STRATIFIED NEGATION IN TEMPORAL LOGIC PROGRAMMING AND THE CYCLE SUM TEST

P. RONDOGIANNINS

25-98

Preprint no. 25-98/1998

Department of Computer Science
University of Ioannina
451 10 Ioannina, Greece

Stratified Negation in Temporal Logic Programming and the Cycle Sum Test[*]

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 the context of temporal functional programming) can also be used as a syntactic 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, Cycle Sum Test.

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 Chronolog program:

```
first p(a).
next p(X) ← ¬ p(X).
```

[*]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 ΠΕΝΕΔ'95, contract no 952.

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 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.

Programs such as the above one are not however 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 also 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], is deadlock free. The test was later extended by S. Matthews [Mat95] to a wider context (but still in the area of functional programming).

We show that the test is also applicable in the area of temporal logic programming with negation, and we demonstrate that programs that pass the test have a well-defined semantics. Our contribution is therefore twofold: on the one hand we exhibit a class of temporal logic programs with negation which have a clear meaning, 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 have a well-defined meaning. 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]. As an informal introduction to the language, consider the following Chronolog [Wad88] program that simulates the operation of the traffic lights:

```
first light(green).
next light(amber) ← light(green).
next light(red) ← light(amber).
next light(green) ← light(red).
```

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 operators will be discussed shortly. A *temporal reference* is a sequence (possibly empty) of the above operators. We will often write $\textbf{next}^k$ to represent a sequence of k **next** operators. A *canonical temporal reference* is a temporal reference of the form **first** $\textbf{next}^k$. An *open temporal reference* is a temporal reference of the form $\textbf{next}^k$. A *temporal atom* is an atom preceded by either a canonical or an open temporal reference.

In this paper we consider an extension of Chronolog that allows negation in the bodies of the rules of a program (and any reference to Chronolog in the rest of the paper will concern this particular extension).

A *temporal clause* in Chronolog is a formula of the form:

$$\mathbf{H} \leftarrow \mathbf{A}_1, \ldots, \mathbf{A}_k, \neg\mathbf{B}_1, \ldots, \neg\mathbf{B}_m.$$

where $\mathbf{H}, \mathbf{A}_1, \ldots, \mathbf{A}_k, \mathbf{B}_1, \ldots, \mathbf{B}_m$ are temporal atoms and $k, m \geq 0$. If $k = m = 0$, the clause is said to be a *unit temporal clause.* A *temporal program* is a finite set of *temporal clauses.*

Chronolog is based on the relatively simple *temporal logic* TL, which uses 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* [OW92a, OW92b]:

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 $[D^n \to D]$; and for each n-ary predicate symbol, an element of $[\mathcal{N} \to 2^{D^n}]$.

In the following definition, the satisfaction relation $\models$ is defined in terms of temporal interpretations. $\models_{I,t} \mathbf{A}$ 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 $\mathbf{f}(\mathbf{e}_0, \ldots, \mathbf{e}_{n-1})$ is a term, then $I(\mathbf{f}(\mathbf{e}_0, \ldots, \mathbf{e}_{n-1})) = I(\mathbf{f})(I(\mathbf{e}_0), \ldots, I(\mathbf{e}_{n-1}))$.
2. For any n-ary predicate symbol $\mathbf{p}$ and terms $\mathbf{e}_0, \ldots, \mathbf{e}_{n-1}$,
   $\models_{I,t} \mathbf{p}(\mathbf{e}_0, \ldots, \mathbf{e}_{n-1})$ *iff* $\langle I(\mathbf{e}_0), \ldots, I(\mathbf{e}_{n-1})\rangle \in I(\mathbf{p})(t)$
3. $\models_{I,t} \neg\mathbf{A}$ *iff it is not the case that* $\models_{I,t} \mathbf{A}$
4. $\models_{I,t} \mathbf{A} \wedge \mathbf{B}$ *iff* $\models_{I,t} \mathbf{A}$ *and* $\models_{I,t} \mathbf{B}$
5. $\models_{I,t} \mathbf{A} \vee \mathbf{B}$ *iff* $\models_{I,t} \mathbf{A}$ *or* $\models_{I,t} \mathbf{B}$
6. $\models_{I,t} (\forall\mathbf{x})\mathbf{A}$ *iff* $\models_{I[d/\mathbf{x}],t} \mathbf{A}$ *for all* $d \in D$*, where the interpretation* $I[d/\mathbf{x}]$ *is the same as* I *except that the variable* $\mathbf{x}$ *is assigned the value* d*.*
7. $\models_{I,t}$ **first A** *iff* $\models_{I,0} \mathbf{A}$
8. $\models_{I,t}$ **next A** *iff* $\models_{I,t+1} \mathbf{A}$

