October 1988

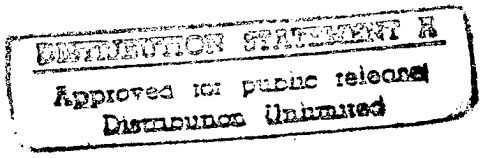
Report No. STAN-CS-88-1232

PB96-148259

Temporal Logic Programming is Complete and Expressive

by

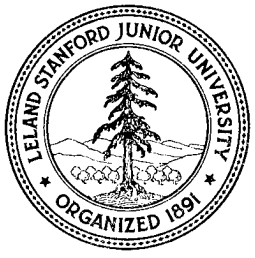
Marianne Baudinet



Department of Computer Science

Stanford University

Stanford, California 94305



DARPA INSERT SHEET

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188
Exp. Date: Jun 30, 1986

1a REPORT SECURITY CLASSIFICATION unclassified	1b. RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release: Distribution Unlimited
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) STAN-CS-88-1232	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Computer Science Department 6b OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Stanford University Stanford, CA 94305	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION DARPA 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) 1400 Wilson Blvd. Arlington, VA 22209	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11 TITLE (Include Security Classification)
Temporal Logic Programming is Complete and Expressive

12 PERSONAL AUTHOR(S)
Marianne Baudinet

13a TYPE OF REPORT	13b TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
		October 1988	14

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

This paper addresses semantic and expressiveness issues for temporal logic programming and in particular for the TEMPLOG language proposed by Abadi and Manna. Two equivalent formulations of TEMPLOG's declarative semantics are given: in terms of a minimal Herbrand model and in terms of a least fixpoint. By relating these semantics to TEMPLOG's operational semantics, we prove the completeness of the resolution proof system underlying TEMPLOG's execution mechanism. To study TEMPLOG's expressiveness, we consider its propositional version. We show how propositional TEMPLOG programs can be translated into a temporal fixpoint calculus and prove that they can express essentially all regular properties of sequences.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION	
22a NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c OFFICE SYMBOL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete. — SECURITY CLASSIFICATION OF THIS PAGE

Temporal Logic Programming is Complete and Expressive*

Marianne Baudinet
Computer Science Department
Stanford University

October 1988

Abstract

This paper addresses semantic and expressiveness issues for temporal logic programming and in particular for the TEMPLOG language proposed by Abadi and Manna. Two equivalent formulations of TEMPLOG's declarative semantics are given: in terms of a minimal Herbrand model and in terms of a least fixpoint. By relating these semantics to TEMPLOG's operational semantics, we prove the completeness of the resolution proof system underlying TEMPLOG's execution mechanism. To study TEMPLOG's expressiveness, we consider its propositional version. We show how propositional TEMPLOG programs can be translated into a temporal fixpoint calculus and prove that they can express essentially all regular properties of sequences.

1 Introduction

Temporal logic is more and more widely acknowledged as a useful formalism for program specification and verification. It has been used quite extensively for concurrent programs and digital hardware, but it is also applicable whenever it is necessary to specify or describe a sequence of states or events, such as in robot planning or historical databases. Recently, the idea has emerged that one could more easily use the expressive power of temporal logic if it could be made directly executable, for instance as is done with first-order logic in PROLOG. This has lead to the definition of a number of programming languages based on temporal logic ([FKTMo86], [Mos86], [AM87], [Gab87], [Wad88], [OW88a], [Sak]).

The earliest of these languages, the TEMPURA language of [Mos84, Mos86] is based on a subset of interval temporal logic whose formulas can be interpreted as traditional imperative programs. In logical terms, executing a TEMPURA formula (program) amounts to building a model for that formula. The TOKIO language of [FKTMo86] is an extension of logic programming, but resembles TEMPURA in the way it treats its temporal constructs. The other temporal programming languages ([Aba87], [AM87], [Gab86, Gab87], [Wad85, Wad88], [OW88a], [Sak]) are based on the logic programming paradigm and view an execution of a program as a refutation proof.

For this last class of languages, important semantical questions are left unanswered. First among these is the relation between the operational and the logical semantics of the languages. Indeed, in classical logic programming, the operational and the logical semantics coincide because of the completeness of SLD-resolution ([Hil74], [Cla79], [AvE82]). Unfortunately, first-order temporal logic is inherently incomplete ([Aba87]). So, one could very well expect that the operational and the logical semantics of temporal programming languages do not and even cannot coincide. Another unanswered question is the expressiveness of these languages. Classical Horn-clause logic programming, though in some respects weaker than first-order logic, is able to express predicates that are not first-order, e.g., the transitive closure of a relation ([CH85]). Similar issues appear in temporal logic programming languages. For instance, what temporal properties are they actually capable of expressing?

*This research was supported by the National Science Foundation under Grants DCR-84-13230, DCR-86-11272 and CCR-87-14170, by the Defense Advanced Research Projects Agency under Contract N00039-84-C-0211, and by the United States Air Force Office of Scientific Research under Contract AFOSR-87-0149.

To appear in the *Proceedings of the Sixteenth ACM Symposium on Principles of Programming Languages*, Austin, Texas, January 1989.

Can they go beyond the expressiveness of temporal logic?

In this paper, we examine these questions for the TEMPLOG language of [AM87]. We capture both the declarative and the operational semantics of this language and prove that they coincide, hence proving that the fragment of temporal logic defined by TEMPLOG admits a complete proof system. Then, turning to the expressiveness issue, we relate the propositional version of TEMPLOG with the temporal fixpoint calculus μTL of [Var88]. We show that TEMPLOG corresponds to a fragment of μTL and we characterize its expressiveness in terms of finite automata.

TEMPLOG extends classical Horn logic programming to allow specific use of the temporal operators $\bigcirc$ (next), $\Box$ (always), and $\Diamond$ (eventually). Programs are sets of temporal clauses, and computations are proofs by refutation. The proof method used is a resolution method for temporal logic to which we refer as TSLD-resolution. We study the declarative (logical) semantics of TEMPLOG and define it both in model-theoretic terms and in fixpoint terms. For this, we define the notions of temporal Herbrand interpretation and of temporally ground formulas. We prove that the declarative semantics of a program is characterized by the minimal Herbrand model of the program. We then show how to associate with a TEMPLOG program a mapping whose least fixpoint coincides with the minimal Herbrand model of the program. This provides a fixpoint characterization of the declarative semantics. Next, we examine the TSLD-resolution method that is the basis of the operational semantics of TEMPLOG. We establish a correspondence between membership in the fixpoint of the mapping associated with programs and existence of a temporally ground resolution proof, thereby obtaining a type of ground-completeness theorem. From this result, we establish the completeness of TSLD-resolution using a temporal lifting lemma. Our proof techniques extend those that have been used for giving semantics to classical logic programming ([vEK76], [Cla79], [AvE82], [Llo84], [Apt87]).

