Negation in Chronolog*

P. Rondogiannis

Department of Computer Science, University of Ioannina
P.O. Box 1186, GR-45110 Ioannina, Greece
e-mail: prondo@cs.uoi.gr

Abstract

We consider stratified negation in temporal logic programming. We demonstrate that the cycle-sum test (which was initially proposed for detecting deadlocks in temporal functional programming) can also be used as a syntactic local stratification test for temporal logic programming. Therefore, on the one hand we exhibit a class of temporal logic programs with negation which have a well-defined semantics, and on the other hand we provide further evidence that the cycle-sum test is a fundamental one in the area of temporal programming.

Keywords: Temporal Logic Programming, Negation, Stratification.

1 Introduction

Negation in logic programming has received considerable research attention largely due to its applications in areas such as artificial intelligence and deductive databases. From a semantic point of view, the addition of negation in classical logic programming is far from straightforward, and many different approaches have been developed [PP90, AB94]. One of the earliest such approaches is the so-called stratified negation [ABW88]. Intuitively, a stratified logic program is one in which negation is not used in a circular way, and this (syntactically determinable) condition ensures that the program has a unique perfect model. Stratification was generalized by T. Przymusinski [Prz88] to local stratification which is more powerful but can not in general be detected syntactically.

In this paper, we consider stratified negation in temporal logic programming (and more specifically in the context of the temporal logic programming language Chronolog [Wad88, OW92a, OW92b]). The simple stratification test of [ABW88] appears to be too restrictive for temporal logic programming with negation: even the simplest programs, that have an obvious meaning, fail to pass the test. Consider for example the following program:

\[
\text{first } p(a).
\]
\[
\text{next } p(X) \leftarrow \neg p(X).
\]

The declarative reading of the above program is: "p is true of a at time 0. Moreover, p is true of X at time t + 1 if p is not true of X at time t". A temporal model of the program

*This work has been partially supported by the Greek General Secretariat of Research and Technology under the project "Logic Programming Systems and Environments for Developing Logic Programs" of ΕΕΕΝΕΑ'95, contract no 952.

89
that suggests itself is the one in which \( p \) is true of \( a \) at the time points 0, 2, 4, \ldots\). However, the simple stratification test fails for the above program, due to the circularity in the second clause.

However, programs such as the above one are not truly circular. The meaning of temporal logic programs depends on an implicit time parameter which needs to be taken into consideration or otherwise most programs would have to be rejected. In other words, an effective stratification test for temporal logic programming should examine for temporal circularities in the program. In [Wad81] W. Wadge developed the cycle-sum test which ensures that a given temporal functional program (of the language Lucid [WA85] in particular) is deadlock free. We adapt the test to apply to temporal logic programming with negation, and we show that programs that pass the test are locally stratified. Our contribution is therefore twofold: on the one hand we exhibit a class of temporal logic programs with negation which have a well-defined semantics, and on the other hand we provide further evidence that the cycle-sum test is a fundamental one in the area of temporal programming in general.

The rest of the paper is organized as follows: section 2 contains preliminary material on temporal logic programming. Section 3 presents a transformation algorithm from temporal logic programs into classical ones, in such a way that the model theory of the initial programs is preserved. Section 4 defines the cycle-sum test for the classical logic programs that result from the transformation. Section 5 demonstrates that programs passing the cycle-sum test are locally stratified. Section 6 concludes the paper with a discussion of possible extensions.

2 Preliminaries: Temporal Logic Programming

The temporal logic programming language we consider here is the language Chronolog [Wad88, OW92a, OW92b]. In particular, we consider an extension of Chronolog that allows negation in the bodies of the rules of a program.

The syntax of Chronolog programs is an extension of the syntax of classical logic programs [Llo87] with two temporal operators, namely first and next. The declarative reading of these temporal operators will be discussed shortly. A temporal reference is a sequence (possibly empty) of the above temporal operators. We will often write next\(^k\) to represent a sequence of \( k \) next operators. A canonical temporal reference is a temporal reference of the form first next\(^k\). An open temporal reference is a temporal reference of the form next\(^k\).

