1986. Revised in 1987. Also in M. Nivat and K. Fuchi, editors,*Proc. Programming of Future Generation Computers*, North-Holland, 1987, 441-456.

[U 88] K. Ueda, K. Furukawa. Trasformation rules for GHC Programs. *Prooc. of the Int. Conference on Fifth Generation Computer Systems*, Tokyo 1988, 582-591.

Appendix.

In the following, we define a NORM procedure which derives from a sequence S admissible w.r.t. the set of variables H, an equivalent sequence S_1 which is normalized w.r.t. H, i.e. such that $p(X_1, \dots X_n)$:- S_1 is a unit NGHC clause in strong normal form. The normalization procedure is an extension of the one defined in [L 88] for the case of FGHC programs. The essential difference is in the definition of the normalization of an input guard since in FGHC (input) guards contain standard unifications and the constraints on the possible bindings of variables are expressed by a suspension rule, while NGHC input guards consist of one-way unification atoms. Before showing the procedure, let us give a few definitions.

A.1 Definitions.

Definition a1. [PM 88]. Let s and t be first order terms. t *subsumes* s ($t \sqsupseteq s$) iff there exists an idempotent substitution θ such that $t\theta = s$. ♦

Definiton a2. [PM 88]. Let s and t be first order terms. An *orderer of t over s* is an idempotent substitution θ such that $t\,\theta \sqsupseteq s\theta$. θ is a *most general orderer (mgo)* of t over s iff for any other orderer σ there exists a substitution μ such that $\theta\,\mu = \sigma$.

♦

Definition a3. [PM 88]. Let S be a set of first order terms. An *impartial substitution* for S is an idempotent substitution $\theta = \{X_1/u_1, \dots, X_n/u_n\}$ where u_i, $1 \le i \le n$, does not contain variables of any term in S. ♦

It is easy show that if two terms s and t are unifiable, they have a *most general impartial unifier (mgiu)*. Namely if t and s have an idempotent mgu $\theta = \{X_1/u_1, \dots, X_n/u_n\}$ and ρ is a substituion which renames variables in $\{u_1, \dots, u_n\}$ with variables not in s and t, $\theta\,\rho$ is a mgiu for t and s.
The following theorem shows how to obtain an mgo for t over s computing the most general impartial unifier (mgiu) for t and s.

Theorem a1. [PM 88] Let θ be a mgiu for s and t. Define $\theta_{[s,t]} = \{X/u \mid X/u \in \theta$ and X is a variable that appears in t$\}$. Then $\theta_{[s,t]}$ is a mgo of s over t (i.e. $s\,\theta_{[s,t]} \sqsupseteq t\,\theta_{[s,t]}$). ♦

As already mentioned the NGHC interpretation of an input guard $I_i = \{s_1 \le t_1, \dots, s_n \le t_n\}$ is $s \le t$ where $s = (s_1, \dots, s_n)$ and $t = (t_1, \dots, t_n)$. For the definiton of NGHC it is $Var(t_1, \dots, t_n) \cap VAR(s_1, \dots, s_n) = \emptyset$. We can then consider the normalization of the one-way unification atom $s \le t$ where t and s do not share variables.

As pointed out in [PM 88], the primitive $\le$ defined in PARLOG is semantically very close to the partial ordering of subsumption among terms ($\sqsupseteq$). In fact, if we consider idempotent substitutions as we do, $s \le t$ succeeds iff $t \sqsupseteq s$. In particular, when s and t are two terms that have no shared variables, we can use theorem 1 to get an mgo of s over t, without binding variables in s, since in this case $s\,\theta_{[s,t]} = s$. We will use this fact to normalize a generic constraint $s \le t$.

In fact, we normalize an atom $s \le t$, where s and t do not share variables, computing an mgiu $\theta = \theta_{[s,t]} \cup (\theta - \theta_{[s,t]})$ for s and t. If such an mgiu does not exist, the evaluation of $s \le t$ always fails, hence the normalization of $s \le t$ (and of the sequence containing it) results in the fail value. If $t \sqsupseteq s$ we can already succesfully evaluate the atom $s \le t$, i.e. $(\theta - \theta_{[s,t]})$ is only a variable renaming. Then the atom (equivalent to $s \le t$) resulting from the normalization of $s \le t$ is *true* and $\theta_{[s,t]}\,(\theta - \theta_{[s,t]})^{-1}$ is the substitution obtained from the evaluation of $s \le t$ which has to be passed to the rest of the sequence. Otherwise, if $s \sqsupseteq t$, (a simplified version of) $s \le t\,\theta_{[s,t]}$ is the normalization of $s \le t$ and $\theta_{[s,t]}$ is the output substitution. We can simplify $s \le t\,\theta_{[s,t]}$ in a straightforward way to obtain an atom $(X_1, \dots, X_n) \le (m_1, \dots, m_n)$ which defines n atoms in strong normal form, since we have, by construction, $s \sqsupseteq t\,\theta_{[s,t]}$.