The fixpoint semantics provides the necessary tool for studying the expressiveness of the language. To focus on the *temporal* expressiveness of the language, we study its expressiveness in the propositional case. Using our least fixpoint semantics it is quite easy to show that the expressiveness of TEMPLOG queries corresponds to a fragment of μTL allowing only least fixpoints applied to positive formulas. We further characterize the expressiveness of TEMPLOG and show that it essentially corresponds to the finite-word regular languages (more precisely to the ω-languages that are obtained by extending finite-word regular languages). TEMPLOG can thus express some properties that are not expressible in pure temporal logic as this last language cannot express all regular behaviors ([Wol83])[1]. On the other hand, there are formulas of temporal logic that are not expressible in TEMPLOG since expressing all of temporal logic in μTL can require using greastest fixpoints or the alternation of a greatest and a least fixpoint ([Par81]). In conclusion, if one is only interested in queries that can be can be checked on a finite prefix of the temporal sequence, as most likely would be the case for historical databases, the temporal expressiveness of TEMPLOG is perfectly adequate.

2 The Temporal Language

The TEMPLOG language of [AM87] is based on a clausal subset of first-order temporal logic with time considered discrete, linear and extending infinitely in the future but not in the past. First-order temporal logic extends the first-order predicate calculus by allowing the application of temporal operators to formulas. The operators of interest here are $\bigcirc$ (next), $\Box$ (always) and $\Diamond$ (eventually). Constant and function symbols are assumed to have a time-independent interpretation; they are said to be *rigid*. Predicate symbols can have an interpretation that varies with time, in which case they are said to be *flexible*. In fact, we assume that all the predicate symbols are flexible. (We discuss this assumption below.)

A formula of temporal logic is interpreted over a structure that we call a *temporal interpretation*. A temporal interpretation $\mathcal{I} = \langle D, \Sigma, \alpha, J \rangle$ consists of a domain D, a sequence of states (time instants) $\Sigma = \sigma_0, \sigma_1, \sigma_2, \ldots$ that is isomorphic to ω, an assignment α to variables, and an interpretation J. Since the constant and function symbols are rigid, the interpretation J assigns them a global meaning over the domain D, as in classical logic. But to predicate symbols, which are flexible, the interpretation assigns a relation over D for every state σ_i in the sequence Σ. If i is a natural number, $\mathcal{I}^{(i)}$ is the temporal interpretation obtained from $\mathcal{I}$ by taking the initial state to be σ_i and the sequence of states to be $\sigma_i, \sigma_{i+1}, \sigma_{i+2}, \ldots$. Given a language, that is, a collection of variables and of constant, function, and predicate symbols, the meaning of the terms and formulas of the language with respect to a temporal interpretation $\mathcal{I} = \langle D, \Sigma, \alpha, J \rangle$ is given by a function $\mathcal{T}_\mathcal{I}$ that provides the meaning of the terms, and the satisfac-

[1] Temporal logic is known to have the expressiveness of star-free ω-regular behaviors ([Tho81]) whereas the temporal fixpoint calculus corresponds to ω-regular behaviors ([BB86]).

tion relation $\models_{\mathcal{I}}$ for the formulas. They are defined inductively in the usual way. The function $\mathcal{T}_{\mathcal{I}}$ uses the assignment α to interpret the free variables and the interpretation J to interpret the constant and the function symbols. The most interesting cases of the definition of the satisfaction relation $\models_{\mathcal{I}}$ are given below. Let p be an ℓ-ary predicate symbol and let $t_1, \ldots, t_\ell$ be terms.

$\models_{\mathcal{I}} p(t_1, \ldots, t_\ell)$	iff	$J[p][\sigma_0](\mathcal{T}_{\mathcal{I}}[t_1], \ldots, \mathcal{T}_{\mathcal{I}}[t_\ell])$
$\models_{\mathcal{I}} \bigcirc F$	iff	$\models_{\mathcal{I}^{(1)}} F$
$\models_{\mathcal{I}} \Box F$	iff	for every i in ω: $\models_{\mathcal{I}^{(i)}} F$
$\models_{\mathcal{I}} \Diamond F$	iff	for some i in ω: $\models_{\mathcal{I}^{(i)}} F$

The notions of model, satisfiability, validity and logical consequence (denoted $F_1 \models F_2$) are defined in the usual way. Informally, we will say that F holds at time i when $\models_{\mathcal{I}^{(i)}} F$.

The TEMPLOG language is the subset of first-order temporal logic with the following syntax. Let A denote an atom and N denote a *next-atom*, that is, an atom preceded by a finite number of $\bigcirc$'s.

Body: $B ::= \varepsilon \mid A \mid B_1, B_2 \mid \bigcirc B \mid \Diamond B$
where ε denotes the empty body
Initial clause: $IC ::= N \leftarrow B \mid \Box N \leftarrow B$
Permanent clause: $PC ::= \Box(N \leftarrow B)$
Program clause: $C ::= IC \mid PC$
Goal clause: $G ::= \leftarrow B$

Throughout this paper, we use the symbol A to denote an atom, N for a next-atom, B for a body (empty or not), C for a clause, P for a program, and G for a goal clause. If F is a formula, we use the abbreviation $\bigcirc^i F$ to denote the formula consisting of F preceded by i occurrences of $\bigcirc$.

The free variables in program and goal clauses are implicitly universally quantified. A TEMPLOG program consists of a set of program clauses, that is, a conjunction of program clauses. In a clause, the consequent of the implication is called the *head* (the antecedent is the body). In a body, the comma stands for the conjunction operator (we use "," and "$\wedge$" interchangeably in the semantic development). A program clause that has an empty body is a *fact*. An empty body corresponds to "true". A goal clause can be seen as an initial clause with an empty head, the empty head corresponding to "false". Hence, a goal of the form $\leftarrow B$ with free variables $X_1, \ldots, X_n$ corresponds to the formula $(\forall X_1) \cdots (\forall X_n) \neg B$, that is, $\neg(\exists X_1) \cdots (\exists X_n) B$ (we use "$\leftarrow B$" and "$\neg B$" interchangeably in the semantic development).