A temporal atom is an atom preceded by either a canonical or an open temporal reference. A temporal clause is a formula of the form:

\[
H \leftarrow A_1, \ldots, A_k, \neg B_1, \ldots, \neg B_m.
\]

where \( H, A_1, \ldots, A_k, B_1, \ldots, B_m \) are temporal atoms and \( m, k \geq 0 \). If \( m = k = 0 \), the clause is said to be a unit temporal clause. A temporal program is a finite set of temporal clauses.

The temporal logic programming language we consider here, is based on a relatively simple temporal logic (TL). TL is based on a linear notion of time with unbounded future. The set of time moments can then be modeled by the set \( \mathcal{N} \) of natural numbers. The operator first is used to express the first moment in time (i.e. time 0), while next refers to the next moment in time. The syntax of the formulas of TL is an extension of the syntax of first-order logic with two formation rules: if \( A \) is a formula, then so are first \( A \) and next \( A \).

The semantics of temporal formulas of TL are given using the notion of temporal interpretation [Org91]:

90
Definition 2.1. A temporal interpretation $I$ of the temporal logic $TL$ comprises a non-empty set $D$, called the domain of the interpretation, over which the variables range, together with an element of $D$ for each variable; for each $n$-ary function symbol, an element of $[D^n \rightarrow D]$; and for each $n$-ary predicate symbol, an element of $[\mathcal{N} \rightarrow 2^{D^n}]$.

In the following definition, the satisfaction relation $\models$ is defined in terms of temporal interpretations. $\models_{I,t} A$ denotes that a formula $A$ is true at a moment $t$ in some temporal interpretation $I$.

Definition 2.2. The semantics of the elements of the temporal logic $TL$ are given inductively as follows:

1. If $f(e_0, \ldots, e_{n-1})$ is a term, then $I(f(e_0, \ldots, e_{n-1})) = I(f)(I(e_0), \ldots, I(e_{n-1}))$.

2. For any $n$-ary predicate symbol $p$ and terms $e_0, \ldots, e_{n-1}$, $\models_{I,t} p(e_0, \ldots, e_{n-1}) \iff (I(e_0), \ldots, I(e_{n-1})) \in I(p)(t)$

3. $\models_{I,t} \neg A$ iff it is not the case that $\models_{I,t} A$

4. $\models_{I,t} A \land B \iff \models_{I,t} A$ and $\models_{I,t} B$

5. $\models_{I,t} A \lor B \iff \models_{I,t} A$ or $\models_{I,t} B$

6. $\models_{I,t} (\forall x) A \iff \models_{I[d/x],t} A$ for all $d \in D$, where the interpretation $I[d/x]$ is the same as $I$ except that the variable $x$ is assigned the value $d$.

7. $\models_{I,t} \text{first } A \iff \models_{I,0} A$

8. $\models_{I,t} \text{next } A \iff \models_{I,t+1} A$

If a formula $A$ is true in a temporal interpretation $I$ at all moments in time, it is said to be true in $I$ (we write $\models_I A$) and $I$ is called a model of $A$.

The semantics of Chronolog are defined in terms of temporal Herbrand interpretations. A notion that is crucial in the discussion that follows, is that of canonical atom:

Definition 2.3. A canonical temporal atom is a temporal atom whose temporal reference is canonical.

As in the theory of classical logic programming [Llo87], the set $Up$ generated by constant and function symbols that appear in $P$, called Herbrand universe, is used to define temporal Herbrand interpretations. Temporal Herbrand interpretations can be regarded as subsets of the temporal Herbrand Base $TB_P$ of $P$, consisting of all ground canonical temporal atoms whose predicate symbols appear in $P$ and whose arguments are terms in the Herbrand universe $Up$ of $P$. A temporal Herbrand model is a temporal Herbrand interpretation, which is a model of the program.
3 Temporal vs Classical Logic Programs

In this section we demonstrate that a temporal logic program can be easily transformed into a classical one whose model theory is closely related to that of the initial program. Intuitively, given a temporal program \( P \) we obtain its classical counterpart \( P^* \) by adding to the predicates of \( P \) an extra parameter that represents explicitly the notion of time.