Definition a4. Given two terms $t = f(t_1, \dots, t_n)$ and $s = f(s_1, \dots, s_n)$, a subterm t^* of t *corresponds in s* to a subterm s^* (of s) iff either
1) ($t^* = t_i$ and $s^* = s_i$) or
2) (t^* is a subterm of t_i, s^* is a subterm of s_i and t^* corresponds in s_i to s^*). ♦

A.2 The NORM(S,H) procedure.

The following NORM(S,H) procedure, given a set of variables H and a sequence S, admissible w.r.t. H, returns a sequence S_Nor which is normalized w.r.t. H or *fail*. In the procedure, normalized means normalized w.r.t. H.

begin
Let H be a set of variables, $S = I_1|O_1, \dots, I_n|O_n$ a sequence admissible w.r.t. H.
$S\_Nor_i = I\_Nor_1|O\_Nor_1, \dots, I\_Nor_{n(i)}|O\_Nor_{n(i)}$ is the sequence resulting from the normalization of $I_1|O_1 \dots I_i|O_i$.
$S\_Rem_i = I\_Rem_i|O\_Rem_i, \dots, I\_Rem_n|O\_Rem_n$ is the sequence to be normalized after the normalization of $I_1|O_1, \dots, I_{i-1}|O_{i-1}$. We define $S\_Rem_1 = I_1|O_1, \dots, I_n|O_n$, $S\_Nor_1 = [\,]$.
For i = 1 ... n perform the following steps (in the specified order):

a) Let $S\_Rem_i = I\_Rem_i|\, O\_Rem_i, \dots, I\_Rem_n|\, O\_Rem_n$.
If $I\_Rem_i = \{s_{1,i} \le t_{1,i}, \dots, s_{m(i),i} \le t_{m(i),i}\} \neq \emptyset$ let $t = (t_{1,i}, \dots, t_{m(i),i})$, $s = (s_{1,i}, \dots, s_{m(i),i})$, $T = Var(t)$ and $S = Var(s)$. Then:

1) (Normalizing I_Rem_i) If t and s are not unifiable then STOP and NORM(S,H) = *fail*, otherwise let θ be a most general impartial unifier θ for $\{S\_Nor_i, S\_Rem_i\}$.
Let $\theta_{[s,t]} = \{X/u \in \theta \mid X \in Var(T)\}$ be an mgio of s over t, i.e. $s\,\theta_{[s,t]} \supseteq t\,\theta_{[s,t]}$ and, since t and s do not share variables, $s \supseteq t\,\theta_{[s,t]}$. Let
$I_i\_1 = \{Y \le m \mid Y \in Var(s)$ and Y corresponds in $t\,\theta_{[s,t]}$ to the term m$\}$,
$I_i\_2 = A \cup B$ where
$A = \{Y_j \le M \mid Y_j \le M \in I_i\_1, Y_j \in H \cup Var(S\_Nor_i)$, M is a variable, it doesn't exist $Y_i \le m \in I_i\_1$ with $Y_i \in H \cup Var(S\_Nor_i)$, $Y_i \neq Y_j$ and the variable M in (= to) the term $m_j\}$ and
$B = \{ Y_j \le M \mid Y_j \le M \in I_i\_1$, M is a variable, $Y_j \notin H \cup Var(S\_Nor_i)\}$. Then we define
$\mathbf{I_i\_Nor} = I_i\_1 \setminus I_i\_2$.

2) (Cases of failure). The procedure stops with NORM(S,H) = *fail*, if one of the following cases hold:
i) there exists $X \le m \in I_i\_1$, such that m is not a variable and $X \notin H \cup Var(S\_Nor_i)$.
ii) There exist $X \le Y$, $Z \le n \in I_i\_1$, such that $X \notin H \cup Var(S\_Nor_i)$, $Z \in H \cup Var(S\_Nor_i)$ and Y appear in n.
iii) There exist $X \le Y$, $Z \le Y \in I_i\_1$, such that $X, Y \notin H \cup Var(S\_Nor_i)$.
Otherwise I_i_Nor is the set of (one-way) unification atoms in strong normal form resulting from the normalization of I_Rem_i. The elements of I_i_2 can be eliminated as they are not real constraints.