Example 2.1 The following simple program P defines a predicate p such that $p(X)$ is true at time i for $X = s^{2i}(a)$. (We use capital letters for variables, and (strings of) lower-case letters for constant, function and predicate symbols.)

$$p(a) \leftarrow$$
$$\Box(\bigcirc p(s(s(X))) \leftarrow p(X)) \quad \blacksquare$$

Proof Method

Given a TEMPLOG program and a goal, a computation consists in trying to derive a contradiction using temporal resolution rules. When a refutation is obtained, it is usually for a certain instantiation of the variables in the goal, called an *answer substitution*. We assume some familiarity with the notions of substitution and unification (e.g. [Rob65], [LMM88], [MW89]). If θ and ϕ are substitutions, we denote their composition by $\theta \circ \phi$, and we write $\theta \succeq \phi$ to mean that θ is more general than ϕ, that is, there is a substitution λ such that $\theta \circ \lambda = \phi$.

We refer to the refutation procedure underlying TEMPLOG as *TSLD-resolution* (for *T*emporal *L*inear resolution for *D*efinite clauses[2] with a *S*election function) by analogy with the SLD-resolution procedure for classical logic programming ([AvE82]). Every step of a TSLD-derivation consists in resolving a *candidate next-atom* from the current goal with the head of a program clause, to produce a new goal. Before defining the notion of candidate next-atom precisely, we have to make a comment about the bodies of clauses. Syntactically distinct bodies may in fact be logically equivalent. So we assume that we are always dealing with the *canonical form* of the body, a body (or a goal) being in *canonical form* if its occurrences of $\bigcirc$ are pushed all the way inwards and if its next-atoms are in the scope of the least possible number of $\Diamond$'s. Each body has a unique equivalent canonical form (up to commutativity and associativity of the conjunction). A next-atom in a goal is said to be *candidate* if it is in the scope of at most one $\Diamond$ in the canonical form of the goal. There is at least one candidate next-atom in any nonempty goal. At every step of a derivation using the TSLD-resolution method, the *selection function* or *computation rule* selects from the current goal the candidate next-atom to be resolved in the next resolution step. This next-atom is referred to as the *selected next-atom*. The resolution rules used in TSLD-derivations are given in Table 1. For each rule, the selected candidate next-atom is $\bigcirc^i A$, and θ is the most-general unifier (*mgu*) of A and A'. The resolvent is also referred to as the *derived goal*.

Let P be a program, G a goal, and R a computation rule. A *TSLD-derivation* for $P \cup \{G\}$ *via* rule R is

[2] A definite clause is a Horn clause with a nonempty head.

	Cond.	Goal	Clause	Resolvent (Derived Goal)
1		$\leftarrow B_1, \bigcirc^i A, B_2$	$\bigcirc^i A' \leftarrow B'$	$\leftarrow (B_1, B', B_2)\theta$
2	$i \geq j$	$\leftarrow B_1, \bigcirc^i A, B_2$	$\Box \bigcirc^j A' \leftarrow B'$	$\leftarrow (B_1, B', B_2)\theta$
3	$i \geq j$	$\leftarrow B_1, \bigcirc^i A, B_2$	$\Box(\bigcirc^j A' \leftarrow B')$	$\leftarrow (B_1, \bigcirc^{i-j} B', B_2)\theta$
4	$j \geq i$	$\leftarrow B_1, \Diamond(B_2, \bigcirc^i A, B_3), B_4$	$\bigcirc^j A' \leftarrow B'$	$\leftarrow (B_1, \bigcirc^{j-i} B_2, B', \bigcirc^{j-i} B_3, B_4)\theta$
5	$j \geq i$	$\leftarrow B_1, \Diamond(B_2, \bigcirc^i A, B_3), B_4$	$\Box \bigcirc^j A' \leftarrow B'$	$\leftarrow (B_1, B', \Diamond(\bigcirc^{j-i} B_2, \bigcirc^{j-i} B_3), B_4)\theta$
6	$i \geq j$	$\leftarrow B_1, \Diamond(B_2, \bigcirc^i A, B_3), B_4$	$\Box \bigcirc^j A' \leftarrow B'$	$\leftarrow (B_1, B', \Diamond(B_2, B_3), B_4)\theta$
7	$j \geq i$	$\leftarrow B_1, \Diamond(B_2, \bigcirc^i A, B_3), B_4$	$\Box(\bigcirc^j A' \leftarrow B')$	$\leftarrow (B_1, \Diamond(\bigcirc^{j-i} B_2, B', \bigcirc^{j-i} B_3), B_4)\theta$
8	$i \geq j$	$\leftarrow B_1, \Diamond(B_2, \bigcirc^i A, B_3), B_4$	$\Box(\bigcirc^j A' \leftarrow B')$	$\leftarrow (B_1, \Diamond(B_2, \bigcirc^{i-j} B', B_3), B_4)\theta$

Table 1: TSLD-Resolution Rules for TEMPLOG ($\theta = mgu(A, A')$)

characterized by a sequence of goals $G_0, G_1, \ldots$ where $G_0 = G$; a sequence of candidate next-atoms N_0, $N_1, \ldots$ selected by R from $G_0, G_1, \ldots$, respectively; a sequence of program clauses $C_1, C_2, \ldots$ where each C_i has been renamed so that none of the variables appearing in it also appears in G_{i-1} or in $C_1, \ldots, C_{i-1}$; and a sequence of substitutions $\theta_1, \theta_2, \ldots$, such that G_{i+1} is the goal obtained by applying one of the TSLD-resolution rules to C_{i+1} and G_i with selected next-atom N_i and mgu θ_{i+1}. A *TSLD-refutation* for $P \cup \{G\}$ *via* R is a finite TSLD-derivation whose last goal is empty. (We assume implicitly that the initial goal G is nonempty.) The *R-computed answer substitution* associated with an n-step refutation of $P \cup \{G\}$ via R is the substitution obtained by restricting the composition $(\theta_1 \circ \cdots \circ \theta_n)$ to the variables of G. An answer substitution θ for $P \cup \{G\}$ is said to be *correct* if $P \models (\forall *)B\theta$. The (nonempty) goal G is said to be *n-refutable* ($n \geq 1$) if there is a TSLD-refutation of $P \cup \{G\}$ of length less than n via each computation rule; it is *refutable* if it is n-refutable for some n. Notice that a goal is refutable not simply if it has one TSLD-refutation, but if it has a TSLD-refutation via every computation rule, which is stronger.

