A Synchronization Logic: Axiomatics and Formal Semantics of Generalized Horn Clauses

M. FALASCHI, G. LEVI, AND C. PALAMIDESSI

Dipartimento di Informatica, Università di Pisa, Pisa, Italy

An extension of Horn clause logic is defined based on the introduction of a synchronization operator. Generalized Horn clauses (GHC) are introduced through an informal description of their operational semantics, which allows discussion of some typical synchronization problems. GHC are first considered formally as a programming language by defining the syntax, the operational semantics, the model-theoretic semantics, and the fixed-point semantics. The above mentioned semantics are given in the Van Emden-Kowalski style (1976, J. Assoc. Comput. Mach. 23, 733–742) and are proved equivalent. GHC are then characterized as axiomatic theories. A set of axiom schemata concerned with the newly introduced synchronization operator is defined and it is proved that the operational semantics inference rule is both sound and complete. Finally, the relation between GHC and Horn clauses is analyzed, and it is proved that Horn clause logic is strictly included in the generalized Horn clause logic. © 1984 Academic Press, Inc.

1. INTRODUCTION

There is a growing interest in the area of logic programming due to the relevance of new applications, such as knowledge bases and inference systems. On the other hand, recent achievements in hardware technology make the design of high level architectures oriented towards the direct execution of logic programming languages both feasible and appealing. A logic machine language must be adequate to systems programming, i.e., it must provide language constructs concerned with concurrency, processes, and synchronization.

Horn clause logic (Kowalski, 1974; 1979) is currently considered a good candidate kernel, because of its basic features which are:

(i) A program is structured as a set of facts and inference rules, thus providing a uniform framework for the representation of declarative information (knowledge) and of procedures.

(ii) The formal semantics of a program is a set of relations defined on the Herbrand universe and is essentially the model-theoretic semantics of first-order logic.
The operational semantics (which is a refutation procedure) can be implemented by very efficient interpreters, which have the inference power of a theorem prover.

As we will briefly show in the next section, Horn clause programs can be executed by a parallel interpreter. The resulting concurrency model, however, is not adequate to the description of the low-level synchronization problems, which typically occur in systems programming. Thus several extensions to Horn clause logic concerned with processes and concurrency have been proposed in the last few years (Bellia et al., 1982; Clark and Gregory, 1981; Van Emden et al., 1982, Monteiro and Pereira, 1978; Shapiro, 1983).

In most of the extended languages, however, the improved expressive power is achieved at the expense of relevant features. On one hand, the resulting language does not have a clean formal semantics in terms of Herbrand models. On the other hand, the operational semantics is not given in terms of the inference rule of an axiomatic theory. In other words, a program cannot be viewed as a logic theory and a computation cannot be given a simple interpretation as a proof. Monteiro's distributed logic (DL) (Monteiro, 1981a) is the only extension which preserves most of the basic features of Horn clause logic, while providing a set of new operators to describe synchronization problems. DL is fully characterized from the programming language viewpoint, i.e., both an operational and a fixed-point semantics are given. However, the underlying logic is not explicitly defined, i.e., the new operators are not syntactically characterized by a set of axioms.

In this paper we define an extension of Horn clause logic, based on the introduction of a single synchronization operator, which is similar to one of the operators defined in DL. In Section 2, we will introduce generalized Horn clauses (GHC) through an informal description of their operational semantics, which will allow us to discuss some typical synchronization problems. Sections 3 and 4 formally define GHC as a programming language, by defining the syntax, the operational semantics, the model-theoretic semantics, and the fixed-point semantics. The above mentioned semantics are given in the Van Emde–Kowalski style (Apt & Van Emden, 1982; Van Emden & Kowalski, 1976) and are proved equivalent. Section 5 provides a characterization of GHC as axiomatic theories. We define a set of axiom schemata concerned with the newly introduced synchronization operator and we prove that the operational semantics inference rule is both sound and complete. Finally, in Section 6 we analyze the relation between GHC and Horn clauses, and we prove that Horn clause logic is strictly included in our generalized Horn clause logic.
2. HORN CLAUSES, SYNCHRONIZATION PROBLEMS, AND GENERALIZED HORN CLAUSES

A logic program (Kowalski, 1974) is a set of Horn clauses, i.e., a set of first-order clauses of the form

(i) \( A \leftarrow B_1, \ldots, B_n, \ n \geq 1 \) (rewrite rule),
(ii) \( A \leftarrow \lambda \) (assertion),
(iii) \( \leftarrow B_1, \ldots, B_n, \ n \geq 1 \) (goal statement),
(iv) \( \leftarrow \lambda \) (halt statement),

where \( A, B_1, \ldots, B_n \) are first-order atomic formulas, \( \leftarrow \) is the implication operator \( \leftarrow \), \( , \) is the conjunction, and all the variable symbols occurring in the atomic formulas are (implicitly) universally quantified. \( \lambda \) is the logical value true, while the blank to the left of the \( \lambda \) in (iii), (iv) stands for the logical value false. Hence the goal statement \( \leftarrow B_1, \ldots, B_n \) is the formula \( \text{not}(B_1 \text{ and } \ldots \text{ and } B_n) \) and the halt statement, corresponding to the empty clause, is a contradiction.

For example, the clause

\[ x(s(x), y, z) \leftarrow x(x, y, w), +y, w, z \]

must be read as

\[ \forall x, y, z, w((x(x, y, w) \text{ and } +y, w, z) \rightarrow x(s(x), y, z)). \]

In the procedural interpretation (Van Emden & Kowalski, 1976), rewrite rules and assertions are viewed as procedure declarations, while a goal statement is viewed as a main program consisting of a set of procedure calls. The computation of a goal statement is a sequence of goal statements, each obtained from the predecessor by expanding a procedure call.

A procedure call can be expanded if it "matches" the left part of a procedure declaration. The matching process will generally bind variable symbols occurring both in the procedure declaration (parameter passing) and in the procedure call (parameter return). If the matching process is successful, the procedure call is replaced in the goal statement by the procedure body (right part of the procedure declaration), and all the variable bindings are performed.

Note that, for a given goal, there are possibly several computations (i.e., the computation is nondeterministic), since a specific procedure call could match different procedure declarations. A computation is successful if the last goal in the sequence is the halt statement.

A successful computation is a proof by contradiction. In fact it shows that
the theory defined by the procedure declarations and the goal statement is unsatisfiable, since it allows us to derive the halt statement.

Parallelism can be achieved by letting several procedure calls in a goal statement to be expanded concurrently. A goal statement can be viewed as a set of parallel process activations and the conjunction operator can be interpreted as parallel composition of processes.

The concurrent (possibly nondeterministic) computations originated from a goal statement are not independent. In fact, process activations can share variable symbols, which must have compatible bindings in all the concurrent computations. Shared variables provide the basic communication mechanism among processes. However, this mechanism is not adequate to synchronize concurrent computations.

A synchronization is a mechanism which forces a computation to wait until a condition on the state of a different computation is verified. Some synchronization mechanism is needed to naturally describe systems consisting of processes and shared resources as well as systems of message passing processes. In the first case, a synchronization is needed between a process and a resource, to guarantee that the resource state is consistent. In the second case, the receiving process must wait for a message from the sending process.

Note that synchronization requires a process to be able to test the existence of another process (the resource or the sending process) and to verify some condition on its state.

A process activation in Horn clause logic can only access the state of another process by variable sharing. Hence, if two processes need a synchronization, they must share a variable for each state component which affects the synchronization condition. Assume we have two process activations \( A(x) \) and \( B(x) \) belonging to computations originated from a single goal statement \( \neg P(x) \), and let \( x \) be the shared variable that could allow the synchronization.

We describe the synchronization condition as follows. \( A \) can be rewritten only if the “state variable” \( x \) in process \( B \) is bound to 0, or can be somehow forced to be bound to 0. This can be expressed by a clause of the form

\[
A(0) \leftarrow \ldots.
\]

The above clause, however, does not state that the rewriting occurs only if \( x \) is bound to 0, within process \( B \), which, in addition, must be present in some computation.

In order to fully define our synchronization, we need some sort of context-dependent rewrite rule of the form

\[
A, B(0) \leftarrow \ldots,
\]
which means that if there exist a process activation $A$ and a process activation $B$ in a state unifiable with $0$, then we can perform a rewriting. Note that $A$ is not required to share the state variable $x$, which, in the example, is assumed to be a private state variable of $B$.

Hence synchronization seems to require a generalization of Horn clauses, in which procedure declarations have the form

$$A_1, \ldots, A_n \leftarrow B_1, \ldots, B_m$$

or

$$A_1, \ldots, A_n \leftarrow \lambda \quad m, n \geq 1,$$

i.e., they are allowed to have more than one atomic formula in the left part.

From the operational viewpoint, this implies that several procedure calls in a goal statement are expanded concurrently by a single matching clause. In our concurrent computation framework, such an operation can be interpreted as a process join operator, which transforms a set of processes into a single process.

We are left with the problem of defining the meaning of the new clauses from the logical viewpoint. The operator "\" occurring in the left parts of the clauses looks like a conjunction operator. We will show that this is not the case and that such an operator cannot be defined in terms of standard logical operators.