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 $\models_I \mathbf{A}$) 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 $U_{\mathbf{P}}$ 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* $B_{\mathbf{P}}$ of **P**, consisting of all *ground canonical temporal atoms* whose predicate symbols appear in **P** and whose arguments are terms in the Herbrand universe $U_{\mathbf{P}}$ of **P**. In particular, given a subset H of $B_{\mathbf{P}}$, we can define a Herbrand interpretation I by the following:

$$\langle \mathrm{e}_0, \ldots, \mathrm{e}_{n-1} \rangle \in I(\mathbf{p})(t) \ \textit{ iff } \ \mathbf{first\ next}^t\ \mathbf{p}(\mathrm{e}_0, \ldots, \mathrm{e}_{n-1}) \in H$$

A *temporal Herbrand model* is a temporal Herbrand interpretation, which is a model of the program. In the rest of the paper when we refer to a "model of a program" we always mean a temporal Herbrand model.

3 The Classical Counterpart of a Temporal Program

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* $\mathbf{P}^*$ 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 $\mathbf{first\ next}^k\ \mathbf{p}(\mathrm{e}_0, \ldots, \mathrm{e}_{n-1})$ in **P** by the classical atom $\mathbf{p}(\mathbf{s}^k(\mathbf{0}), \mathrm{e}_0, \ldots, \mathrm{e}_{n-1})$.
- Replace every open temporal atom of the form $\mathbf{next}^k\ \mathbf{p}(\mathrm{e}_0, \ldots, \mathrm{e}_{n-1})$ by the classical atom $\mathbf{p}(\mathbf{s}^k(\mathbf{T}), \mathrm{e}_0, \ldots, \mathrm{e}_{n-1})$.
- In the body of every clause that contains at least one open temporal atom, add the atom **nat**(**T**) (whose purpose is to restrict the time parameter to obtaining only natural number values). Also, add to the program the axiomatization of **nat**.

In the following, we will also refer to the programs that result from the above transformation as *time-classical* logic programs. Moreover, terms of the form $\mathbf{s}^k(\mathbf{0})$ will be called (ground) *time-terms.*

The above transformation algorithm is illustrated by the following example:

Example 3.1. Consider the following temporal logic program:

```
first next p(0).
next next p(X) ← ¬ next p(X).
```

The transformation described above results in:

```
p(s(0),0).
p(s(s(T)),X) ← ¬ p(s(T),X), nat(T).
nat(0).
nat(s(X)) ← nat(X).
```

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 **nat** that it contains are all the atoms of the set $Nat = \{\mathbf{nat}(\mathbf{s}^k(\mathbf{0})) \mid k \geq 0\}$.
2. All the other atoms that it contains are of the form $\mathbf{p}(\mathbf{s}^k(\mathbf{0}), \mathbf{e}_0, \ldots, \mathbf{e}_{n-1})$, where $k \geq 0$.

The following theorem can then be easily established:

Theorem 3.1 *Let* $\mathbf{P}$ *be a temporal logic program and* $\mathbf{P}^*$ *be its classical counterpart. Then, there is a one-to-one correspondence between the temporal Herbrand models of* $\mathbf{P}$ *and the normal Herbrand models of* $\mathbf{P}^*$.

Proof: Given a temporal Herbrand model I of $\mathbf{P}$, obtain a classical interpretation I^* by replacing every canonical ground temporal atom $\mathbf{first\ next}^k\ \mathbf{p}(\mathbf{e}_0, \ldots, \mathbf{e}_{n-1})$ in I by the classical atom $\mathbf{p}(\mathbf{s}^k(\mathbf{0}), \mathbf{e}_0, \ldots, \mathbf{e}_{n-1})$. It can be easily shown that $I^* \cup \{\mathbf{nat}(\mathbf{s}^k(\mathbf{0})) \mid k \geq 0\}$ is a model of $\mathbf{P}^*$. Similarly, when given a normal Herbrand model of the classical program $\mathbf{P}^*$ one can easily obtain a temporal Herbrand model of $\mathbf{P}$. ■

We see therefore that from a model theory point of view, a temporal logic program is closely related to its classical counterpart. 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 corresponds to the implicit time parameter of the initial temporal program. In this section, we show that the special structure of these programs allows us to define a syntactic test (the *cycle sum test*), which when passed, ensures that the time-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 their intended meaning. 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 to the classical logic programs that resulted from the transformation of section 3.

