A Stratification Test for Temporal Logic Programs*

Christos Nomikos[1†], **Panos Rondogiannis**[2], **Manolis Gergatsoulis**[3]

[1]Department of Computer Science, University of Ioannina,
P.O. Box 1186, 45 110 Ioannina, Greece
e_mail: cnomikos@cs.uoi.gr
[2]Department of Informatics & Telecommunications, University of Athens,
Panepistimiopolis, 15 784 Athens, Greece
e_mail: prondo@di.uoa.gr
[3]Institute of Informatics & Telecommunications, N.C.S.R. 'Demokritos',
153 10 Aghia Paraskevi Attikis, Greece
e_mail: manolis@iit.demokritos.gr

Abstract

In this paper, we propose a new stratification test for linear-time temporal logic programs. More specifically, we extend the *cycle-sum test* [Ron01] to take into account negatively signed edges in the cycle-sum graph. In this way we broaden the class of temporal logic programs which are acceptable by the test. Moreover, we demonstrate that the cycle-sum test can be used to decide local statification for a broad class of temporal logic programs.

Keywords: Temporal Logic Programming, Negation, Stratification, Programming Languages, Deductive Databases.

1 Introduction

One of the most interesting and profound research topics in logic programming, is the study of negation [PP90, AB94]. However, most of the research results in this area concern classical logic programming languages. Recent research [Ron01] undertakes the study of stratified negation in temporal logic programming. In this paper, we extend the results obtained in [Ron01] to cover a broader class of temporal logic programs.

In [Ron01] a test has been proposed for identifying linear-time temporal logic programs that are locally stratified. A locally stratified program is guaranteed to have a unique intended model, which is called the perfect model of this program. The test has been named *cycle-sum test*, as is has been inspired by the homonymous test proposed by W. W. Wadge [Wad81] for detecting deadlocks in the context of dataflow programming. The test of [Ron01] constructs the so-called *cycle-sum graph*, a labeled directed graph whose nodes are program predicates. In this graph, there is an edge from a predicate **p** to a predicate **q** whenever there is a program clause whose head predicate is **p** and whose

*This work has been partially supported by the Greek General Secretariat of Research and Technology under the project ΠΕΝΕΔ'99 (contract no 99ΕΔ265).

†This work has been done while the first author was at N.C.S.R. 'Demokritos'.

body contains an atom with predicate **q**. The label of an edge indicates the temporal difference between the two corresponding temporal atoms. A linear-time temporal logic program passes the test if all the cycles in the graph have positive sums of weights. However, the construction of this graph does not take into consideration the negated atoms of the source program. It is therefore possible (as pointed out in [Ron01]) that certain positive programs will not pass the test. For example, consider the program:

```
first p(a).
p(X) ← next p(X).
```

This program will not pass the test since its 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. In this paper we remedy the above deficiency by augmenting the cycle-sum graph with additional information and by devising an extended cycle-sum test.

The rest of the paper is organized as follows: section 2 contains background material. Section 3 presents an extended cycle-sum test based on a more sophisticated cycle-sum graph. Section 4 demonstrates the correctness of the new test and section 5 concludes the paper by giving pointers to future work.

2 Preliminaries

The temporal logic programming language we consider here is Chronolog [Wad88, OW92a, OW92b]. The syntax of Chronolog programs is an extension of the syntax of classical logic programs [Llo87] with two temporal operators, namely **first** and **next**. A *temporal reference* is a sequence (possibly empty) of the above operators. We will often write $\mathbf{next}^k$ to represent a sequence of k **next** operators. A *canonical temporal reference* is a temporal reference of the form $\mathbf{first\ next}^k$, where $k \geq 0$. An *open temporal reference* is a temporal reference of the form $\mathbf{next}^k$. 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:

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

where $\mathbf{H}, \mathbf{A}_1, \ldots, \mathbf{A}_n, \mathbf{B}_1, \ldots, \mathbf{B}_m$ are temporal atoms and $n, m \geq 0$. If $n = m = 0$, 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 and discrete 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. $\models_{I,t}$ **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 **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. $\models_{I,t} (\forall \mathbf{x})\mathbf{A}$ *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* **x** *is assigned the element* d.
6. $\models_{I,t}$ **first A** *iff* $\models_{I,0} \mathbf{A}$
7. $\models_{I,t}$ **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:

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

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 Herbrand model.