The axiomatization of the logic resulting from the introduction of the new operator and the definition of the semantics of our logic as a programming language are the main contributions of this paper. The operator, as a matter of fact, is not quite new and was first defined in the framework of generalized AND-OR graphs (Degano, 1979; Levi & Sirovich, 1975; 1976). However, at that time, it was characterized only from the operational viewpoint. The same operator is used in Degano and Diomedi (1983) and, together with a sequencing operator, in distributed logic (Monteiro, 1981a; 1981b; 1981c), which is given a semantics as a programming language but still lacks a complete axiomatization.

Before getting into the technical aspects of our logic, let us go back to some pragmatic considerations on the expressive power of Horn clauses, in the framework of synchronization problems. We could argue that Horn clauses allow parallel communicating computations (which are semantically equivalent to sequential computations) but do not allow to control concurrency. This problem has been solved by several authors by adding to Horn clauses a control language (e.g., through annotations) (Bellia et al., 1982; Clark & Gregory, 1981; Van Emden et al., 1982; Monteiro & Pereira, 1978; Shapiro, 1983). This solution is satisfactory for some control aspects, where it does only affect the computation without modifying the formal
semantics. However, in the case of synchronization annotations, we obtain programs whose semantics is not equivalent to the pure Horn clause programs. On the other hand, the annotations do not have a semantics related to the Herbrand semantics of the pure Horn clauses.

We will now show how a classical process-resource problem can be defined in generalized Horn clauses.

The Dining Philosophers (Simplified Version)

There are \( n \) philosophers (whose names are 1, 2,..., \( n \)) sitting at a table, where there are \( n \) forks. Each philosopher has three states:

State 0. The philosopher has no forks, hence he can only think.
State 1. The philosopher has one fork.
State 2. The philosopher has two forks and is allowed to eat.

Forks can be picked up one at a time, and cannot be returned to the table until the philosopher has eaten for a while.

The Dining Philosophers problem is a typical example of resource sharing (the forks), where deadlock is possible (when each philosopher has only a fork). We are not concerned here with the deadlock-free solution, but only with the representation of the problem.

The generalized Horn clauses description is based on two predicates \( PHIL \) and \( FORK \). \( PHIL \) is a diadic predicate, whose arguments are the philosopher name and the philosopher state. \( FORK \) is a monadic predicate whose argument is the state of the resource, i.e., the number of available forks.

The state transitions are defined by the generalized clauses

\[
\begin{align*}
PHIL(x, 0), FORK(n + 2) & \leftarrow PHIL(x, 1), FORK(n + 1) \\
PHIL(x, 1), FORK(n + 1) & \leftarrow PHIL(x, 2), FORK(n) \\
PHIL(x, 2), FORK(n) & \leftarrow PHIL(x, 0), FORK(n + 2)
\end{align*}
\]

The initial state is represented by a goal statement. For instance, the 3 philosophers problem can be stated as

\[
\leftarrow PHIL(1, 0), PHIL(2, 0), PHIL(3, 0), FORK(3).
\]

All the states of the system can be obtained by computing the goal statement. Note that each state transition is a synchronization involving a philosopher (process) and the set of forks (resource).

The same problem can also be described by Horn clauses. However, one can easily convince himself that the solution will require a unique state predicate. In fact, the resource must interact with all the philosophers. Hence
it must have a shared variable for each philosopher state. The state of the whole system can then be represented by the resource predicate, and the state transitions can be defined in terms of this predicate only. With such a solution, the individual processes (philosophers) and the resource are lost and only the global state is considered. Of course, it is not easy to define the state transitions for the general case, when the number of philosophers is not established. The general Horn clauses solution is, on the contrary, simple, general, and natural.

Let us finally show one more interesting feature of generalized Horn clauses. When a synchronization takes place, it is possible to have a communication between processes. This is achieved by letting two different atomic formulas in the left parts of a generalized clause to share a variable symbol. For example, consider the following standard Horn clauses:

\[
A \leftarrow \text{SEND-TO-B}(v), C
\]
\[
B \leftarrow \text{RECEIVE-FROM-A}(y), D(y).
\]

Assume \( A \) and \( B \) occur in the current goal statement and we want SEND-TO-B and RECEIVE-FROM-A to define a synchronous communication from \( A \) to \( B \). This can be achieved by the generalized assertion

\[
\text{SEND-TO-B}(x), \text{RECEIVE-FROM-A}(x) \leftarrow \lambda
\]

which expands concurrently the send and receive processes causing the value \( v \) computed by \( A \) to be passed to \( B \).

3. Generalized Horn Clauses (GHC)

3.1. Syntax

A GHC Program is a finite nonempty set of generalized clauses

\[
W = \{C_1, \ldots, C_n\},
\]

where every \( C_i \) is an expression of the form

\[
A_1 + \cdots + A_h \leftarrow B_1 + \cdots + B_k \quad h > 0 \quad k \geq 0,
\]

where the \( A_i \)'s and \( B_i \)'s are atomic formulas. An atomic formula is an expression of the form

\[
P(t_1, \ldots, t_n),
\]
where $P$ is a predicate symbol and the $t_i$'s are terms. A term is either a variable or an expression of the form

$$f(t_1, \ldots, t_m),$$

where $f$ is a function symbol and the $t_i$'s are terms. Constants are 0-ary function symbols. When $k = 0$, the clause denoted by

$$A_1 + \cdots + A_h \leftarrow 0 \lambda$$

is called a unit clause. A goal statement in GHC is an expression of the form

$$\leftarrow A_1 + \cdots + A_k \quad k \geq 0,$$

which stands for not($A_1 + \cdots + A_k$), where the $A_i$'s are atomic formulas. If $k = 0$ the goal statement denoted by

$$\leftarrow 0 \lambda$$

is called null clause. In order to simplify our constructions we introduce the notion of a generalized formula:

(i) $\lambda$ is a generalized formula.

(ii) Any atomic formula $P(t_1, \ldots, t_n)$ is a generalized formula.

(iii) If $r$ and $s$ are generalized formulas, then $r + s$ is a generalized formula.

In the following, generalized clauses and goal statements will be, respectively, denoted as $r \leftarrow s$ and $\leftarrow s$, where $r$ and $s$ are generalized formulas and $r \neq \lambda$. We assume that the $+$ operator is commutative and associative. Moreover we assume that $\lambda$ is the neutral element of $+$.

3.2. Derivation Rule

Let $W$ be a program and let $\leftarrow s$ be a goal statement. The goal statement $\leftarrow t$ is directly derivable from $\leftarrow s$ iff:

(i) $s = s_1 + s_2 \ (s \neq \lambda)$.

(ii) There exists in $W$ a clause $r_1 \leftarrow r_2$ such that $r_1$ and $s_1$ are unifiable.

(iii) If $\theta$ is the MGU (most general unifier) of $r_1$ and $s_1$, then $t = \theta(r_2 + s_2)$.
The notation \( s \vdash_{W}^{\theta} t \) will denote that \( \vdash_{W}^{t} \) is directly derivable from \( \vdash_{W}^{s} \) (and that \( \theta \) is the MGU used in the derivation). If for some \( n \geq 0 \),

\[
S = s_1 \vdash_{W}^{\theta_1} s_2 \vdash_{W}^{\theta_2} \cdots \vdash_{W}^{\theta_{n-1}} s_n = t
\]

and

\[
\theta = \theta_1 \bullet \theta_2 \bullet \cdots \bullet \theta_{n-1}
\]

the relation: \( s \vdash_{W}^{\theta} t \) (\( \vdash_{W}^{t} \) is derivable from \( \vdash_{W}^{s} \)) holds.

The computation of a goal statement \( \vdash_{W}^{s} \) in a program \( W \) is the sequence of derivations which are obtained starting from \( \vdash_{W}^{s} \) and applying the clauses of \( W \). This process is nondeterministic.

**Definition 1.** The goal statement \( \vdash_{W}^{s} \) has a refutation (in \( W \)) iff there exists a \( \theta \) such that

\[
S \vdash_{W}^{\theta} \lambda.
\]

If this is the case the computation of \( \vdash_{W}^{s} \) in \( W \) terminates successfully and \( s \) is a theorem of \( W \).

Note that our notion of computation is essentially similar to the standard HCL one, namely a formula \( s \) is a theorem if there exists a refutation for its negation, i.e., for the goal statement \( \vdash_{W}^{s} \).

We will now give a simple example of direct derivation. Assume \( W \) contains the generalized clause

\[
P(s(x_1, s(x_2)) + Q(s(x_1), x_3) \vdash_{W}^{+} R(x_1, x_3) + S(x_2, x_3)
\]

and let \( S \) be the goal statement

\[
\vdash_{W}^{-}P(s(x), y) + Q(s(x), z) + R(x, y).
\]

The goal statement \( S \) can be decomposed in

\[
S_1 = P(s(x), y) + Q(s(x), z) \quad \text{and} \quad S_2 = R(x, y).
\]

The most general unifier of \( S_1 \) and the clause left part is the substitution

\[
\{ x = x_1, y = s(x_2), z = x_3 \}.
\]
The new goal statement is then

\[ \leftarrow^+ R(x_1, x_3) + S(x_2, x_3) + R(x_1, s(x_2)). \]

4. The Semantics of Generalized Horn Clauses

4.1. Operational Semantics

The operational semantics is defined according to the above described derivation rule.