Remark: We have augmented the original definition of TEMPLOG given in [AM87] to allow function symbols in terms. Also, we have assumed that all the predicate symbols are flexible, unlike in [AM87] where both rigid and flexible predicate symbols are allowed. However, our assumption is not restrictive as the time-independence of a (ℓ-ary) predicate p can easily and efficiently be expressed in TEMPLOG with the clause $\Box p(X_1, \ldots, X_\ell) \leftarrow \Diamond p(X_1, \ldots, X_\ell)$. The proof method underlying the execution of programs was given in [AM87] for a fixed computation rule that consists in always selecting the leftmost candidate next-atom as in PROLOG ([CM84]). Here, we study the semantics of TEMPLOG for an arbitrary computation rule.

3 Declarative Semantics for TEMPLOG

A TEMPLOG program is a set of statements in temporal logic. Given such a program, a computation consists in trying to derive information that follows from the program. So the declarative meaning of a logic program is characterized by the set of bodies that are logical consequences of the program, that is, the set of bodies that are true in every model of the program. In a first stage, we give a characterization of this denotation of programs in terms of minimal Herbrand model. For this, we introduce the notion of temporal Herbrand model and prove that if a program has a temporal model then it has a temporal Herbrand model. Then we show that the class of temporal Herbrand models of a program is closed under intersection. Combining these results, we prove that the minimal Herbrand model, that is, the intersection of the temporal Herbrand models of a program, satisfies exactly the bodies that are logical consequences of the program, and hence provides a characterization of the denotation of a program. In a second stage, we show how to associate with a TEMPLOG program P a function T_P on the domain of the temporal Herbrand interpretations for P. Intuitively, this mapping corresponds to one step of ground inference from P. We prove that this mapping is continuous and that its least fixpoint is exactly the minimal Herbrand model of the program, thereby providing a fixpoint characterization of the declarative meaning of TEMPLOG programs.

3.1 Model-Theoretic Semantics

Let L be a language characterized by its collection of variables and of constant, function and predicate symbols. The *Herbrand universe* U_L of L is the set of variable-free (that is, *ground*) terms constructed from the constant and the function symbols in L. This notion coincides with the notion of Herbrand universe in classical logic, which is quite natural since the constant and function symbols are rigid. The *temporal Herbrand base* B_L of L is the set of ground next-atoms constructed from the predicate symbols of L and the ground terms of the Herbrand universe U_L. A *temporal Herbrand interpretation* for a language L is a temporal interpretation with the Herbrand universe U_L as domain mapping the ground terms to "themselves" in U_L. A temporal Herbrand interpretation for (the closed formulas of) a language L coincides with a subset of the temporal Herbrand base B_L: it is the set of ground next-atoms that are true under the interpretation (at the initial time). So a ground next-atom N is satisfied by a temporal Herbrand interpretation I, denoted $\models_I N$, iff $N \in I$. Notice that one could equivalently consider the Herbrand base B_L to be, as in classical logic, the set of ground atoms of L. Then, a temporal Herbrand interpretation I could be defined as an ω-sequence of subsets of B_L, or equivalently, a function $I : \omega \rightarrow 2^{B_L}$ that associates with every natural number i the set of ground atoms that are true at time i.

The satisfaction relation for ground TEMPLOG clauses has a simple reformulation when one introduces the notions of temporally ground formula and of temporally ground instance. A formula is said to be *temporally ground* (TG) if $\bigcirc$ is the only temporal operator that appears in it. So atoms and next-atoms as well as program clauses of the form $\bigcirc^i A \leftarrow \bigcirc^{i_1} A_1, \ldots, \bigcirc^{i_m} A_m$, and goal clauses of the form $\leftarrow \bigcirc^{i_1} A_1, \ldots, \bigcirc^{i_m} A_m$ are temporally ground[3]. A *temporally ground instance* (TGI) of a body B is a temporally ground body obtained from B by replacing every occurrence of $\Diamond$ by a finite number of $\bigcirc$'s. Similarly, a *temporally ground instance* (TGI) of a program clause C is obtained from C by replacing each occurrence of $\Box$ and each occurrence of $\Diamond$ by a finite number of $\bigcirc$'s. Using the definition of the satisfaction relation, one can prove the following.

Proposition 3.1 *Let $\mathcal{I}$ be a temporal interpretation of a program or goal clause C (a body B, resp.). Then $\mathcal{I}$ satisfies C (B, resp.) if and only if $\mathcal{I}$ satisfies every TGI of C (some TGI of B, resp.)*

[3]Beware of the difference between *ground* and *temporally ground*: *ground* means *variable-free* whereas *temporally ground* means $\Box$-*free and* $\Diamond$-*free*.

PROOF: The proof is straightforward, once one has noticed that $\models_{\mathcal{I}(i)} F$ if and only if $\models_{\mathcal{I}} \bigcirc^i F$. ■

Intuitively, the property holds because the temporal operators other than $\bigcirc$ are of $\Box$-force in clauses and of $\Diamond$-force in bodies.

A clause is said to be *strictly ground* (SG) if it is both ground (variable-free) and temporally ground ($\Box$-free and $\Diamond$-free). A *strictly ground instance* (SGI) of a clause is an instance of the clause that is both ground and temporally ground. It follows from Proposition 3.1 that a temporal Herbrand interpretation for a program P satisfies P if and only if it satisfies every strictly ground instance of every clause in P.