The following definitions are needed:

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

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

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

Intuitively, the value of *dif*($\mathbf{H}, \mathbf{A}$) expresses how far (ie. how many time-points) in the worst case the head $\mathbf{H}$ of a clause leads the atom $\mathbf{A}$ in the body of the clause. In particular, the value $-\infty$ used in the last case of the above definition, signifies that in this case it is not possible to determine a finite integer value by which the head leads the atom in the body in the worst case (because the head refers to a specific moment in time while the atom in the body has an open temporal reference).

Example 4.1. Consider the following temporal logic program:

```
next p(X) ← next next q(X), ¬ first r(X).
first next next p(X) ← next q(X).
```

The corresponding time-classical logic program is (we omit the clauses for **nat**):

```
p(s(T),X) ← q(s(s(T)),X), ¬ r(0,X),nat(T).
p(s(s(0)),X) ← q(s(T),X),nat(T).
```

Then, using the definition of *dif* we have:

$$\begin{array}{lcl} dif(\texttt{p(s(T),X)},\texttt{q(s(s(T)),X)}) & = & -1 \\ dif(\texttt{p(s(T),X)},\texttt{r(0,X)}) & = & 1 \\ dif(\texttt{p(s(s(0)),X)}, \texttt{q(s(T),X)}) & = & -\infty \end{array}$$

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

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

We can now state the cycle sum test:

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

Example 4.2. The lights program of section 2 can be also coded using negation in the bodies of the clauses:

```
first light(green).
next light(amber) ← ¬ light(red), ¬ light(amber).
next light(red) ← ¬ light(green), ¬ light(red).
next light(green) ← ¬ light(red), ¬ light(green).
```

The above program can easily be shown that passes the cycle sum test.

In the following section we show that every time-classical logic program passing the cycle sum test is locally stratified and therefore has a well-defined meaning.

There is an important difference between the test described above and the usual stratification tests for classical logic programming. Given a classical logic program with negation, a stratification algorithm usually constructs a program dependency graph [PP90] and the edges of the graph are labeled as *positive* or *negative* depending on whether they "connect" the head of a program clause with a positive or negative literal in the body of the clause. The cycle sum test as defined above, does not take into consideration positive and negative edges, and it is therefore possible that certain positive programs will not be directly acceptable by the test. For example, consider the following:

```
first p(a).
p(X) ← next p(X).
```

This program is not directly acceptable by the test (the cycle sum graph contains a cycle with weight -1) but it obviously has a well-defined meaning under the standard semantics of temporal logic programming.[1] The obvious solution is to define as acceptable all the programs that are either positive or pass the cycle sum test.

We believe however that a refinement of the test that would additionally take into account positive and negative edges is possible although not straightforward (this is further discussed in the concluding section). In this article we restrict attention to the formulation of the test given by Definition 4.4.

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 $\mathbf{P}$ be a classical logic program (with negation). The *dependency graph* $DG_{\mathbf{P}}$ of $\mathbf{P}$ is a graph whose vertex set is the Herbrand base $B_{\mathbf{P}}$ of $\mathbf{P}$ and whose edges are determined as follows: if $\mathbf{A}$ and $\mathbf{B}$ are two atoms in $B_{\mathbf{P}}$, there exists a directed edge from $\mathbf{A}$ to $\mathbf{B}$ if and only if there exists an instance of a clause in $\mathbf{P}$ whose head is $\mathbf{A}$ and one of whose premises is either $\mathbf{B}$ or $\neg\mathbf{B}$. In the latter case, the edge is called *negative*.

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

The following theorem from [PP90] actually defines the notion of local stratification in terms of the priority relation $<$:

Theorem 5.1 *[PP90] A logic program* $\mathbf{P}$ *is locally stratified if and only if every increasing sequence of ground atoms under* $<$ *is finite.*

[1]However, notice that programs such as the above in fact contain some form of deadlock [Wad81] and are in a sense against the spirit of temporal logic programming which views predicates as infinite *streams* [Wad88].