Let \( x_1, \ldots, x_n \) be the variables occurring in a generalized formula \( s \). The semantics of \( s \) with respect to a program \( W \) is

\[ D_0(s, W) = \{ v(\theta(x_1, \ldots, x_n)) \mid v \text{ is a valuation}^1 \text{ and } s \vdash^W_{\theta} \} \]

In the following we will define a fixpoint and a model-theoretic semantics which are consistent with the operational definition.

4.2. The Model-Theoretic Semantics

There are some problems in the interpretation of the new symbol "+." It can be considered as a function that transforms two formulas \( r \) ans \( s \) into a new formula, \( r + s \) as the standard logic connectives (or, and, etc...) do.

Nevertheless, the symbol "+" is essentially different from classical logic connectives, because the "truth" of a ground composite formula \( r + s \) does not depend on the truth of \( r \) and \( s \) (whereas the values of composite expressions like \( r \text{ and } s \), \( r \text{ or } s \) depend on the values of components only).

The following property holds:

if \( r \) and \( s \) are theorems, then also \( r + s \) is a theorem.

In fact, if there is a refutation for both \( \leftarrow^+r \) and \( \leftarrow^+s \) (i.e., \( r \) and \( s \) are theorems), then also \( \leftarrow^+r + s \) has a refutation (i.e., also \( r + s \) is a theorem).

The reverse in general does not hold. For example, in the program \( W = \{ A + B \leftarrow^+ \lambda \} \), \( \leftarrow^+A + B \) has a refutation, while neither \( \leftarrow^+A \) nor \( \leftarrow^+B \) have a refutation. Therefore, in most cases, \( A + B \) (and, in general, any composite expressions) must be considered as a new predicate (when \( A + B \) is true and either \( A \) or \( B \) is false).

We must then introduce a new definition of Herbrand interpretation: not

---

1 A valuation (ground instance) of a set \( \Sigma \) of expressions (terms or generalized formulas) is any set obtained by applying to \( \Sigma \) a substitution of variables with ground terms (i.e., terms containing no variables).
only atomic ground formulas, but also generalized ground formulas need to belong to it. Moreover, our interpretations must be as simple as possible, i.e., they must not contain any redundant information. Let us explain our notion of redundancy through an example:

Let $h$ be a function defined by composition as $h = f \circ g$ and assume the following properties hold:

1. $h(n_1) = m_1$,
2. $f(n_1) = k$,
3. $g(k) = m_1$.

The information provided by (1) is redundant, because it can be derived from (2) and (3), which, on the other hand, give more information than (1).

Accordingly the truth of $r$ and $s$ gives us more "information" than the truth of $r + s$. Therefore, if $r, s \in I$ (where $I$ is a Herbrand interpretation) $r + s$ does not need to belong to $I$; its truth can be expressed as a consequence of the truth of $r$ and $s$. Let $W$ be a program.

**Definition 2.** The *Herbrand universe* $U$ of $W$ is the set of all the ground terms.

**Definition 3.** The *Herbrand base* $B$ of $W$ is the set of all the ground generalized formulas which are obtained as composition by "+," from $\lambda$ and the atomic formulas generated by the predicate symbols occurring in $W$ applied to the elements of $U$.

**Definition 4.** A *Herbrand interpretation* $I$ of $W$ is any subset of the Herbrand base such that $\lambda$ is in $I$ and if $s_1, \ldots, s_n$ are in $I$ then $s_1 + \cdots + s_n$ is not in $I$.

Let $I$ be a Herbrand interpretation of $W$.

**Definition 5.** A ground formula $s$ is true in $I$ iff:

$$\exists s_1, \ldots, s_n \in I \text{ such that } s = s_1 + \cdots + s_n \quad (n \geq 1).$$

The set of all the ground formulas which are true in $I$ is given by the function $\sigma$:

$$\sigma(I) = \{ s_1 + \cdots + s_n \mid s_1, \ldots, s_n \in I, n \geq 1 \}$$

($\sigma$ is a monadic function from Herbrand interpretations into subsets of the Herbrand base).

It is worth noting that $\lambda$ is true in all the Herbrand interpretations. The
function \( \rho \) from subsets of the Herbrand base to Herbrand interpretations is defined as

**Definition 6.** Let \( H \) be a subset of the Herbrand base

\[
\rho(H) = H - \{ s_1 + \cdots + s_n \mid s_1, \ldots, s_n \in H, n \geq 2 \}.
\]

**Proposition 1.** If \( I \) is a Herbrand interpretation, then

\[
\rho(\sigma(I)) = I.
\]

The converse, in general, does not hold, since \( H \subseteq \sigma(\rho(H)) \).

We will now define the relation \( \vdash_{C_i} \), established by a clause \( C_i \in W \) on the elements of \( B \).

**Definition 7.** (i) If \( r \vdash_{C_i} s \) is a ground instance of \( C_i \), then \( r \vdash_{C_i} s \).

(ii) If \( r \vdash_{C_i} s \) holds, then for any \( t \in B \), \( r + t \vdash_{C_i} s + t \).

Note that the relation \( r \vdash_{C_i} s \) (where \( r, s \in B \)) holds iff \( r \vdash \) is directly reducible to \( s \), using the clause \( C_i \).

Definition 7(ii) shows that the operator \( \vdash_{C_i} \) establishes a relation not only on the left (\( r \)) and the right part (\( s \)), but also on every pair of formulas (\( r + t, s + t \)) which can be obtained from \( r \) and \( s \) by adding another formula \( t \). It is exactly this property that makes \( \vdash_{C_i} \) different from the logical implication. A clause \( C_i \) is true in a Herbrand interpretation \( I \) iff:

\[
\forall r, s \in B, \quad \text{if} \quad r \vdash_{C_i} s \quad \text{and} \quad s \in \sigma(I) \quad \text{then} \quad r \in \sigma(I).
\]

The relation \( \vdash_{W} \) is defined as the union of the relations \( \vdash_{C_i} \) for each \( C_i \) in \( W \).

**Definition 8.** Let \( I \) be a Herbrand interpretation of \( W \). Then, \( I \) is a Herbrand model of \( W \) iff all the clauses of \( W \) are true in \( I \).

We would like to have a notion of least model, that is a model in which only those formulas which are true in every model are true, as is the case for the least Herbrand model for Horn clauses. Therefore, we must introduce a partial ordering relation on interpretations. The set inclusion relation defined for Horn clauses (Van Emdeh & Kowalski, 1976) is not adequate to our aims. Consider, for example, the program

\[
W = \{ P(a) + Q(a) \vdash_{=} \lambda \}.
\]
The interpretation $I_1 = \{P(a) + Q(a), \lambda\}$ is a model which contains all the formulas which are true in all the models. However, the interpretation $I_2 = \{P(a), Q(a), \lambda\}$ is also a model of $W$, and $I_1 \not\subseteq I_2$ does not hold. Our relation is defined

**Definition 9.** If $I, J$ are Herbrand interpretations, then

$I \leqslant J$ \iff $I \subseteq \sigma(J)$.

It is straightforward to prove (by definition of $\sigma$) that $\leqslant$ is a partial ordering relation on the set of Herbrand interpretations of $W$, and that $I \leqslant J$ holds iff the formulas that are true in $I$ are true also in $J$. Note that, in the above example, $I_1 \leqslant I_2$ holds. We will now characterize the structure of the class of Herbrand interpretations with respect to the partial ordering relation $\leqslant$.

**Definition 10.** Let $L$ be a (possibly not finite) set of Herbrand interpretations:

$$
\bigcap L = \rho \left( \bigcap_{I \in L} \sigma(I) \right)
$$

$$
\bigcup L = \rho \left( \bigcup_{I \in L} \sigma(I) \right) = \rho \left( \bigcup_{I \in L} I \right),
$$

where the symbols $\cap$ and $\cup$ denote the well-known operations of intersection and union on sets, respectively.

**Theorem 1.** Let $L$ be a set of Herbrand interpretations:

1. $\cap L$ and $\cup L$ are Herbrand interpretations.
2. $\cap L = \text{glb}(L)$ and $\cup L = \text{lub}(L)$.
3. If $L$ is a set of models, then also $\cap L$ is a model (and therefore $\cap L = \text{glb}(L)$ holds in the class of models, too).

**Proof.**

1. By definition, since both $\cap L$ and $\cup L$ are obtained by applying the operator $\rho$ to sets containing $\lambda$.
2. We prove that $\cap L = \text{glb}(L)$ (a similar proof applies to the second part of the theorem)

   - (i) $\forall I \in L \cap L \leqslant I$. In fact, if $s \in \cap L$, then $s \in \bigcap_{I \in L} \sigma(I)$ (being $\cap L = \rho(\bigcap_{I \in L} \sigma(I)) \subseteq \bigcap_{I \in L} \sigma(I)$), and therefore $s \in \sigma(I)$.

   - (ii) If $H$ is a Herbrand interpretation such that $\forall I \in L$, $H \leqslant I$, then $H \leqslant \cap L$. In fact, if $s \in H$, then $\forall I \in L$, $s \in \sigma(I)$ (being $H \leqslant I$). Therefore $s \in \bigcap_{I \in L} \sigma(I)$, hence (since $\bigcap_{I \in L} \sigma(I) \subseteq \sigma(\cap L)$), $s \in \sigma(\cap L)$. 