Proposition 3.2 *Let S be a set of* TEMPLOG *clauses. If S has a temporal model, then S has a temporal Herbrand model.*

PROOF: Let L be the language of the clauses in S, and let $\mathcal{I}$ be a temporal model of S. We associate with $\mathcal{I}$ the temporal Herbrand interpretation

$$I = \{N \in B_L : \models_{\mathcal{I}} N\}.$$

Using Proposition 3.1, one can show that I is a model of S. ■

Property 3.3 (Model Intersection) *Let P be a* TEMPLOG *program. The intersection of a collection of temporal Herbrand models of P is a temporal Herbrand model of P.*

PROOF: Using Proposition 3.1. ■

Intuitively, the Model Intersection Property holds because the temporal operators other than $\bigcirc$ are all of $\Box$-force in clauses. It would not hold for example if the language allowed the use of clauses of the form $\Diamond p \leftarrow$. Indeed, both $I_1 = \{\bigcirc p\}$ and $I_2 = \{\bigcirc^3 p\}$ are models of this clause, but their intersection is not.

Knowing that the intersection of the temporal Herbrand models of a program P is also a model for P, we can now establish that this smallest Herbrand model, denoted M_P, provides a characterization of the declarative semantics of P.

Theorem 3.4 *Let P be a* TEMPLOG *program and B a ground body: $P \models B$ if and only if $\models_{M_P} B$.*

PROOF: [$\Rightarrow$] Trivial (M_P is a model of P).
[$\Leftarrow$] Let $\models_{M_P} B$. By Prop. 3.1, there exists a TGI B^* of B such that $\models_{M_P} B^*$. Let B^* be $N_1 \wedge \ldots \wedge N_m$. Then

$\models_{M_P} N_1 \wedge \ldots \wedge N_m \Rightarrow \{N_1, \ldots, N_m\} \subseteq M_P$
$\Rightarrow$ for every temporal Herbrand model M of P: $\{N_1, \ldots, N_m\} \subseteq M$
$\Rightarrow$ for every temporal Herbrand model M of P: $\models_M B^*$.

It follows that for every temporal Herbrand model M of P there is a TGI B^* of B such that $\models_M B^*$. By Proposition 3.1, we thus have $\models_M B$ for every temporal Herbrand model M of P. So $P \cup \{\neg B\}$ has no temporal Herbrand model, and hence $P \cup \{\neg B\}$ is unsatisfiable (Proposition 3.2). Therefore $P \models B$. ■

The following corollary specifies the contents of M_P as a subset of the Herbrand base. It is simply a restriction of Theorem 3.4 to the case of bodies that are single ground next-atoms.

Corollary 3.5 $M_P = \{\bigcirc^i A \in B_L : P \models \bigcirc^i A\}$.

3.2 Fixpoint Semantics

Let P be a TEMPLOG program with language L. We associate with P a mapping T_P that intuitively represents one step of strictly ground inference from P (we will prove it in the next section). The domain of this mapping is the complete lattice $(2^{B_L}, \subseteq)$. Let I be a temporal Herbrand interpretation of P, that is, $I \in 2^{B_L}$. The mapping T_P is defined by:

$$T_P(I) = \{N \in B_L : N \leftarrow N_1, \ldots, N_m \text{ is a SGI of a clause in } P \text{ and } \{N_1, \ldots, N_m\} \subseteq I\}.$$

For example, let $\Box(\bigcirc^j A \leftarrow B)$ be a ground instance of a permanent clause in P. For every $k \in \omega$, if there is a TGI $N_1 \wedge \ldots \wedge N_m$ of B such that $\{\bigcirc^k N_1, \ldots, \bigcirc^k N_m\} \subseteq I$, then $\bigcirc^{j+k} A \in T_P(I)$. Notice that this definition of T_P is similar to the definition of the mapping associated with classical logic programs, except that in classical logic one deals with atoms and with ground instances of clauses where in temporal logic we deal with next-atoms and with strictly ground instances of clauses, respectively. As a result of this resemblance, the properties of T_P given below (continuity, Proposition 3.6, and Theorem 3.7) admit proofs that are very similar to the proofs of the analogous results for classical logic programming ([vEK76], [AvE82], [Llo84], [Apt87]). The mapping T_P is continuous on $(2^{B_L}, \subseteq)$, and so, by the fixpoint theorem its least fixpoint $lfp(T_P)$ is given by $T_P \uparrow \omega = lub\{T_P{}^i(\emptyset) : i \geq 0\} = \bigcup_{i=0}^{\infty} T_P{}^i(\emptyset)$ (e.g. [Llo84])[4]. The next proposition provides a criterion for a temporal Herbrand interpretation to be a model of a program P as a condition on T_P.

Proposition 3.6 *Let I be a temporal Herbrand interpretation for P. Then $\models_I P$ iff $T_P(I) \subseteq I$.*

PROOF: $\models_I P$ iff for every SGI $N \leftarrow N_1, \ldots, N_m$ of every clause in P: $\models_I N \leftarrow N_1, \ldots, N_m$ (Prop. 3.1), that is, $N \in I$ if $\{N_1, \ldots, N_m\} \subseteq I$. This condition is equivalent to $T_P(I) \subseteq I$. ■

Using Proposition 3.6, we can prove the correspondence between the least Herbrand model M_P and the least fixpoint of T_P.

Theorem 3.7 $M_P = T_P \uparrow \omega$.

PROOF: The least Herbrand model M_P is the intersection of the temporal Herbrand models of P. So in the complete lattice $(2^{B_L}, \subseteq)$:

$$\begin{aligned} M_P &= glb\{I \in 2^{B_L} : \models_I P\} \\ &= glb\{I \in 2^{B_L} : T_P(I) \subseteq I\} \qquad \text{(by Prop. 3.6).} \end{aligned}$$

In other words, M_P is the greatest lower bound of the pre-fixpoints of T_P, which is $lfp(T_P)$ by a version of the fixpoint theorem (e.g. [Llo84]). And so $M_P = T_P \uparrow \omega$ since T_P is continuous. ■

4 Soundness and Completeness of TSLD-resolution

In this section, we establish the soundness and the completeness of the TSLD-resolution proof method underlying TEMPLOG's execution. The soundness proof is straightforward. We first establish the correctness of each resolution rule.

Lemma 4.1 (Soundness of the Rules) *Let $\leftarrow B'$ be the resolvent of the goal $\leftarrow B$ and the* TEMPLOG *program clause C with most general unifier θ. Then $C \models (B\theta \leftarrow B')$.*

PROOF: The proof is carried out separately for each of the eight TSLD-resolution rules of Table 1. ■

Lemma 4.1 allows us to prove the following theorem of which soundness is an immediate corollary.