3 The Extended Cycle Sum Test

In this section we present an extension of the cycle-sum test [Ron01]. For this purpose, the following definitions are needed:

Definition 3.1. Let $\mathbf{A}$ be a temporal atom. Then, $time(\mathbf{A})$ is the temporal reference of $\mathbf{A}$.

Definition 3.2. Let $\mathbf{P}$ be a Chronolog 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 in the body of $\mathbf{C}$. The *temporal difference dif* between $\mathbf{H}$ and $\mathbf{A}$ is defined as follows:

$$dif(\mathbf{H},\mathbf{A}) \quad = \quad \begin{cases} k-m, & \text{if } time(\mathbf{H}) = \textbf{first next}^k \text{ and } time(\mathbf{A}) = \textbf{first next}^m \\ k-m, & \text{if } time(\mathbf{H}) = \textbf{next}^k \text{ and } time(\mathbf{A}) = \textbf{next}^m \\ k-m, & \text{if } time(\mathbf{H}) = \textbf{next}^k \text{ and } time(\mathbf{A}) = \textbf{first next}^m \\ -\infty, & \text{if } time(\mathbf{H}) = \textbf{first next}^k \text{ and } time(\mathbf{A}) = \textbf{next}^m \end{cases}$$

Intuitively, the value of $dif(\mathbf{H},\mathbf{A})$ expresses a lower bound on how far (ie. how many time-points) 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). The following definition formalizes the notion of the *extended cycle-sum graph* of a given Chronolog program.

Definition 3.3. Let $\mathbf{P}$ be a given Chronolog program. The *extended cycle-sum graph* of $\mathbf{P}$ is a directed weighted multi-graph with self-loops $CG_{\mathbf{P}} = (V, E)$. 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 $(\mathbf{p}, \mathbf{q}, l)$, where $\mathbf{p}, \mathbf{q} \in V$ and $l \in (\mathcal{Z} \cup \{-\infty\}) \times \{+, -\}$. An edge $(\mathbf{p}, \mathbf{q}, \langle w, s\rangle)$ belongs to E if there exists a clause in $\mathbf{P}$ 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 $\mathbf{p}$ and the predicate symbol of $\mathbf{A}$ is $\mathbf{q}$; $w = dif(\mathbf{H},\mathbf{A})$; $s =$ '$-$' if $\mathbf{A}$ occurs negatively in the clause body and $s =$ '$+$' otherwise.

We can now state the extended cycle-sum test:

Definition 3.4. A Chronolog program $\mathbf{P}$ passes the extended cycle sum test if in any strongly connected component containing a negatively signed edge the following conditions both hold:

1. The sum of weights across every cycle in $CG_{\mathbf{P}}$ is non-negative.
2. Every cycle which has a zero sum of weights does not contain a negatively signed edge.

There are some important aspects which differentiate the extended cycle-sum test from the initial one:

- The test is defined directly on Chronolog programs instead of the so-called time-classical ones [Ron01]. Time-classical programs correspond to a translation of Chronolog programs into equivalent classical logic programs.

- The extended cycle-sum graph contains additional information related to the occurrences of negative atoms in the bodies of the source program clauses.
- The extended cycle-sum test only examines those strongly connected components of the cycle-sum graph which contain negatively signed edges.

In the next section we provide a theoretical justification of the extended cycle-sum test.

4 Correctness of the Extended Cycle Sum Test

In the following, we demonstrate that a Chronolog program accepted by the extended cycle-sum test is locally stratified. The following definitions are required (notice that these definitions are the temporal analogs of the corresponding ones for classical logic programs [PP90]):

Definition 4.1. Let $\mathbf{P}$ be a Chronolog program. The *dependency graph* $DG_{\mathbf{P}}$ of $\mathbf{P}$ is a graph whose vertex set is the temporal 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 a canonical temporal 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 4.2. Let $\mathbf{P}$ be a Chronolog program (with negation). For any two canonical temporal ground atoms $\mathbf{A}$ and $\mathbf{B}$ in $B_{\mathbf{P}}$ we write $\mathbf{A} < \mathbf{B}$ if there exists a directed walk 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 canonical ground atoms.

The notions of local stratification and perfect model for Chronolog programs are straightforward extensions of the ones for classical logic programs [Prz88], and it can be proved that any locally stratified Chronolog program has a unique perfect model. Moreover, the following theorem provides an alternative definition for local stratification in Chronolog (actually it is a temporal analog of a theorem presented in [PP90]):

Theorem 4.1 *A Chronolog program* $\mathbf{P}$ *is locally stratified if and only if every increasing sequence of canonical temporal ground atoms under* $<$ *is finite.*

The following theorem demonstrates that programs passing the cycle sum test are locally stratified.