(1.3) Let $L$ be a class of models of $W$. Let $r, s \in B$ be formulas such that $r \vdash_w s$, and $s \in \sigma(\bigcap L)$. Then, for every $J \in L$, $s \in \sigma(J)$, and therefore $r \in \sigma(J)$, $J$ being a model. Hence $r \in \sigma(\bigcap L)$. This proves that $\bigcap L$ is a model.

**Theorem 2.** (2.1) The class of Herbrand interpretations $I$ of $W$, partially ordered by $\leq$, is a complete lattice. Its bottom element is $\{\lambda\}$ and its top element $p(B)$.

(2.2) The class of Herbrand models $M$ of $W$, partially ordered by $\leq$, is a complete lattice (and is a sublattice of $(I, \leq)$). Its top element is $p(B)$.

*Proof.* (2.1) It comes directly from Theorem 1.2.

(2.2) Theorem 1.3 ensures that there exists $\text{glb}(L)$ for every subset $L$ of the class of Herbrand Models.

We need to prove that $\text{lub}(L)$ exists too. Let $L$ be a set of models, and let us consider the set:

$$L' = \{H \mid H \text{ is a model and } \forall I \in L, I \leq H\}.$$  

We prove now that $\bigcap L' = \text{lub}(L)$:

(i) $\forall I \in L, I \leq \bigcap L'$ holds, being $\bigcap L' = \text{glb}(L')$ and $\forall J \in L', I \leq J$.

(ii) If $H$ is a model such that $\forall I \in L, I \leq H$, then $\bigcap L' \leq H$. This is also straightforward, being $H \in L'$.

Theorem 2 allows us to define the model-theoretic semantics.

**Definition 11.** The model-theoretic semantics of a generalized formula $s$ in $W$ is defined

$$D_M(s, W) = \left\{\left.(t_1, \ldots, t_n) \mid s[x_i/t_i, i = 1, \ldots, n] \in \sigma \left(\bigcap M\right)\right\},$$

where the $x_i$'s are the variables occurring in $s$ ($i = 1, \ldots, n$), $M$ is the class of Herbrand models of $W$ and $s[x_i/t_i, i = 1, \ldots, n]$ denotes the ground generalized formula obtained from $s$, by replacing the occurrences of the $x_i$'s with the $t_i$'s, $i = 1, \ldots, n$.

4.3. *Fixpoint Semantics*

In the fixpoint semantics the denotation of a recursively defined procedure is defined to be the least fixpoint of a transformation associated with the procedure definition.
It is possible to give a similar definition of fixpoint semantics for GHC programs, which can be seen as sets of mutually recursive procedure declarations.

Let \( W \) be a GHC program (i.e., a set of generalized clauses). Let us consider the transformation \( T_W \) associated with \( W \) which maps Herbrand interpretations to Herbrand interpretations and which is defined

\[
T_W(I) = \rho (\{ r \mid \exists s \in \sigma(I) \text{ such that } r \xleftarrow{W} s \} \cup \{ \lambda \}).
\]

Note that \( T_W(I) \) is a Herbrand interpretation, since it is obtained by applying \( \rho \) to a set containing \( \lambda \).

The following theorems give some relevant properties of the transformation \( T_W \).

**Theorem 3.** \( T_W \) is monotonic and continuous.

**Proof.** (1) \( T_W \) is monotonic, i.e., \( \forall I, J, I \leq J \Rightarrow T_W(I) \leq T_W(J) \). This comes immediately from

\[
I \leq J \Rightarrow \{ r \mid \exists s \in \sigma(I), r \xleftarrow{W} s \} \\
\subseteq \{ r \mid \exists s \in \sigma(J), r \xleftarrow{W} s \}.
\]

(2) \( T_W \) is continuous, i.e., for every chain \( K \),

\[
lub(T_W(K)) = T_W(lub(K))
\]

(i.e., \( \cup T_W(K) = T_W(\cup K) \)). Note that the domain of \( T_W \) is a complete lattice, hence, for every chain \( K \), there exists \( lub(K) \).

(i) \( \cup T_W(K) \leq T_W(\cup K) \) (i.e., \( \rho(\bigcup_{I \in K} T_W(I)) \leq T_W(\rho(\bigcup_{I \in K} I)) \)). Since \( T_W \) is monotonic.

(ii) \( T_W(\cup K) \leq \cup T_W(K) \) (i.e., \( T_W(\rho(\bigcup_{I \in K} I)) \leq \rho(\bigcup_{I \in K} T_W(I)) \)). If \( s \in T_W(\rho(\bigcup K)) \), then \( \exists r \in B \) such that \( s \xleftarrow{W} r \), where \( r \in \sigma(\rho(\bigcup K)) \), that is,

\[
r = r_1 + \cdots + r_h
\]

for some \( r_1, \ldots, r_h \in \rho(\bigcup K) \) (i.e., \( r_1, \ldots, r_h \in \bigcup K \)). Since \( K \) is a chain, that is, a sequence of interpretations

\[
I_1, I_2, \ldots, I_m, \ldots
\]
such that

\[ I_1 \preceq I_2 \preceq \cdots \preceq I_m \preceq \cdots, \]

for every \( r_j, j = 1, \ldots, h \), there exists an index \( n_j \) such that

\[ r_j \in I_{n_j}. \]

Then if \( n = \max\{n_1, \ldots, n_h\} \), \( \forall j, r_j \in \sigma(I_n) \) holds, being

\[ I_{n_1} \preceq I_{n_2} \ldots \preceq I_{n_h} \preceq I_n. \]

Hence \( r \in \sigma(I_n) \) and \( s \in \sigma(T_w(I_n)) \). Therefore \( r \in \sigma(\rho(\cup_{I \in K} \sigma(T_w(I)))) = \sigma(\rho(\cup_{I \in K} T_w(I))). \)

**Theorem 4.** The set of fixpoints of \( T_w \),

\[ \{ I \mid T_w(I) = I \}, \]

and the set of interpretations closed under \( T_w \),

\[ \{ I \mid T_w(I) \preceq I \}, \]

have a minimal element. Moreover,

\[ \min\{ I \mid T_w(I) = I \} = \min\{ I \mid T_w(I) \preceq I \}. \]

**Proof.** This property holds for every monotonic function which maps a complete lattice into itself.

**Theorem 5.** \( \min\{ I \mid T_w(I) = I \} = \cup_{k \geq 0} T_w^k(\{ \lambda \}). \)

**Proof.** This theorem is a direct consequence of the fixpoint theorem, since \( T_w \) is a continuous transformation as stated by Theorem 3.

**Theorem 6.** Let \( I \) be a Herbrand interpretation of \( W \). \( I \) is closed under \( T_w \) iff \( I \) is a Herbrand model of \( W \).

**Proof.** (If part) If \( I \) is a Herbrand model of \( W \), then \( T_w(I) \preceq I \). In fact, if \( s \in T_w(I) \), then there exists \( r \in \sigma(I) \) such that

\[ s \leftarrow_{w} r \quad \text{or} \quad s = \lambda. \]

In the first case, \( s \in \sigma(I) \) since \( I \) is a model. In the second case, \( s \in I \) since \( \lambda \in I \) by definition. Therefore, \( s \in \sigma(I) \).

(Only if part) If \( T_w(I) \preceq I \) then \( I \) is a model of \( W \). In fact, \( \forall r \in \sigma(I) \) if
there exists \( s \in B \) such that \( s \xrightarrow{w} r \) then \( s \in \sigma(T_w(I)) \). Since \( T_w(I) \leq I \) implies \( \sigma(T_w(I)) \subseteq \sigma(I) \), \( s \in \sigma(I) \), and then \( I \) is a model.

Theorem 4 states that there exists an interpretation which is the least interpretation mapped into itself by the transformation \( T_w \). Therefore it is possible to define the semantics of a generalized formula, in a set \( W \) of generalized clauses, in analogy to the standard definition of the fixpoint semantics for a recursive program.

**Definition 12.** Let \( W \) be a set of generalized clauses, \( s \) be a generalized formula, and \( x_1, \ldots, x_n \) be the variables occurring in \( s \). The denotation of \( s \), according to the fixpoint semantics, is defined as

\[
\mathcal{D}_F(s, W) = \{ (t_1, \ldots, t_n) | s[x_i/t_i, i = 1, \ldots, n] \in \sigma(\mu F(T_w)) \},
\]

where \( \mu F(T_w) \) denotes the minimal fixpoint of \( T_w \).

### 4.4. Equivalence Theorems

The following theorems prove that all the semantics we have defined (operational, model-theretic, and fixpoint) are equivalent.

**Theorem 7.** For every program \( W \) and every generalized formula \( s \),

\[
\mathcal{D}_M(s, W) = \mathcal{D}_F(s, W)
\]

(equivalence of model-theoretic and fixpoint semantics).

**Proof.** This equivalence is a direct consequence of the equalities \( \{ I | T_w(I) \leq I \} = \{ I | I \text{ is a Herbrand model of } W \} \) (Theorem 6) and \( \min \{ I | T_w(I) \leq I \} = \min \{ I | T_w(I) = I \} \) (Theorem 4).