Notice that given a time-classical logic programs $\mathbf{P}^*$, there is a close relationship between the structure of the cycle sum graph $CG_{\mathbf{P}^*}$ and the dependency graph $DG_{\mathbf{P}^*}$. The following (graph theoretic) definition is needed before stating Lemma 5.1 below:

Definition 5.3. [Har72] Let G be a directed graph. A *walk* in G is a sequence of vertices and edges, $v_0, e_1, v_1, \ldots, e_n, v_n$ in which each edge e_i connects v_{i-1} with v_i. A *closed walk* has the same first and last vertices.

Lemma 5.1 *Let $\mathbf{P}^*$ be a time-classical logic program. Then, a directed path in $DG_{\mathbf{P}^*}$ corresponds to a walk in $CG_{\mathbf{P}^*}$.*

Proof: Let $v_0, e_1, v_1, \ldots, e_n, v_n$ be a directed path in $DG_{\mathbf{P}^*}$. The graph $DG_{\mathbf{P}^*}$ is constructed in a similar way as the graph $CG_{\mathbf{P}^*}$ by looking at the structure of the clauses of $\mathbf{P}^*$. The main difference is that the vertices of $DG_{\mathbf{P}^*}$ are terms of the Herbrand base of $\mathbf{P}^*$ while the vertices of $CG_{\mathbf{P}^*}$ are predicate names appearing in $\mathbf{P}^*$. Therefore, there exists a corresponding sequence of vertices and edges $v'_0, e'_1, v'_1, \ldots, e'_n, v'_n$ in $CG_{\mathbf{P}^*}$, such that each v'_i refers to the same predicate name $\mathbf{p}_i$ as v_i. The only difference is that while all of the v_i's are distinct as they are different terms of the Herbrand base of $\mathbf{P}^*$, the v'_i's might not be all distinct because they are predicate names (and not terms). ■

The lemma given below will be used in the following discussion:

Lemma 5.2 *Let W 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 an edge 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.*

Proof: By induction on the length of the walk W. ■

The following theorem demonstrates that programs passing the cycle sum test are locally stratified. The proof uses Lemmas 5.1 and 5.2.

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

Proof: Assume that $\mathbf{P}^*$ is not locally stratified. This means that there exists an infinite increasing sequence $\mathbf{A}_1 < \mathbf{A}_2 < \cdots$ of atoms of the Herbrand base of $\mathbf{P}^*$. 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 $\mathbf{C}$. These atoms form an infinite subsequence $\mathbf{B}_1 < \mathbf{B}_2 < \cdots$ of the initial sequence.

Consider now two atoms $\mathbf{B}_i$ and $\mathbf{B}_{i+1}$ from the new sequence. The first argument of $\mathbf{B}_i$ is of the form $\mathbf{s}^k(\mathbf{e})$, where $k \geq 0$ and $\mathbf{e}$ is either $\mathbf{0}$ or a non time-term. Moreover, by Definition 5.2, there exists a directed path from $\mathbf{B}_i$ to $\mathbf{B}_{i+1}$ in the dependency graph of $\mathbf{P}^*$. By Lemma 5.1, this path corresponds to a walk in the cycle sum graph of $\mathbf{P}^*$, which is in fact a closed walk because $\mathbf{B}_i$ and $\mathbf{B}_{i+1}$ are atoms having the same predicate name. A closed walk can be decomposed into a number of cycles (Lemma 5.2) and therefore 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 $\mathbf{B}_i$ leads the first argument of $\mathbf{B}_{i+1}$ by a positive amount (i.e., the first argument of $\mathbf{B}_{i+1}$ is either of the form $\mathbf{s}^m(\mathbf{e})$ or $\mathbf{s}^m(\mathbf{0})$, with $m < k$). Consequently, the first arguments of the members of the sequence $\mathbf{B}$ decrease in complexity and therefore the sequence can not be infinite. This is a contradiction, which implies that program $\mathbf{P}^*$ is locally stratified. ∎

Theorem 5.3 *Let* $\mathbf{P}^*$ *be a time-classical logic program that passes the cycle sum test. Then* $\mathbf{P}^*$ *has a unique perfect model which is also normal.*