3) (Passing the input bindings to the rest of the sequence). Now we have to pass to O_Rem_i and to $I\_Rem_{i+1}|O\_Rem_{i+1}, \dots, I\_Rem_n|O\_Rem_n$ the bindings defined by $\theta_{[s,t]}$ obtained in the normalization of I_i_1. We then define:
$O\_Rem_i\_1 = O\_Rem_i\,\theta_{[s,t]}$ and
$S\_Rem_{i+1}\_1 = (I\_Rem_{i+1}|\, O\_Rem_{i+1}, \dots, I\_Rem_n|\, O\_Rem_n)\theta_{[s,t]}$
Moreover, to obtain a normalizated sequence, we have to pass to O_Rem and to $I\_Rem_{i+1}|\, O\_Rem_{i+1}, \dots, I\_Rem_n|\, O\_Rem_n$ also the bindings defined by I_i_Rem. We then define:
$O\_Rem_i\_2 = O\_Rem_i\ \alpha$ and
$S\_Rem_{i+1}\_2 = (I\_Rem_{i+1}|\, O\_Rem_{i+1}, \dots, I\_Rem_n|\, O\_Rem_n)\ \alpha$ where
$\alpha = \{ X/t \mid X \le t \in I_i\_Nor \cup B\} \cup \{M/Y \mid Y \le M \in A\}$

b) (Normalizing O_Remi). Now we have to normalize O_Rem_i as previously modified at step a-3. Since

O_Rem$_i$_2 is a set of (normal) unification atoms, the normalization of O_Rem$_i$_2 is similar to that one considered in [L 88] for the output unifications atoms of an FGHC clause. Therefore perform the following three steps (in the specified order):

1) If there exist atoms of the form $t_1 = t_2$ in O_Rem$_i$_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. If such a unification fails then STOP, NORM(S,H) = *fail*..

2) If O_Rem$_i$_2 contains two atoms $X = t_1$ and $X = t_2$, $X=t_2$ is replaced, if possible, by the set of unification atoms resulting from the unification of t_1 and t_2. As before if such an unification fails then STOP, NORM(S,H) = *fail*.

3) Now we can substitute O_Rem$_i$_2, if possible, with a set O$_i$_Nor of atoms which defines an equivalent idempotent substitution as follows. Such a set is obtained by the following simple algorithm.
Let be A_1 = O_Rem$_i$_2 = $\{X_1 = m_1, \dots, X_n = m_n\}$.

i) For $i = 1, \dots, n$:
If $X_i = m \in A_i$ and m is a term which contains the variable X, then STOP, NORM(S,H)=*fail*.
Otherwise $A_{i+1} = A_i(\{X_i/m\})$.

ii) If we can obtain A_{n+1} from the previous step, then we define
O$_i$_Nor $= A_{n+1} / (C \cup D)$ where
$C = \{X{=}t \in A_{n+1} \mid X \notin H \cup$ Var(S_Nor$_i$) $\cup$ Var(I$_i$_Norm)$\}$ and
$D = \{X{=}Y \in A_{n+1} \mid X \in H \cup$ Var(S_Nor$_i$) $\cup$ Var(I$_i$_Norm)$\}$ Y is a variable and $Y \notin H \cup$ Var(S_Nor$_i$) $\cup$ Var(I$_i$_Norm)$\}$.

c) *(Passing the output bindings to the rest of the sequence).* Now we have to apply the substitutions defined by A_{n+1} to S_Rem$_{i+1}$_2 to obtain S_Rem$_{i+1}$ (the remaining sequence to be normalized). We then define S_Rem$_{i+1}$ = (S_Rem $_{i+1}$ _2) γ, where
$\gamma = \{ X/t \mid (X{=}t) \in$ O$_i$_Nor$\} \cup \{ X/t \mid (X{=}t) \in C\} \cup \{ Y/X \mid X{=}Y \in D\}$

d) *(Defining the normalized sequence).* Finally we define the sequence **S_Nor$_i$** resulting from the normalization of $I_1 \mid O_1 \dots I_i \mid O_i$. Recall that the already normalized sequence is S_Nor$_{i-1}$ = I_Nor$_1$|O_Nor$_1$, ... , I_Nor$_{n(i)}$|O_Nor$_{n(i)}$. S_Nor$_i$ is defined as follows:

i) if I$_i$_Nor = O$_i$_Nor = $\varnothing$ *(true)* then **S_Nor$_i$** = S_Nor$_{i-1}$.