**Theorem 8.** For every program \( W \) and every generalized formula \( s \),

\[
\mathcal{D}_F(s, W) = \mathcal{D}_O(s, W)
\]

(equivalence of fixpoint and operational semantics).

**Proof.** \( \mathcal{D}_F(s, W) \subseteq \mathcal{D}_O(s, W) \).

8.1.1. Let \( x_1, \ldots, x_n \) be the variables occuring in \( s \). If \( (t_1, \ldots, t_n) \in \mathcal{D}_F(s, W) \) then, by definition,

\[
r = s[x_i/t_i, i = 1, \ldots, n] \in \sigma(\bigcap \{ I | T_w(I) = I \})
\]
and then, by Theorem 5, \( r \in \sigma(\bigcup_{k > 0} T^k_w(\{\lambda\})) \). Hence, there exist \( r_1, \ldots, r_m \in \bigcup_{k > 0} T^k_w(\{\lambda\}) \) such that
\[
    r = r_1 + \cdots + r_m.
\]
Let \( n_j \) be the index such that \( r_j \in T^m_w(\{\lambda\}) \). There exists a sequence of generalized ground formulas
\[
    s^j_1, \ldots, s^j_{n_j}
\]
such that
\[
    r_j = s^j_{n_j} \leftarrow \frac{+}{w} s^j_{n_{j-1}} \leftarrow \frac{+}{w} \cdots \leftarrow \frac{+}{w} \lambda.
\]
By definition, \( s^j_1 \leftarrow \frac{+}{w} s^j_{1-1} \) means that there exist:
- a clause \( s' \leftarrow \frac{+}{w} s'' \) in \( W \),
- a generalized ground formula \( t \),
- a ground substitution \( \theta_i \),
such that
\[
    s^j_1 = \theta_i(s') + t \quad (= \theta_i(s' + t)),
\]
\[
    s^j_{j-1} = \theta_i(s'') + t \quad (= \theta_i(s'' + t)).
\]
Consequently, \( s^j_1 \leftarrow \frac{\theta_i}{w} s^j_{1-1} \) holds and, in general,
\[
    r_j = s^j_{n_j} \leftarrow \frac{\theta_{n_j}}{w} s^j_{n_{j-1}} \leftarrow \frac{\theta_{n_{j-1}}}{w} \cdots \leftarrow \frac{\theta_1}{w} \lambda.
\]
Therefore, if \( \tau_j = \theta_{n_j} \circ \theta_{n_{j-1}} \circ \cdots \circ \theta_1 \), \( r_j \leftarrow \frac{\tau_j}{w} \lambda \), and then
\[
    r = r_1 + r_2 + \cdots + r_m \leftarrow \frac{\tau_1}{w} \lambda + \cdots + \frac{\tau_m}{w} \lambda.
\]
Remark. The \( \theta_i \)'s and \( \tau_i \)'s do not concern the formulas \( r_j \)'s (since they are ground), but only concern the variables of the clauses used in the refutation. In the following, when this will be the case, the symbol \( \theta \) over \( \frac{\tau_i}{w} \lambda \) will be omitted.
8.1.2. If \( r \) is a ground instance of \( s \) and if \( r \overset{\theta}{\overset{\omega}{\rightarrow}} \lambda \), then there exists a substitution \( \theta \) such that \( s \overset{\theta}{\omega}{\rightarrow} \lambda \) and such that \( r \) is an instance of \( \theta(s) \). This property can be proved by induction, using the definition of derivation. Intuitively, this result follows from the properties of \( \theta \) which is the composition of a sequence of MGU's, i.e., the most general substitution necessary to obtain a matching between a subformula of a goal statement and the left part of a clause.

As a consequence of 8.1.1, if \( (t_1, \ldots, t_n) \in D_{\theta}(s, W) \) and \( r = s[x_i/t_i, i = 1, \ldots, n] \), then \( r \overset{\omega}{\rightarrow} \lambda \). Then, as a consequence of 8.1.2, there exist a substitution \( \theta \) and a valuation \( v \) such that

\[
 s \overset{\theta}{\omega}{\rightarrow} \lambda \quad \text{and} \quad r = v(\theta(s)).
\]

Therefore \( v(\theta(x_1, \ldots, x_n)) = (t_1, \ldots, t_n) \) and then

\[
 (t_1, \ldots, t_n) \in D_{\theta}(s, W).
\]

8.2. \( D_{\theta}(s, W) \subseteq D_{\theta}(s, W) \). In fact, if \( (t_1, \ldots, t_n) = v(\theta(x_1, \ldots, x_n)) \in D_{\theta}(s, W) \), then \( s \overset{\theta}{\omega}{\rightarrow} \lambda \), that is,

\[
 s = s_1 \overset{\theta_1}{\omega}{\rightarrow} s_2 \overset{\theta_2}{\omega}{\rightarrow} \cdots \overset{\theta_{k-1}}{\omega}{\rightarrow} s_k \overset{\theta_k}{\omega}{\rightarrow} \lambda,
\]

where \( \theta_1 \bullet \theta_2 \bullet \cdots \bullet \theta_k = \theta \). It is easy to see that \( s_k \overset{\theta_k}{\omega}{\rightarrow} \lambda \) implies that there exists a clause in \( W \) of the form \( r \overset{\omega}{\rightarrow} \lambda \) such that \( \theta_k(s_k) = \theta_k(r) \). Therefore, since \( v \) is a valuation of the variable of \( \theta_k(s_k) \),

\[
 v(\theta_k(s_k)) = v(\theta_k(r)) \overset{\omega}{\Rightarrow} \lambda \text{ holds}.
\]

Let us consider now the derivation expressed by \( s_{k-1} \overset{\theta_k}{\omega}{\rightarrow} s_k \). Let \( r \overset{\omega}{\rightarrow} r' \) be the clause of \( W \), and \( t, t' \) the generalized formulas such that

(a) \( s_{k-1} = t + t' \)

(b) \( \theta_{k-1}(t) = \theta_{k-1}(r) \);

then, also \( s_k = \theta_{k-1}(t' + r') \) holds, and this implies that

\[
 v(\theta_k(s_k)) = v(\theta_k(\theta_{k-1}(t' + r'))) = v(\theta_k(\theta_{k-1}(t'))) + v(\theta_k(\theta_{k-1}(r')))
\]

and then

\[
 v(\theta_k(\theta_{k-1}(t'))) + v(\theta_k(\theta_{k-1}(r'))) \overset{\omega}{\Rightarrow} v(\theta_k(\theta_{k-1}(t'))) + v(\theta_k(\theta_{k-1}(r'))).
\]
Therefore, since, because of (a) and (b),

\[ v(\theta_k(\theta_{k-1}(t'))) + v(\theta_k(\theta_{k-1}(r))) = v(\theta_k(\theta_{k-1}(s_{k-1}))), \]
\[ v(\theta_k(\theta_{k-1}(s_{k-1}))) \in \sigma(T^2_W(\{\lambda\})) \] holds.

By proceeding this way bottom-up, eventually we obtain

\[ v(\theta_k(\theta_{k-1} \cdots (\theta_1(s)) \cdots )) = v(\theta(s)) = s[x_i/t_i, i = 1, \ldots, n] \in \sigma(T^k_W(\{\lambda\})) \]

and then

\[ (t_1, \ldots, t_n) \in D_F(s, W). \]

**Corollary 1.** \( D_G(s, W) = D_M(s, W) \). (Direct consequence of Theorems 7 and 8).

### 4.5. Towards an Axiomatic Theory of Generalized Clauses

In the previous sections we have characterized the semantics of generalized Horn clauses, from a programming language viewpoint, i.e., in the style of the semantics given for Horn clauses (Van Emden & Kovalski, 1976). The derivation rule, given in Sub-section 3.2 is, on one hand, the evaluation rule of an abstract interpreter for GHC programs. On the other hand, it can be seen as an inference rule, which can be used to define proofs and theorems. The last result in Sub-section 4.4 (equivalence of operational and model-theoretic semantics) is a sort of completeness theorem, which states, for the theory defined by a program \( W \) that

(i) All the ground generalized formulas which can be derived from \( W \) are true in the minimal Herbrand model (and therefore in all the Herbrand models) of \( W \).

(ii) All the ground generalized formulas which are true in the minimal Herbrand model can be syntactically derived from \( W \).

We are still left with the problem of formalizing an axiomatic theory of generalized Horn clauses in which the notion of generalized formula and the properties of the operators \( + \) and \( \rightleftharpoons \) are expressively characterized by a suitable set of axioms. Such a formalization will eventually allow us to investigate the relation between generalized Horn clauses and standard first order Horn clauses. In the next section, we will provide such a formalization. We will first define a generalization of first-order predicate calculus. From the syntactical viewpoint the generalization consists of:

(i) The notions of generalized formula (obtained by composition through the operator \( + \)) and of generalized clause (containing the operator \( \rightleftharpoons \)), which act as standard atomic formulas.
(ii) A set of axiom schemata which characterize the properties of + and \( \, \mapsto \, \)\. It is worth noting that the standard predicate calculus inference rules still apply. Generalized Horn clauses will then be shown to be specific generalized first-order theories, which satisfy simple syntactical constraints. For these theories, we will prove that every ground generalized formula which has a refutation, according to the derivation rule of Sub-section 3.2, can be derived from the axioms.