Theorem 4.2 (Correctness of Computed Answer Substitution) *Let P be a* TEMPLOG *program and B a body. If $P \cup \{\leftarrow B\}$ has a refutation with computed answer substitution θ, then θ is correct, that is, $P \models (\forall *)B\theta$.*

PROOF: By induction on the length of the refutation of $P \cup \{\leftarrow B\}$ and using Lemma 4.1. ■

Corollary 4.3 (Soundness) *Let P be a* TEMPLOG *program and G a goal. If $P \cup \{G\}$ has a TSLD-refutation then $P \cup \{G\}$ is unsatisfiable.*

In classical logic, the proof of the completeness of resolution is based on two main lemmas: a lemma stating the completeness of ground resolution and a *lifting lemma* to "lift" the ground-completeness result to the first-order completeness result ([Rob65],

[4] *lub* stands for least upper bound and *glb* stands for greatest lower bound.

[AvE82], [Llo84], [Apt87]). In the case of temporal logic, our strategy is somewhat similar. We first establish the correspondence between membership in the fixpoint of the mapping T_P and temporally ground refutability (notion to be defined precisely below), thereby obtaining a completeness result for strictly ground formulas (Lemma 4.6). Then we introduce a *temporal lifting lemma* (Lemma 4.8) that allows us to "lift" this completeness result for both ground and temporally ground formulas to a completeness result for ground formulas (Lemma 4.9). Finally, combining this ground-completeness lemma with a lifting lemma (Lemma 4.11) we obtain the desired completeness theorem (Theorem 4.12). It is via the Temporal Lifting Lemma that the notion of temporal groundedness plays its crucial role. The completeness theorem that we prove, that is, Theorem 4.12, is a strong form of completeness. It states that unsatisfiability of a program and goal implies not simply existence of a refutation but rather existence of a refutation via *each* computation rule (that is, *refutability*). At the end of this section, we prove a version of the completeness theorem that takes the computed answer substitutions into account.

Let us first introduce the notions of temporally ground refutation and temporally ground refutability. A *temporally ground derivation/refutation* (TG-derivation/refutation) for a program P and a TG-goal G is a TSLD-derivation/refutation for G that only uses TGI of the clauses in P (and hence only uses the first TSLD-resolution rule of Table 1). There is no occurrence of $\Diamond$ in the goals of a TG-refutation and no occurrence of $\Box$ or $\Diamond$ in the clauses used in a TG-refutation. Given a program P, a temporally ground goal G is said to be *n-TG-refutable* ($n \geq 1$) if there is a TG-refutation for $P \cup \{G\}$ of length less than n via every computation rule; G is *TG-refutable* if it is n-TG-refutable for some $n \geq 1$. As a first step of the completeness proof, we introduce a lifting lemma for TG-refutations (Lemma 4.5) that will be needed in the proof of the completeness theorem for strictly ground refutations (Lemma 4.6). This lifting lemma follows from the following lemma which establishes a correspondence between TG-refutations of a temporally ground goal and an instance of this goal.

Lemma 4.4 *Let P be a* TEMPLOG *program, G a temporally ground goal, θ a substitution, and $n \geq 1$. To any TG-refutation of $P \cup \{G\theta\}$ with mgu's $\theta_1, \ldots, \theta_n$, there corresponds a TG-refutation of $P \cup \{G\}$ with mgu's $\theta'_1, \ldots, \theta'_n$ such that the atom selected at any step of the TG-refutation of $P \cup \{G\theta\}$ is an instance of the atom selected at the corresponding step of the TG-refutation of $P \cup \{G\}$ and the program clauses used are the same in both TG-refutations. Moreover, $(\theta'_1 \circ \cdots \circ \theta'_n) \succeq (\theta \circ \theta_1 \circ \cdots \circ \theta_n)$.*

PROOF: By induction on n. The substitution θ can be assumed to not affect the variables occurring in the program clauses without loss of generality. The key to this proof is the fact that if the atom $A\theta$ selected for the first step of the TG-refutation for $P \cup \{G\theta\}$ unifies with the atom A' in the head of a program clause and $\theta_1 = mgu(A\theta, A')$, then A and A' also unify. This follows from $A\theta\theta_1 = A'\theta_1 = A'\theta\theta_1$, which holds because θ does not affect the variables in A'. Moreover, if $\theta'_1 = mgu(A, A')$ then $\theta'_1 \succeq (\theta \circ \theta_1)$ (by definition of an mgu). In the inductive case ($n > 1$), one also has to show by a similar argument that the derived goal obtained after the first resolution step for $P \cup \{G\theta\}$ is an instance of the derived goal obtained after the corresponding step for $P \cup \{G\}$. ■

Lemma 4.5 (Lifting for TG-Refutability) *Let P be a* TEMPLOG *program, G a temporally ground goal, θ a substitution, and $n \geq 1$. If $G\theta$ is n-TG-refutable, then G is n-TG-refutable.*

PROOF: Immediate consequence of Lemma 4.4. ■

Lemma 4.6 (Strictly Ground Completeness) *Let P be a* TEMPLOG *program and N a ground next-atom. If $N \in M_P$ then $P \cup \{\leftarrow N\}$ is TG-refutable.*

PROOF: Let $N \in M_P$. Since $M_P = T_P \uparrow \omega$ (Theorem 3.7), there is a $k \in \omega$ such that $N \in {T_P}^k(\emptyset)$. One proves by induction on k that if $N \in {T_P}^k(\emptyset)$ then $P \cup \{\leftarrow N\}$ is TG-refutable. The base case is immediate. In the inductive step, let $N \in T_P({T_P}^{k-1}(\emptyset))$. So there is a SGI $(N' \leftarrow N_1, \ldots, N_m)\theta$ of a clause in P such that $N'\theta = N$ and $\{N_1\theta, \ldots, N_m\theta\} \subseteq {T_P}^{k-1}(\emptyset)$. By the induction hypothesis, each of $P \cup \{\leftarrow N_1\theta\}$, $\ldots$, $P \cup \{\leftarrow N_m\theta\}$ is TG-refutable. Since the $N_i\theta$ are ground, their TG-refutations are independent from one another, and they can be combined in any desired way. So $P \cup \{\leftarrow (N_1, \ldots, N_m)\theta\}$ is TG-refutable.