The transformation can be formalized as follows:

- Replace every canonical temporal atom first next
  \( p(e_0, \ldots, e_{n-1}) \)
  in \( P \) by the classical atom
  \( p(s^k(0), e_0, \ldots, e_{n-1}) \).

- Replace every open temporal atom of the form next
  \( p(e_0, \ldots, e_{n-1}) \)
  by the classical atom
  \( p(s^k(T), e_0, \ldots, e_{n-1}) \).

- In the body of every clause, add the atom \( \text{nat}(T) \)
  (whose purpose is to restrict the time parameter to obtaining only natural number values). Also, add to the program the axiomatization of \( \text{nat} \).

In the following, we will also refer to the programs that result from the above transformation as time-classical logic programs.

The above transformation algorithm is illustrated by the following example:

**Example 3.1.** Consider the following temporal logic program:

\[
\begin{align*}
\text{first} & \quad \text{next } p(0). \\
\text{next} & \quad \text{next } p(X) \leftarrow \neg \text{next } p(X).
\end{align*}
\]

The transformation described above results in:

\[
\begin{align*}
p(s(0), 0) & \leftarrow \text{nat}(T). \\
p(s(s(T)), X) & \leftarrow \neg p(s(T), X), \text{nat}(T). \\
\text{nat}(0). \\
\text{nat}(s(X)) & \leftarrow \text{nat}(X).
\end{align*}
\]

which is the classical counterpart of the initial program.

**Definition 3.1.** A Herbrand interpretation of a time-classical program is called normal if:

1. The only atoms regarding the predicate \( \text{nat} \) that it contains are all the atoms of the set
   \( \text{Nat} = \{ \text{nat}(s^k(0)) \mid k \geq 0 \} \).
2. All the other atoms that it contains are of the form \( p(s^k(0), e_0, \ldots, e_{n-1}) \), where \( k \geq 0 \).

The following theorem can then be easily established:

**Theorem 3.1** Let \( P \) be a temporal logic program and \( P^* \) be its classical counterpart. Then, there is a one-to-one correspondence between the temporal Herbrand models of \( P^* \) and the normal Herbrand models of \( P \).
Proof: Given a temporal Herbrand model \( I \) of \( P \), obtain a classical interpretation \( I^* \) by replacing every canonical ground temporal atom first next\(^k\) \( p(e_0, \ldots, e_{n-1}) \) in \( I \) by the classical atom \( p(s^k(0), e_0, \ldots, e_{n-1}) \). It can be easily shown that \( I^* \cup \text{Nat} \) is a model of \( P^* \). On the other hand, when given a normal Herbrand model of the classical program \( P^* \) one can easily obtain in a similar way a temporal Herbrand model of \( P \). 

In the rest of this paper, we will consider and analyze the classical counterpart of a given temporal logic program.

4 The Cycle Sum Test

The classical programs that result from the transformation defined in the previous section, have a specific structure: the first argument of each predicate is the time parameter. In this section, we show that the special structure of these programs allows us to define a syntactic test, which when passed, ensures that the classical program is locally stratified [Prz88].

It is well known [Prz88] that locally stratified logic programs have a unique perfect model, which is taken as the intended meaning of the program. Due to the model correspondence theorem 3.1, we can then take the corresponding temporal Herbrand model as the intended meaning of the initial temporal program.

In the following, we formally define the Cycle Sum test which is applied on the classical logic programs that resulted from the transformation of section 3. The following definitions are needed:

**Definition 4.1.** Let \( A \) be an atom of a time-classical logic program. Then, \( \text{time}(A) \) is the term that corresponds to the first argument of \( A \).

**Definition 4.2.** Let \( P^* \) be a time-classical logic program and \( C \) be a clause in \( P^* \). Let \( H \) be the head of \( C \) and \( A \) be an atom (different from \( \text{nat}(T) \)) in the body of \( C \). The temporal difference \( \text{dif} \) between \( H \) and \( A \) is defined as follows:

\[
\text{dif}(H, A) = \begin{cases} 
  k - m, & \text{if } \text{time}(H) = s^k(0) \text{ and } \text{time}(A) = s^m(0) \\
  k - m, & \text{if } \text{time}(H) = s^k(T) \text{ and } \text{time}(A) = s^m(T) \\
  k - m, & \text{if } \text{time}(H) = s^k(T) \text{ and } \text{time}(A) = s^m(0) \\
  -\infty, & \text{if } \text{time}(H) = s^k(0) \text{ and } \text{time}(A) = s^m(T) 
\end{cases}
\]