Theorem 4.2 *Let* $\mathbf{P}$ *be a Chronolog program that passes the extended cycle sum test. Then* $\mathbf{P}$ *is locally stratified.*

Proof: (Sketch) Assume that $\mathbf{P}$ is not locally stratified. By Theorem 4.1, this means that there exists an infinite increasing sequence $\mathbf{A}_1 < \mathbf{A}_2 < \cdots$ of canonical atoms of the temporal Herbrand base of $\mathbf{P}$. Since the program contains a finite set of predicate names, there exists an infinite subsequence of the form $\mathbf{first\ next}^{k_1}\ \mathbf{p}(\cdots) < \mathbf{first\ next}^{k_2}\ \mathbf{p}(\cdots) < \cdots$, i.e., a subsequence in which all atoms have the same predicate. There must be

some i such that $k_{i+1} \geq k_i$. It is easy to check that this implies the existence of a closed walk in the cycle-sum graph which contains a negatively signed edge and has non-positive sum of weights (equal to $k_i - k_{i+1}$). Moreover this closed walk is entirely contained in a single strongly connected component. This closed walk can be decomposed into a set of simple cycles. There are two cases:

- Either there exists a cycle with negative sum of weights, or
- all cycles of the walk have zero sum of weights (and at least one of these cycles contains a negatively signed edge).

In both cases, the cycle-sum test will fail for **P**. ∎

Corollary 4.1 *Let* **P** *be a Chronolog program that passes the extended cycle sum test. Then* **P** *has a unique perfect model.*

Theorem 4.3 *Let* **P** *be a propositional Chronolog program in which all non-unit clauses contain only open temporal atoms. Then,* **P** *passes the extended cycle sum test if and only if* **P** *is locally stratified.*

Proof: (Sketch) The one direction is theorem 4.2. For the other direction assume that **P** fails to pass the extended cycle-sum test. Then, it is easy to prove that we can always construct a closed walk with non-positive sum of weights that contains a negatively signed edge, since a non-positive cycle and a negatively signed edge appear in the same strongly connected component. Let **p** be a propositional atom that corresponds to a node in this closed walk. Then, for some sufficiently large integer k, there exists a walk in the dependency graph from **first next**^{k} **p** to **first next**$^{k+d}$ **p** with $d \geq 0$ that contains a negative edge. This can be obtained by following the edges of the closed walk in the extended cycle some graph. Thus **first next**^{k} **p** $<$ **first next**$^{k+d}$ **p**. Repeating the same construction one can create an infinitely increasing sequence of canonical propositional temporal atoms. Therefore, **P** is not locally stratified. ∎

Corollary 4.2 *The local stratification problem for Chronolog programs without function symbols, in which all non-unit clauses contain only open temporal atoms, is decidable.*

Notice that the local stratification problem for the general logic programs (and thus for Chronolog programs) is undecidable [CB94]. The above corollary does not mean that we can decide local stratification of a Chronolog program without function symbols by directly applying the extended cycle-sum test to it. The decision method in that case requires to apply the test to the program obtained by taking the ground instantiation of the clauses of the initial program and replacing each ground atom with a propositional symbol.

5 Future Work

In this paper we have extended 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.

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 [RGP98] 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. R. Apt and B. N. Bol. Logic programming and negation : A survey. *The Journal of Logic Programming*, 19/20:9–71, May/July 1994.

[CB94] P. Cholak and H. A. Blair. The complexity of Local Stratification. *Fundamenta Informaticae*, 21(4):333–344, 1994.

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

[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):413–440, August 1992.

[PP90] H. Przymusinska and T. Przymusinski. Semantic issues in deductive databases and logic programming. In R. Banerji, editor, *Formal Techniques in Arrtificial Intelligence*, pages 321–367. North Holland, 1990.

[Prz88] T. C. Przymusinski. On the declarative semantics of deductive databases and logic programs. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Morgan Kaufmann Publishers,Inc., 1988.

[RGP98] P. Rondogiannis, M. Gergatsoulis, and T. Panayiotopoulos. Branching-time logic programming: The language Cactus and its applications. *Computer Languages*, 24(3):155–178, October 1998.

[Ron01] Panos Rondogiannis. Stratified negation in temporal logic programming and the cycle-sum test. *Theoretical Computer Science*, 254:663–676, 2001.

[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.