5. AN AXIOMATIC THEORY OF GENERALIZED HORN CLAUSES

5.1. Generalized First-Order Predicate Calculus

5.1.1. Syntax. The alphabet consists of symbols belonging to the sets \( V, F, \) and \( P \) and of special symbols.

(i) \( V \) is a set of variable symbols
\[
x, x_1, \ldots, x_n, \ldots, y, y_1, \ldots, y_n, \ldots
\]

(ii) \( F \) is a set of \( n \)-adic function symbols
\[
f^0, f^1, \ldots, f_n, \ldots, f^1, f^1, \ldots
\]

Zeradic function symbols are the constant symbols.

(iii) \( P \) is a set of \( n \)-adic predicate symbols:
\[
P^0, P^1, \ldots, P_n, \ldots, P^1, P^1, \ldots
\]

(iv) The special symbols are: \( +, \, \mapsto +, \, \text{not}, \, \rightarrow, \, \forall, \, (, ) \).

Terms and atomic formulas are defined as follows, according to the standard first-order definition:

\[
\langle \text{term} \rangle ::= \langle \text{constant symbol} \rangle | \langle \text{variable symbol} \rangle \\
\quad f^1(\langle \text{term} \rangle) | \cdots | f^n(\langle \text{term} \rangle \cdots \langle \text{term} \rangle)
\]

\[
\langle \text{atomic formula} \rangle ::= P^0 | \cdots | P^1(\langle \text{term} \rangle) | \cdots \\
\quad P^n(\langle \text{term} \rangle \cdots \langle \text{term} \rangle).
\]

Generalized formulas are defined as

\[
\langle \text{non-null generalized formula} \rangle ::= \\
\quad \langle \text{atomic formula} \rangle | \langle \text{atomic formula} \rangle + \langle \text{non-null generalized formula} \rangle \\
\quad \langle \text{generalized formula} \rangle ::= + | \langle \text{non-null generalized formula} \rangle
\]
Generalized clauses are defined as

\[
\text{\langle definite clause\rangle ::=}
\]

\[
\text{\langle non-null generalized formula\rangle \leftarrow \text{\langle generalized formula\rangle}}
\]

\[
\text{\langle goal statement\rangle ::= \text{not} \text{\langle generalized formula\rangle}}
\]

\[
\text{\langle generalized clause\rangle ::= \langle definite clause\rangle | \langle goal statement\rangle}.
\]

Generalized formulas and definite clauses act as the atomic formulas in the standard first order predicate calculus. Hence (generalized) well-formed formulas are defined as

\[
\text{\langle w-f formula\rangle ::= \text{\langle generalized formula\rangle | \langle definite clause\rangle | not((\text{\langle w-f formula\rangle}) | ((\text{\langle w-f formula\rangle})) | \forall \text{\langle variable symbol\rangle (\text{\langle w-f formula\rangle})}.}
\]

5.1.2. Axioms and theorems. The axioms of generalized first-order predicate calculus include all the axioms schemata obtained from first-order axiom schemata by replacing the standard definition with the new definition of well-formed formula.

If \( A, B \) are (generalized) well-formed formulas:

(A1) \( A \rightarrow (B \rightarrow A) \).

(A2) \( (A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C)) \).

(A3) \( (B \rightarrow A) \rightarrow ((B \rightarrow A) \rightarrow B) \).

(A4) \( (\forall x(A(x))) \rightarrow A(t) \) (where \( x \) occurs in \( A(x) \) and \( t \) is free from \( x \) in \( A \)).

(A5) \( (\forall x(A \rightarrow B)) \rightarrow (A \rightarrow \forall x(B)) \) if there are no free occurrences of \( x \) in \( A \).

In addition, the following axiom schemata define the properties of the operators \(+\) and \(\leftarrow ^+\). If \( r, s, \) and \( t \) are generalized formulas:

(B1) \( r + s \rightarrow s + r \) (commutative property).

(B2) \( (r \leftarrow ^+ s) \rightarrow (s + t \rightarrow r + t) \) (additive property of \(\leftarrow ^+\)).

(B3.1) \( r \rightarrow r + \lambda \).

(B3.2) \( r + \lambda \rightarrow r \) (\( \lambda \) is the neutral element of \(+\)).

(B4) \( \lambda \).

Inference rules. The inference rules of our calculus are the same of standard first-order predicate calculus:
(R1) \( A, A \rightarrow B \Rightarrow B \) (modus ponens).

(R2) \( A \Rightarrow \forall x(A) \) (generalization).

A (generalized) theory is characterized by a set of generalized well-formed formulas \( W \) (which are the theory-specific axioms). A demonstration (or proof) in a theory is a finite sequence of well-formed formulas \( A_1, \ldots, A_n \) such that every \( A_k \) (1 \( \leq k \leq n \)):

1. is one of the calculus axioms (A1–A5, B1–B4) or one of the theory specific axioms, or
2. is derived from some of the formulas \( A_1, \ldots, A_{k-1} \) by one of the inference rules.

A (generalized) theorem of a theory is the last formula of a demonstration. The notation \( W \vdash A \) means that \( A \) is a theorem of \( W \).

5.2. Semantics (Model-Theoretic Semantics)

5.2.1. Herbrand semantics of a first order generalized theory. The semantics of first order generalized theories, in analogy with the semantics of standard Horn clauses, will be given in terms of Herbrand models, defined on the set \( U \), which is the standard Herbrand universe. The general notion of models and arbitrary domains can be easily obtained by a suitable adaptation of our definitions.

Let \( W \) be a set of generalized well-formed formulas (axioms):

— The Herbrand universe \( U \) is the set of all the ground terms which can be obtained from constants and function symbols occurring in the clauses of \( W \).

— The Herbrand base \( B \) of \( W \) is the set of all the ground generalized formulas which can be obtained by applying the predicate symbols occurring in the formulas of \( W \) to the terms of \( U \).

— A Herbrand interpretation \( I \) of \( W \) is any subset of \( B \) containing \( \lambda \) such that, if \( s_1, \ldots, s_n \) \((n \geq 2)\) are in \( I \), then \( s_1 + \cdots + s_n \) is not in \( I \).

Let \( I \) be an interpretation. The following definition allows us to verify if a well-formed formula is true under \( I \).

(i) If \( r \) is a ground generalized formula, \( r \) is true under \( I \) iff there exist \( s_1, \ldots, s_n \) in \( I \), such that \( r = s_1 + \cdots + s_n \).

(ii) If \( r \) is a ground generalized formula, and \( s \) is a ground non-null generalized formula, then the generalized clause

\[
  s \leftarrow^+ r
\]
is true in $I$ iff for every ground formula $t$, if $r + t$ is true in $I$, then $s + t$ is true in $I$, as well.

(iii) If $A$ is a ground well-formed formula, then $\neg(A)$ is true in $I$ iff $A$ is not true in $I$.

(iv) If $A$, $B$ are ground well-formed formulas, then $(A) \rightarrow (B)$ is true in $I$ iff whenever $A$ is true in $I$ then also $B$ is true in $I$.

(v) If $A$ contains free occurrences of the variable $x$, then $\forall x (A)$ is true in $I$ iff, for every valuation $v$ of $x$, $v(A)$ is true in $I$, where $v(A)$ denotes the formula obtained from $A$ by substituting every occurrence of $x$ with $v(x)$.

(vi) If $A$ is a well-formed formula, then $A$ is true under $I$ iff for every valuation $v$ of all its free variables, $v(A)$ is true in $I$.

Herbrand models. According to standard first order logic, a Herbrand model of a set of axioms is any Herbrand interpretation $I$ such that the axioms are true under $I$.

5.2.2. Soundness of generalized predicate calculus. It is easy to see that the axioms of (pure) generalized predicate calculus are true in every Herbrand interpretation, and then every interpretation is a model of (pure) generalized predicate calculus, in fact:

— (A1)–(A5) are true in every interpretation because of definitions (iii)–(vi).

— (B1) and (B3) are true in every interpretation because of definition (i).

— (B2) is true in every interpretation because of definition (ii).

— (B4) is true because every interpretation contains $\lambda$.

Moreover, inference rules preserve the truth, namely:

1. If $A$ and $A \rightarrow B$ are true in $I$, then also $B$ is true in $I$ (this comes directly from (iv))

2. If $A$ is true in $I$, then also $\forall x (A)$ is true in $I$ (this comes from (v)).

In other words, all the formulas derived, by the inference rules, from formulas true in $I$, are true in $I$.

**Theorem 9** (Soundness Theorem of Generalized Predicate Calculus). If $W$ is a set of axioms, then every theorem of $W$ is true in every model of $W$. (Hence theorems of pure generalized predicate calculus are true in every interpretation of $W$.)
Proof (by induction on the number of formulas in the proof). Let $A = A_n$ be the last well-formed formula of a proof $D = (A_1, \ldots, A_n)$.

We prove that every $A_k$ is true:

(i) $(k = 1)$ $A_1$ is an axiom of the theory and then is necessarily true in every model.

(ii) If $A_1, \ldots, A_k$ $(k \leq n - 1)$ are true in $I$ then also $A_{k+1}$ is true in $I$. In fact, $A_{k+1}$ is an axiom or is obtained from some of the formulas $A_1, \ldots, A_k$ (which are true by hypothesis) by using an inference rule, which preserves the truth.