The first step of a TG-refutation for $P \cup \{\leftarrow N\}$ uses $N' \leftarrow N_1, \ldots, N_m$. The derived goal is a goal of which $\leftarrow (N_1, \ldots, N_m)\theta$ is an instance, and so, by Lemma 4.5, it is TG-refutable. Therefore $P \cup \{\leftarrow N\}$ is TG-refutable. ■

The next step in the proof of the completeness of TSLD-resolution is the "temporal lifting" of the Strictly Ground Completeness Lemma (by the Temporal Lifting Lemma). We first introduce Lemma 4.7 which establishes the correspondence between the steps of a TG-derivation and those of a TSLD-derivation. The Temporal Lifting Lemma follows immediately from Lemma 4.7.

Lemma 4.7 *Let G be a goal, and let G^* be a temporally ground instance of G. Let N be the next-atom*

selected from G by a given computation rule, and let N^* be the corresponding next-atom in G^*. Let C be a program clause, and let C^* be a temporally ground instance of C. If there is a TG-resolution step between C^* and G^* with selected next-atom N^* that produces the (temporally ground) derived goal G_1^*, then there is a TSLD-resolution step between C and G with selected next-atom N that produces the derived goal G_1, and G_1^* is a temporally ground instance of G_1.

PROOF: The proof separates in cases. We have to consider the cases where the next-atom N^* corresponds to a next-atom N that is in the scope of zero or one $\Diamond$ in G. For each of these two cases, we consider the subcases where the program clause C^* is the TGI of a clause C that is initial with or without $\Box$ in the head or permanent. In studying all these cases, we exhaust the eight TSLD-resolution rules of Table 1. ■

Lemma 4.8 (Temporal Lifting) *Let P be a* TEMPLOG *program and G a goal. If G has a temporally ground instance G^* such that $P \cup \{G^*\}$ is n-TG-refutable for some $n \geq 1$, then $P \cup \{G\}$ is refutable*[5].

PROOF: By induction on n and using Lemma 4.7. ■

Lemma 4.9 (Ground Completeness) *Let P be a* TEMPLOG *program and B a ground body. If $\models_{M_P} B$ then $P \cup \{\leftarrow B\}$ is refutable.*

PROOF: Let $\models_{M_P} B$. By Prop. 3.1, there is a TGI $N_1 \wedge \ldots \wedge N_m$ of B such that $\{N_1, \ldots, N_m\} \subseteq M_P$. For this TGI of B, we have

$\{N_1, \ldots, N_m\} \subseteq M_P$
$\Rightarrow \forall i = 1, \ldots, m$: $P \cup \{\leftarrow N_i\}$ is TG-refutable
(by the Strictly Ground Completeness Lemma)
$\Rightarrow P \cup \{\leftarrow N_1, \ldots, N_m\}$ is TG-refutable

since the N_i's are ground and their refutations are temporally ground. Therefore, $P \cup \{\leftarrow B\}$ is refutable (by the Temporal Lifting Lemma). ■

Next we introduce a lifting lemma to be used together with the Ground Completeness Lemma in the proof of the Completeness Theorem. It is the analogous for TSLD-refutability of the Lifting Lemma for TG-refutability (Lemma 4.5). As for TG-refutability, we introduce a preliminary lemma from which the Lifting Lemma directly follows.

Lemma 4.10 *Let P be a* TEMPLOG *program, G a goal, θ a substitution, and $n \geq 1$. To any TSLD-refutation of $P \cup \{G\theta\}$ with mgu's $\theta_1, \ldots, \theta_n$, there corresponds a TSLD-refutation of $P \cup \{G\}$ with mgu's $\theta'_1, \ldots, \theta'_n$ such that the atom selected at any step of the refutation of $P \cup \{G\theta\}$ is an instance of the atom selected at the corresponding step of the refutation of $P \cup \{G\}$ and the program clauses used are the same in both TSLD-refutations. Moreover, $(\theta'_1 \circ \cdots \circ \theta'_n) \succeq (\theta \circ \theta_1 \circ \cdots \circ \theta_n)$.*

PROOF: Similar to the proof of Lemma 4.4. ■

Lemma 4.11 (Lifting) *Let P be a* TEMPLOG *program, G a goal, θ a substitution, and $n \geq 1$. If $G\theta$ is n-refutable, then G is n-refutable.*

PROOF: Immediate consequence of Lemma 4.10. ■

Theorem 4.12 (Completeness) *Let P be a* TEMPLOG *program and G a goal. If $P \cup \{G\}$ is unsatisfiable, then $P \cup \{G\}$ is refutable, that is, $P \cup \{G\}$ has a refutation via every computation rule.*

PROOF: Let G be the goal $\leftarrow B$ such that $P \cup \{\leftarrow B\}$ is unsatisfiable. For every temporal model $\mathcal{I}$ of P, we have $\not\models_{\mathcal{I}} \neg B$, and in particular $\not\models_{M_P} \neg B$. So there is a ground instance $B\theta$ of B such that $\models_{M_P} B\theta$, and by the Ground Completeness Lemma $P \cup \{\leftarrow B\theta\}$ is refutable. Therefore $P \cup \{\leftarrow B\}$ is refutable (by the Lifting Lemma). ■

Next, we extend this result to take the computed answer substitutions into account. One cannot show that any correct answer substitution can be computed by a refutation. Instead, we prove Theorem 4.14 which states that for any correct answer substitution, one can compute via every computation rule an answer substitution that is more general than the correct answer substitution. For this, we first introduce Lemma 4.13. The proofs of Lemmas 4.13 and Theorem 4.14 do not use the Completeness Theorem which could then also be derived as a corollary to Theorem 4.14.

Lemma 4.13 *Let P be a* TEMPLOG *program and B a body. If $P \models (\forall *)B$, then there is a TSLD-refutation of $P \cup \{\leftarrow B\}$ via every computation rule with the empty substitution as computed answer substitution.*

PROOF: Let θ be a substitution that replaces the free variables of B with arbitrary new constants. Then $P \models B\theta$ where $B\theta$ is ground. So by the Ground Completeness Lemma, $P \cup \{\leftarrow B\theta\}$ has a TSLD-refutation (with empty computed answer substitution) via every computation rule. But the new constants can be textually replaced by the original variables in the refutations of $P \cup \{\leftarrow B\theta\}$ to produce refutations of $P \cup \{\leftarrow B\}$ with the empty substitution as computed answer substitutions. ■

Theorem 4.14 (Computability of Correct Answer Substitution) *Let P be a* TEMPLOG *program, G a goal, and θ a correct answer substitution for*

[5] Remember that we defined refutability to mean existence of a refutation via every computation rule.