Proof: By Theorem 5.2, $\mathbf{P}^*$ is locally stratified, which means that it has a unique perfect model M^* [Prz88]. This model obviously contains the set $Nat = \{\mathbf{nat}(\mathbf{s}^k(\mathbf{0})) \mid k \geq 0\}$ due to the two clauses in $\mathbf{P}^*$ 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 only contains atoms of the form $\mathbf{p}(\mathbf{s}^k(\mathbf{0}), \mathbf{e}_0, \ldots, \mathbf{e}_{n-1})$. 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 $\mathbf{p}(\mathbf{s}^k(\mathbf{0}), \mathbf{e}_0, \ldots, \mathbf{e}_{n-1})$. We have $N^* \subset M^*$. It can be easily shown that every ground instance of a clause in $\mathbf{P}^*$ is true under N^*. Therefore, N^* is also a model of $\mathbf{P}^*$, which contradicts the fact that perfect models are minimal models [Prz88]. ∎

From the above discussion, we conclude that given a temporal logic program $\mathbf{P}$, if its classical counterpart $\mathbf{P}^*$ passes the cycle sum test, then $\mathbf{P}^*$ 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 $\mathbf{P}$, which we take as the intended meaning of $\mathbf{P}$.

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 are guaranteed to have a well-defined meaning. In particular, the test represents an application of the notion of local stratification in the context of temporal logic programming.

Obviously, not all locally stratified programs can be detected and in fact, as noted in section 4, certain positive programs are not directly acceptable by the test. The obvious solution is to define as acceptable all the programs that are either positive or pass the cycle sum test. We conjecture that an extension of the test can be devised that can handle a significantly broader class of temporal logic programs with negation and which directly accepts positive programs. Such an extension would probably rely on a generalized cycle sum graph, which apart from the integer weights would also contain indications of whether edges are positive or negative. The notion of "sum" across cycles would then have to be generalized to take into account the new indications.

Another interesting topic for further research would be the consideration of other temporal logic programming languages, that use an extended set of temporal operators or 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.

References

[AB94] K. Apt and R. Bol. Logic Programming and Negation: A Survey. *Journal of Logic Programming*, 19,20:9–71, 1994.

[ABW88] K.R. Apt, H.A. Blair, and A. Walker. Towards a Theory of Declarative Knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann, Los Altos, CA, 1988.

[Har72] F. Harary. *Graph Theory*. Addison-Wesley, 1972.

[Llo87] J. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987.

[Mat95] S.G. Matthews. An Extensional Treatment of Lazy Dataflow Deadlock. *Theoretical Computer Science*, 151:195–205, 1995.

[OW92a] M. A. Orgun and W. W. Wadge. Theory and practice of temporal logic programming. In L. Farinas del Cerro and M. Penttonen, editors, *Intensional Logics for Programming*, pages 23–50. Oxford University Press, 1992.

[OW92b] M. A. Orgun and W. W. Wadge. Towards a unified theory of intensional logic programming. *The Journal of Logic Programming*, 13(4):113–145, August 1992.

[PP90] H. Przymusinska and T. Przymusinski. Semantic Issues in Deductive Databases and Logic Programs. In R. Banerji, editor, *Formal Techniques in Artificial Intelligence*, pages 321–367. North Holland, 1990.

[Prz88] T. Przymusinski. On the Declarative Semantics of Deductive Databases and Logic Programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Morgan Kaufmann, Los Altos, CA, 1988.

[RGP97] P. Rondogiannis, M. Gergatsoulis, and T. Panayiotopoulos. *Cactus*: A branching-time logic programming language. In *Proc. of the First International Joint Conference on Qualitative and Quantitative Practical Reasoning, ECSQARU-FAPR'97, Bad Honnef, Germany*, Lecture Notes in Artificial Intelligence (LNAI) 1244, pages 511–524. Springer, June 1997.

[WA85] W. W. Wadge and E. A. Ashcroft. *Lucid, the Dataflow Programming Language*. Academic Press, 1985.

[Wad81] W. W. Wadge. An Extensional Treatment of Dataflow Deadlock. *Theoretical Computer Science*, 13:3–15, 1981.

[Wad88] W. W. Wadge. Tense logic programming: A respectable alternative. In *Proc. of the 1988 International Symposium on Lucid and Intensional Programming*, pages 26–32, 1988.