**Example 4.1.** Consider the following temporal logic program:

\[
\text{next next next p(X) } \leftarrow \text{next next next next q(X), first next next r(X).}
\]

The corresponding time-classical logic program is (we omit the clauses for \( \text{nat} \)):

\[
p(s(s(s(T))), X) \leftarrow q(s(s(s(s(s(T)))))), X), r(s(s(0)), X), \text{nat}(T).
p(s(s(0)), X) \leftarrow q(s(T), X), \text{nat}(T).
\]

Then, using the definition of \( \text{dif} \) we have:

\[
\begin{align*}
\text{dif}(p(s(s(s(T))), X), q(s(s(s(s(T)))), X)) &= -1 \\
\text{dif}(p(s(s(s(T))), X), r(s(s(0)), X)) &= 1 \\
\text{dif}(p(s(s(0)), X), q(s(T), X)) &= -\infty
\end{align*}
\]
Intuitively, the above values of $\text{dif}$ express how far (in the worst case) the head of a clause leads each atom in the body.

The following definition formalizes the notion of cycle sum graph of a given time-classical logic program.

**Definition 4.3.** Let $P^*$ be a given time-classical logic program. The cycle sum graph of $P^*$ is a directed weighted graph $CG_{P^*} = (V,E)$. The set $V$ of vertices of $CG_{P^*}$ is the set of predicate symbols appearing in $P^*$. The set $E$ of edges consists of triples $(p,q,w)$, where $p,q \in V$ and $w \in Z \cup \{ -\infty \}$. An edge $(p,q,w)$ belongs to $E$ if in $P^*$ there exists a clause with an atom $H$ as its head and an atom $A$ in its body, such that the predicate symbol of $H$ is $p$, the predicate symbol of $A$ is $q$ and $\text{dif}(H,A) = w$.

We can now state the cycle sum test:

**Definition 4.4.** A time-classical logic program $P^*$ passes the cycle sum test if the sum of weights across every cycle in $CG_{P^*}$ is positive.

In the following section we show that every time-classical logic program passing the cycle sum test is locally stratified.

## 5 Justification of the Cycle Sum Test

In this section we show that time-classical logic programs passing the cycle sum test are locally stratified and therefore have a unique perfect Herbrand model. The following definitions are necessary:

**Definition 5.1.** [PP90] Let $P$ be a classical logic program. The dependency graph of $P$ is a graph whose vertex set is the Herbrand base $B_P$ of $P$ and whose edges are determined as follows: if $A$ and $B$ are two atoms in $B_P$, there exists a directed edge from $B$ to $A$ if and only if there exists an instance of a clause in $P$ whose head is $A$ and one of whose premises is either $B$ or $\neg B$. In the latter case, the edge is called negative.

**Definition 5.2.** [PP90] Let $P$ be a classical logic program. For any two ground atoms $A$ and $B$ in $B_P$ we write $A \prec B$ if there exists a directed path in the dependency graph of $P$ leading from $B$ to $A$ and passing through at least one negative edge. We call the relation $\prec$ the priority relation between ground canonical atoms. We write $A \preceq B$ if there exists a directed path from $B$ to $A$.

The following proposition from [PP90] actually defines the notion of local stratification in terms of the priority relation $\prec$.

**Proposition 5.1** [PP90] A logic program $P$ is locally stratified if and only if its priority relation $\prec$ is a partial order and if every increasing sequence of ground atoms under $\prec$ is finite.

The following theorem will be used in the following:

**Theorem 5.1** Let $W = e_1 \cdots e_n$ be a closed walk in a directed graph $G$. Then, there exists a sequence of (not necessarily distinct) cycles $C_1, \ldots, C_k$ in $G$ such that: (i) if $e_i$ appears $m$ times in $W$ then it also appears in exactly $m$ of the cycles, and (ii) every edge that appears in a cycle also appears in $W$. 