ii) If I$_i$_Nor = *true* then **S_Nor$_i$** = I_Nor$_1$|O_Nor$_1$, ... ,I_Nor$_{n(i-1)}$ |(O_Nor$_{n(i-1)}$)δ $\cup$ O$_i$_Nor_A where:
O$_i$_Nor_A = $\{X = t \mid (X = t) \in$ O$_i$_Nor and $X \notin$ Var(O_Nor$_{n(i-1)}$) and $\delta = \{X/t \mid X{=}t \in$ O$_i$_Nor \ O$_i$_Nor_A$\}$

iii) If O_Nor$_{n(i-1)}$ = *true* then **S_Nor$_i$** = I_Nor$_1$|O_Nor$_1$, ... ,I_Nor$_{n(i-1)}$ ξ $\cup$ I$_i$_Nor_A where:
I$_i$_Nor_A = $\{X = t \mid (X = t) \in$ I$_i$_Nor and $X \notin$ Var(I_Nor$_{n(i-1)}$) and $\xi = \{X/t \mid X{=}t \in$ I$_i$_Nor \ I$_i$_Nor_A$\}$
If $i = n$ the procedure stops and NORM(S,H) = S_Nor$_n$.

End. ♦

A.3 Notes about the procedure.

i) Obviously, since $s \sqsupseteq t\theta_{[s,t]}$, s is different from $t\theta_{[s,t]}$ only because if there exists a subterm $s_1 = f(\dots,X,\dots)$ of s, there is a corresponding (as defined in definition 10) subterm $t_1 = f(\dots,t_h,\dots)$ in $t\theta_{[s,t]}$ where t_h is a term. So the set I$_i$_1 at step a-1 is well defined.

ii) The elements of the set I$_i$-2 defined at step a-1, can be deleted since they do not define constraints. Moreover notice that, because of the definitions of I$_i$-2, we do not further reduce input guards similar to that in the following clause:

c1) *p(X,Y) :- }X≤ f(Z), Y≤ Z} | true.*
In fact in this clause we cannot reduce the input guard to the single atom X≤f(Y), since the resulting clause p(X,Y) :- X≤f(Y) is not a legal NGHC clause.

iii) Cases of failure considered at step a-2 are those of this kind:

c2) *p(X) :- {Y≤ f(...)} | ...*
c3) *p(X) :- {Z≤ K, M≤ K} | ...*
c4) *p(X) :- {W≤ Y, X≤ f(Y)} | ...*

Clearly clauses c2, c3 and c4 are always indefinitely suspended, since Y, Z, M and W cannot be instantiated. Hence these clauses can be desregarded when considering the success set only.

iv) The substitution α defined at step a-3, contains the bindings $\{M/Y \mid Y \le M \in A\}$ because if we get rid of the atom $Y \le M$, we lose the variable M from the sequence, and this visibility is important in defining steps a-1 and a-2 of the procedure.

v) The atoms in the sets C and D defined at step b-3-ii are eliminated since they define bindings for variables which are not in the previous part of the sequence. Of course (at step c) the bindings defined in C and D are passed to the sequence still to be normalized.

The following lemma guarantees that after performing a step of the previous normalization procedure, we can apply the procedure to the subsequence still to be normalized (i.e. the subsequence is still an admissible sequence).

Lemma a1. Let H be a set of variables and $S = I_1|O_1, \dots, I_n|O_n$ a sequence admissible w.r.t. H. Let $s\_Rem_2 = I\_Rem_2\ O\_Rem_2, \dots, I\_Rem_n|O\_Rem_n$ be the sequence defined as in the previous procedure, i.e. the sequence still to be normalized after the application of one step of the procedure to Sin. Then s_Rem_2 is a sequence admissinble w.r.t. H. ♦

It is strightforward to show that the sequence S_Nor obtained with the procedure is a normalized sequence. In fact can be easily proved the following lemma.

Lemma a2. Let H be a set of variables and let S be an admissible sequence w.r.t. H. Then if NORM(S,H) = S_Nor, S_Nor is a sequence which is normalized w.r.t. H. ♦