5.3. The Relation between Generalized First-Order Predicate Calculus and GHC

In the following, we will analyze the relation between the above defined axiomatic calculus and the calculus of generalized Horn clause defined in the previous sections. We will first introduce a syntactic restriction on the calculus, namely we consider those theories only, whose specific axioms are definite clauses. Hence our theories correspond exactly to the programs of the previous sections. From a semantic viewpoint, it can be proved that, for the class of models of such theories, every set of models has a glb (with respect to set inclusion) which is the intersection of the models. The proof is very similar to the proof given in Section 4.

The class of Herbrand models of a theory $W$ is then a complete lattice with respect to set inclusion. Let $M$ be the least model. Clearly, $M$ contains exactly those ground generalized formulas that are contained in every model of $W$. We are specifically interested in those theorems of $W$ which are generalized formulas. If we denote by $D$ the class of all the generalized formulas derivable from $W$, the following property holds, because of the soundness theorem.

**Corollary 2.** $D$ is included in $M$.

Consider now the operational semantics of generalized Horn clauses. Let $R$ be the set of all the ground generalized formulas which are refutable in a program $W$, that is,

$$r \in R \quad \text{iff} \quad r \xrightarrow{W}^* \lambda.$$

**Theorem 10.** $M$ is included in $R$.

The proof is very similar to that of Theorem 8 (completeness theorem for GHC).
Now we prove that $R \subseteq D$, namely that every ground generalized formula which has a refutation in $W$ can be derived from the axioms of $W$ (i.e., it is a theorem). It is worth noting that the following theorem applies to generalized formulas only and does not hold, in general, for well-formed formulas.

**Theorem 11.** For every theory $W$, $R \subseteq D$.

**Proof** (by induction on the number of derivations in the refutation). We prove a more general result, that is, if $r \vdash_w^\vartheta \lambda$ then $\vartheta(r)$ is a theorem.

Let $r \vdash_w^\vartheta \lambda$. There exist $r_1, \ldots, r_{n+1}$ and $\vartheta_1, \ldots, \vartheta_n$ such that

$$r = r_1 \vdash_w^{\vartheta_1} r_2 \vdash_w^{\vartheta_2} \cdots r_n \vdash_w^{\vartheta_n} r_{n+1} = \lambda, \quad \vartheta = \vartheta_1 \bullet \vartheta_2 \bullet \cdots \bullet \vartheta_n,$$

where every $r_{i+1}$ is obtained from $r_i$ through a clause $C_i$ and a substitution $\vartheta_i$.

(i) If $n = 1$ then $r = \lambda$ and then $r$ is a theorem (it is an axiom).

(ii) Assume that for every $s$ refutable in $n$ steps with a substitution $\phi$, $\phi(s)$ is a theorem. Then, if $r$ is refutable in $n + 1$ steps with the substitution $\vartheta$, also $\vartheta(r)$ is a theorem.

**Proof.** Let $r$ be refutable in $n + 1$ steps, and consider the relation (1). By inductive hypothesis, $\vartheta'(r_2)$ is a theorem, where $\vartheta' = \vartheta_2 \bullet \cdots \bullet \vartheta_n$. Moreover, the clause $C_1$ which is used to derive $r_2$ from $r_1$ must satisfy the following conditions:

1. $C_1 = s_1 \leftarrow s_2$,
2. $\vartheta_i(r_1) = \vartheta_i(s_1 + t)$ for a suitable $t$,
3. $r_2 = \vartheta_i(s_2 + t)$ for the same $t$.

The well-formed formula

$$F = s_2 + t \rightarrow s_1 + t$$

is a theorem, since it can be derived from axiom (B2) and clause $C_1$. Therefore, also

$$F' = \vartheta_i(s_2 + t) \rightarrow \vartheta_i(s_1 + t)$$

is a theorem, since it can be derived from the theorem $F$, the axiom (A4) and the inference rule (R2). Analogously,

$$\vartheta'(r_2) \rightarrow \vartheta'(\vartheta_1(r_1))$$
is a theorem, too. Being \( \theta'(r_2) \) a theorem, by (R1) we eventually obtain the theorem \( \theta'(\theta_1(r_1)) \), i.e., \( \theta(r) \), since \( \theta = \theta_1 \bullet \theta' \) and \( r = r_1 \).

Theorem 11 states that, for ground generalized formulas, the inference system based on the derivation rule of Sub-section 3.2 is equivalent to the inference system consisting of axioms (A1)-(A5), (B1)-(B4) and the standard inference rules (R1)-(R2). This result extends to generalized Horn clauses properties that have been proved (Robinson, 1965) for the resolution principle in standard first-order predicate calculus.

6. The Relation between GHC and HCL

6.1. Monadic Generalized Horn Clauses

The language of generalized Horn clauses described in the previous sections is an extension of Horn clauses, since any Horn program can be “translated” into an equivalent GHC program by simply replacing the symbol “,” with “+” and the symbol “⇌” with “+⇌”, as we will show.

Let \( C \) be an HCL clause, i.e., a definite clause (rewrite rule or assertion), or a negative clause (goal statement), and let \( C^G \) be the expression obtained by replacing each occurrence of “,” and “⇌” with “+” and “+⇌”, respectively.

\( C^G \) is a legal GHC clause (definite clause or goal statement). In the case of definite clauses, \( C^G \) has exactly one atomic formula in the left part. These definite clauses are called monadic clauses.

If \( W \) is an HCL program, the corresponding GHC program is defined

\[
W^G = \{ C^G \mid C \in W \}
\]

**Theorem 12.** If \( W \) is an HCL program, \( S \) is an HCL goal statement, and \( x_1, \ldots, x_n \) are the variable occurring in \( s \) (and therefore in \( s^G \)), then

\[
D_0(s, W) = D_0^G(s^G, W^G),
\]

where

\[
D_0(s, W) = \{ v(\theta(s)) \mid v \text{ is a valuation and } s \overset{\theta}{\underset{W}{\vdash}^*} \lambda \}
\]

and

\[
D_0^G(s^G, W^G) = \{ v'(\theta(s^G)) \mid v' \text{ is a valuation and } s^G \overset{\theta}{\underset{W^G}{\vdash}^*} \lambda \}.
\]

Note that \( s^G \overset{\theta}{\underset{W^G}{\vdash}^*} \lambda \) denotes a GHC derivation, while \( s \overset{\theta}{\underset{W}{\vdash}^*} \lambda \) denotes an HCL derivation.
Proof. We only need to prove that the GHC derivation rule, in the case of monadic clauses, corresponds exactly to the HCL derivation rule. Let

\[ C = A \leftarrow B_1, \ldots, B_n \]

be an HCL definite clause, and

\[ s = \leftarrow A_1, \ldots, A_m \]

be an HCL goal statement. The corresponding GHC clauses are

\[ C^G = A \uparrow B_1 + \cdots + B_n \]

\[ s^G = \uparrow A_1 + \cdots + A_m \]

then, if \( A \) and \( A_i \) are unifiable with most general unifier \( \theta \), the goal statement

\[ t = \leftarrow \theta(A_1, \ldots, A_{i-1}, B_i, \ldots, B_n, A_{i+1}, \ldots, A_m) \]

can be derived in HCL from \( s \) and \( C \). Correspondingly, the goal statement

\[ t' = \leftarrow \theta(A_1 + \cdots + A_{i-1} + B_i + \cdots B_n + A_{i+1} + \cdots + A_m) \]

is derivable in GHC from \( C^G \) and \( s^G \). Moreover, \( t' = t^G \), i.e., \( t' \) is exactly the g.s. which is obtained by replacing \( \leftarrow \) with \( + \) and \( \leftarrow \) with \( \uparrow \) in \( t \).

Therefore, \( s \mapsto \leftarrow \theta_{\rightarrow} t \) in HCL if and only if \( s^G \mapsto \leftarrow \theta_{\rightarrow} t^G \) in GHC and, in general, \( s \mapsto \leftarrow \theta_{\rightarrow} t \) if \( s^G \mapsto \leftarrow \theta_{\rightarrow} t^G \).

Remark. Theorem 12 shows that a GHC program consisting of monadic clauses only has the same operational semantics (and therefore the same denotational and model-theoretic semantics) of the corresponding HCL program. Actually, the correspondence is even stronger. In fact, there exists a one-to-one correspondence, not only for the final results (the halt statement and the composition of the substitutions), but also for every computation step, since the derivation rules are essentially similar.

6.2. Generalized Horn Clauses with Simple Least Model

There exists a one-to-one correspondence between the least model of GHC monadic clauses and HCL clauses. In fact, it is possible to prove that the least model of a GHC program \( W \) consisting of monadic clauses, contains atomic formulas only. This is a consequence of the following theorem, and of the equivalence of the model-theoretic and fixed-point semantics.

**Theorem 13.** If \( W \) contains monadic clauses only, then \( \bigcup_{k \geq 0} T_k^W(\lambda) \) (which is the least fixed-point interpretation) contains atomic formulas only.
Proof. We must prove that for every $k$, $T^k_w(\{\lambda\})$ contains atomic formulas only. By induction on $k$:

(i) If $k = 0$, $T^0_w(\{\lambda\}) = \{\lambda\}$ does not contain, obviously, generalized formulas.