94
Proof: By induction on the length of the walk $W$.

We now come to the main theorem of the paper:

**Theorem 5.2** Let $P^*$ be a time-classical logic program that passes the cycle sum test. Then $P^*$ is locally stratified.

**Proof**: Assume that $P^*$ is not locally stratified. This means that there exists an infinite increasing sequence $A_1 < A_2 < \cdots$ of atoms of the Herbrand base of $P^*$. Each atom in the sequence corresponds to a clause of the program whose ground instantiation has the atom as its head. As the sequence is infinite, there exist infinitely many atoms of the sequence that correspond to the same clause of the program, say $C$. These atoms form an infinite subsequence $B_1 < B_2 < \cdots$ of the initial sequence.

Consider now two atoms $B_i$ and $B_{i+1}$ from the new sequence. The first argument of $B_i$ is of the form $s^k(e)$, where $k \geq 0$. The derivation of $B_{i+1}$ from $B_i$ corresponds to a closed walk in the graph $CG_{P^*}$. A walk can be decomposed into a number of cycles (see Theorem 5.1). By the cycle sum test, the sum of the weights that correspond to the edges of the walk is positive. Alternatively, the first argument of $B_i$ "leads" the first argument of $B_{i+1}$ by a positive amount (i.e., the first argument of $B_{i+1}$ is of the form $s^m(e)$, with $m < k$. Therefore, the first arguments of the members of the sequence $B$ decrease in complexity and therefore the sequence can not be infinite. This is a contradiction, which implies that program $P^*$ is locally stratified.

**Lemma 5.1** Let $P$ be temporal logic program and $P^*$ be its classical counterpart. If $P$ passes the cycle sum test then it has a unique perfect model which is also normal.

**Proof**: By Theorem 5.2, $P^*$ is locally stratified, which means that it has a unique perfect model $M^*$ [Prz88]. This model obviously contains $Nat$ due to the two clauses in $P^*$ that define the predicate $nat$. We need to further show that $M^*$ does not contain any other atom regarding the predicate $nat$ and that it does not contain any atom that is not of the form $p(s^k(0), e_0, \ldots, e_{n-1})$. Consider the interpretation $N^*$ that results if we remove from $M^*$ all atoms regarding the predicate $nat$ that do not belong to $Nat$ and all atoms that are not of the form $p(s^k(0), e_0, \ldots, e_{n-1})$. We have $N^* \subseteq M^*$. It can be easily shown that every ground instance of a clause in $P^*$ is true under $N^*$. Therefore, $N^*$ is also a model of $P^*$, which contradicts the fact that perfect models are minimal models [Prz88].

From the above discussion, we conclude that given a temporal logic program $P$, if its classical counterpart $P^*$ passes the cycle-sum test, then $P^*$ is guaranteed to have a unique perfect model $M^*$ which is normal. Therefore, by Theorem 3.1, there exists a temporal Herbrand model $M$ of $P$, which we take as the intended meaning of $P$.

6 Conclusions

In this paper, we have developed a syntactic test (the cycle-sum test) for temporal logic programs with negation. Programs that pass the test have a well-defined meaning. Obviously, not all locally stratified programs can be detected and in fact, many programs that have well-defined semantics are rejected by the test. For example, there exist Chronolog programs that
do not have negation in the bodies of the rules, but fail the test. A simple such example is the following:

\[
\begin{align*}
\text{first } p(a). \\
p(X) &\leftarrow \text{next } p(X).
\end{align*}
\]

This program fails the test but it obviously has a well-defined meaning under the standard semantics of temporal logic programming. The obvious solution is to restrict application of the test on programs that actually use negation. Moreover, notice that programs such as the above in fact contain some form of deadlock [Wad81] and are in some sense against the spirit of temporal logic programming, which views predicates as infinite streams [Wad88].

An interesting topic for further research would be the consideration of other temporal logic programming languages, that are based on different notions of time. One interesting such case is that of branching-time logic programming [RGP97] in which the set of possible worlds of the underlying branching-time logic is the set of lists of natural numbers. Such a language would require a more powerful cycle-test which would have to be applicable to the more complicated underlying set of possible worlds.

Acknowledgment: The author would like to thank Manolis Gergatsoulis for his comments on an earlier version of the paper.

References