(ii) If $T^k_w(\{\lambda\})$ contains atomic formulas only, then also $T^{k+1}_w(\{\lambda\})$ contains atomic formulas only.

Proof. Note that

\[ T^{k+1}_w(\{\lambda\}) = T_w(T^k_w(\{\lambda\})) \]

\[ = \rho (\{ s \in B \mid s \xrightarrow{\lambda} r, r \in \sigma(T^k_w(\{\lambda\})) \}) \cup \{\lambda\}. \]

Let

\[ H = \{ s \in B \mid s \xrightarrow{\lambda} r, r \in \sigma(T^k_w(\{\lambda\})) \} \]

and let

\[ s = A_1 + \cdots + A_n \in H \]

then

\[ A_1, \ldots, A_n \in H. \]

In fact, if $s \xrightarrow{\lambda} r, r \in \sigma(T^k_w(\{\lambda\}))$, then there exists a ground instance of a clause of $W$ having the form

\[ A_i \xrightarrow{\lambda} B_1 + \cdots + B_m \]

such that

\[ r = A_1 + \cdots + A_{i-1} + B_1 + \cdots + B_m + A_{i+1} + \cdots + A_n. \]

Then, by inductive hypothesis,

\[ B_1, \ldots, B_n, A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n \in T^k_w(\{\lambda\}) \subseteq T^{k+1}_w(\{\lambda\}) \]

and, moreover (since $A_i \xrightarrow{\lambda} B_1 + \cdots + B_n$),

\[ A_i \in T^{k+1}_w(\{\lambda\}). \]

It is worth noting that there exist GHC programs $W$ which contain nonmonadic clauses, such that their least models contain atomic formulas only.
Consider, for example, the program

\[ W = \{ A \leftarrow \lambda, B \leftarrow \lambda, A + B \leftarrow \lambda \}. \]

The least model of \( W \) is \( \{ A, B \} \), which is also the least model of

\[ W' = \{ A \leftarrow \lambda, B \leftarrow \lambda \}. \]

Hence the clause \( A + B \leftarrow \lambda \) does not modify the model. If this is the case, the clause is called a property of \( W \) (since it is true in the least model of \( W \)).

In general, if the least model of a program \( W \) contains atomic formulas only, then the nonmonadic clauses which possibly are in \( W \) are certainly properties; namely, they can be removed from \( W \) without modifying the least model. Note that this is not the case for the other models of \( W \). Hence the set of models of \( W \) can be modified by the elimination of a property from \( W \). This result is a consequence of the following theorem.

**Theorem 14.** If the least model of \( W \) contains generalized formulas whose length is at most \( n - 1 \) (i.e., generalized formulas containing at most \( n - 1 \) atomic formulas) then the clauses of \( W \) with \( n \) or more atoms in the left part are properties.

**Proof.** The elimination from \( W \) of a clause of the form

\[ A_1 + \cdots + A_k \leftarrow B_1 + \cdots + B_m, \quad k \geq n, \]

could cause the elimination from the least model of \( W \) of formulas of the form

\[ F = v(A_1 + \cdots + A_k) + t, \]

where \( v \) is a valuation and \( t \) is a ground generalized formula. However, since \( F \) contains more than \( n - 1 \) atomic formulas, it is not contained in the least model.

Another feature of the programs whose least models contain atomic formulas only is that the operator \( + \) can be interpreted as the conjunction operator, i.e., \( s = A_1 + \cdots + A_k \) is true in all the models of \( W \) iff every \( A_i \) is true in all the models of \( W \). In fact, \( s \) is true in every model of \( W \)

1. iff \( s \) is true in the least model \( M \),
2. iff \( s \in \sigma(M) \),
3. iff \( \forall i, A_i \in M \) (since \( M \) contains atomic formulas only),
4. iff \( \forall i, A_i \) is true in every model.
A similar result holds for $\vdash$, which can be interpreted as $\leftarrow$, but only with respect to the least model: If $W$ has a least model consisting of atomic formulas only, then $s \vdash r$ is true in the least model iff $s \leftarrow r$ is true in the least model. In fact, $s \vdash r$ is true in the least model $M$, iff for every ground formula $t$ and every valuation $v$ such that $v(r) + t$ is true in $M$, $v(s) + t$ is true in $M$, iff (since $+$ corresponds to conjunction) for every valuation $v$ such that $v(r), t$ are true in $M$, $v(s), t$ are true in $M$, iff for every $v$ such that $v(r)$ is true in $M$, $v(s)$ is true in $M$, iff $s \leftarrow r$ is true in $M$.

6.3. Adding New Clauses to GHC Programs

Let us consider the following example: Let $W$ be the program defined by the clauses:

(1) $\text{plus}(x, O, x)$, $\vdash \lambda$
(2) $\text{plus}(x, s(y), s(z)) \vdash \text{plus}(x, y, z)$
(3) $\text{minus}(x, O, x)$, $\vdash \lambda$
(4) $\text{minus}(s(x), s(y), z) \vdash \text{minus}(x, y, z)$.

Clearly, the least model $M$ is

$$M = \{ \text{plus}(0, 0, 0), \text{plus}(0, 1, 1), ..., \text{minus}(0, 0, 0), \text{minus}(1, 1, 0), ... \}.$$ 

Let us now consider the clause

(5) $\text{plus}(x, y, z) + \text{minus}(z, x, y) \vdash \text{minus}(z, x, y)$.

The least model of the program $W'$, obtained from $W$ by adding this new clause, is still $M$. The operational meaning of adding this clause is that the computation of some goal statements (the goal statements containing a subformula matching the head of (5)) is faster. It is worth noting that the right hand of (5) controls that the first argument of the predicate minus is greater than the second one.

We want now to look into the problem of the insertion of a new clause into a program $W$, to identify conditions under which the insertion does not modify the least model. In other words which are the clauses, not contained in $W$, which are properties of $W$.

**Theorem 15.** If $W$ has a least model $M$ containing formulas whose length does not exceed $n$, then any clause $C$ of the form

$$s \vdash r,$$
where the length of \( r \) is \( k \), is a property of \( W \) if: for every valuation \( v \) and every ground formula \( t \) whose length does not exceed \( k \ast (n - 1) \), if \( v(r) + t \) is true in \( M \) then \( v(s) + t \) is also true in \( M \).

**Proof.** This condition is sufficient to guarantee that for every \( t' \)

\[
r + t' \rightarrow s + t'
\]
is true in \( M \). In fact, if

\[
v(r) = R_1 + \cdots + R_k
\]
and

\[
t' = T_{1,1} + T_{1,2} + \cdots + T_{k,n-1} + t'' \quad (t'' \neq \lambda)
\]
is a formula of length \( m \) \((m \geq k \ast (n - 1))\), \( v(r) + t' \) can be decomposed in the worst case as

\[
\begin{align*}
& R_1 + T_{1,1} + \cdots + T_{1,n-1} & F_1 \\
& + R_2 + T_{2,1} + \cdots + T_{2,n-1} & F_2 \\
& \vdots \\
& + R_k + T_{k,1} + \cdots + T_{k,n-1} & F_k \\
& + t''
\end{align*}
\]
(where \( t'' \) is long at least \( m - k \ast (n - 1) \)). If \( v(r) + t' \) is true in \( M \), then \( t' \) can be decomposed in two formulas, \( t'_1, t'_2 \), such that

1. \( t'_1 \) is long at most \( k \ast (n - 1) \).
2. \( t'_2 \) is true in \( M \).
3. \( v(r) + t'_1 \) is true in \( M \).

From (1) and (3), by hypothesis, \( v(s) + t'_1 \) is true in \( M \), and therefore, by (2), \( v(s) + t' \) is also true in \( M \).

Two interesting consequences of Theorem 15 are

1. If the least model of \( W \) contains atomic formulas only \((n = 1)\) Theorem 15 asserts that

   if \( s \leftarrow r \) is true in \( M \) then the clause \( C^G = s \leftarrow r \) is a property of \( W \) \((s \leftarrow r \) is true in \( M \)).

2. For unit clauses \((k = 0)\), Theorem 15 asserts that

   if \( s \) is true in \( M \) then the unit clause \( s \leftarrow \lambda \) is a property of \( W \).
Note that in the last case $s \vdash \lambda$ is true not only in $M$, but also in every model of $W$, hence $s \vdash \lambda$ is valid (in the logical sense) for $W$. In general, every clause valid in $W$ is also a property of $W$, but the converse does not hold.

7. Final Remarks

Generalized Horn clauses can easily be implemented by a symple extension of Horn clause interpreters. The extension is meaningful only if the interpreter defines a parallel implementation of and. Future work will be related to proving properties of programs, on the basis of the axiomatization given in Section 5.

Another area, which is worth being deeply investigated is the possibility of defining higher level synchronization mechanisms. In fact, our basic mechanism is very low level and could be viewed as corresponding to the semaphor concept in traditional programming languages. Higher level constructs more structured and easy to understand could be defined in term of our construct, thus inheriting their syntactic and semantic properties.

Received: September 19, 1983; Accepted: March 28, 1984

References


Degano, P. (1979), Una classe di schemi ricorsivi non-deterministici paralleli, Calcolo 14, 97–119.