$P \cup \{G\}$. *For any computation rule R, there is an R-computed answer substitution* σ_R *for* $P \cup \{G\}$ *such that* $\sigma_R \succeq \theta$.

PROOF: Let G be $\leftarrow B$. Since θ is a correct answer substitution for $P \cup \{\leftarrow B\}$, we have $P \models (\forall *)B\theta$. So by Lemma 4.13, $P \cup \{\leftarrow B\theta\}$ has a TSLD-refutation with the empty answer substitution via every computation rule, and the desired result follows by Lemma 4.10. ■

5 A fragment of TEMPLOG: TL1

In this section, we examine a fragment of TEMPLOG that we call TL1. In TL1, the body of a clause cannot contain any occurrence of $\Diamond$ and initial clauses cannot have $\Box$ in their head. So in TL1, a body is a conjunction of next-atoms and a clause is either of the form $N \leftarrow B$ (initial) or of the form $\Box(N \leftarrow B)$ (permanent). The proof method for TL1 is based on the TSLD-resolution rules (1) and (3) of Table 1. There are several reasons that make TL1 worth considering. First, it is one of the smallest extensions of Horn logic programming with temporal operators; it was introduced in [AM87] as a first step towards temporal logic programming. As we will show in the next section, it has theoretically the same expressiveness as TEMPLOG, although in practice TEMPLOG computations can be considerably more efficient than their TL1 counterparts. Moreover, TL1 is one of the few subsets of TEMPLOG that is closed under the applicable TSLD-resolution rules; on the contrary, any proper subset of TEMPLOG that allows the use of $\Diamond$ in the body of clauses is not closed under the TSLD-resolution rules. Finally, TL1 is equivalent to the "pure" fragment of the THLP language[6] introduced by Wadge in [Wad88] and also referred to as CHRONOLOG in [OW88a]. However, the only interpretation method suggested for THLP consists in reducing the programs to classical Horn programs with explicit time parameters and interpreting them with classical logic programming methods. One of the drawbacks of this approach is that the time parameter is treated as any other parameter by the logic programming interpreter.

The declarative semantics of TL1 can be given in model-theoretic and in fixpoint terms like that of TEMPLOG. One can also establish the completeness of the TSLD-resolution method for TL1. This development is omitted here as it is essentially superseded by the semantic development for TEMPLOG. However, it is interesting to note that the proofs can be completely carried out without introducing the notion of temporal groundedness, and completeness can be proved without the need for a temporal lifting lemma.

[6] THLP stands for Temporal Horn Logic Programming.

6 TEMPLOG's Expressiveness

In this section, we consider exclusively the propositional subset of TEMPLOG, that is, the subset in which all predicates are 0-ary. This will enable us to study the purely temporal aspect of TEMPLOG's expressiveness. The fixpoint formulation of TEMPLOG's semantics suggests a relation to temporal fixpoint calculi ([BB86], [Var88]). Indeed, propositional TEMPLOG queries can be translated into a fragment of the μTL of [Var88], namely the positive fragment of μTL that allows only least fixpoint operators. We give the flavor of the translation between TEMPLOG programs and formulas of this fragment of μTL on an example.

Example 6.1 The following two program clauses define a predicate u that holds whenever p holds an even number of time instants later.

$$\Box(u \leftarrow p)$$
$$\Box(u \leftarrow \bigcirc\bigcirc u)$$

Notice that u can be seen as the result of applying a temporal operator to p, and that this operator is the dual of the *even* operator shown in [Wol83] to be inexpressible in temporal logic. The least-fixpoint semantics of the clauses for u can be expressed by the μTL formula $\mu X.(p \vee \bigcirc\bigcirc X)$. It is the least fixpoint (with respect to propositional variable X) of the disjunction of the bodies of the clauses defining u (in which u is replaced by the variable X). ■

This example shows that there are properties expressible in TEMPLOG which are not expressible in temporal logic. On the other hand, there are formulas of temporal logic that are not expressible in TEMPLOG since expressing all of temporal logic in μTL can require using greatest fixpoints or the alternation of a greatest and a least fixpoint ([Par81]). In terms of languages, μTL has the expressive power of ω-regular expressions whereas temporal logic has the expressiveness of star-free ω-regular expressions ([Tho81]). The expressiveness of TEMPLOG is clearly less than that of ω-regular languages. On the other hand, it is incomparable to star-free ω-regular languages. We will prove that the expressiveness of TEMPLOG is essentially that of *finitely regular ω-languages*. An ω-language L is *finitely regular* if there is a regular language L' such that each (infinite) word in L has a finite prefix in L'.

Let us first formally set up the framework for studying the expressiveness of TEMPLOG in terms of ω-languages. For propositional TEMPLOG, a temporal

interpretation consists of a sequence of states isomorphic to ω together with an interpretation function giving, for each state, the (0-ary) predicates true in that state. Such an interpretation can be seen as an infinite word over the alphabet $2^{\mathcal{P}}$, where $\mathcal{P}$ is the set of predicates in the language. Notice that there is no distinction between temporal interpretations and temporal Herbrand interpretations in the propositional case. So we can characterize an interpretation by the set of next-atoms that hold in it. A *finite prefix* of an interpretation is a restriction of the interpretation to a prefix of ω. Any finite set of next-atoms is a finite prefix of an interpretation.

To give a meaningful characterization of the expressiveness of TEMPLOG, we consider sets P of program clauses that define some predicates $u_1, \ldots, u_m$ in terms of themselves and in terms of other predicates $p_1, \ldots, p_n$ not defined in P. To emphasize the fact that the predicates $p_1, \ldots, p_n$ are not defined by P, we denote the program by $P(p_1, \ldots, p_n)$. Each of the u_i defined by P corresponds to a temporal operator whose arguments are $p_1, \ldots, p_n$.

Example 6.2 The following program $P(p, q)$ defines a predicate u that holds exactly when $p \mathcal{U} q$ holds, $\mathcal{U}$ denoting the *strong-until* operator.

$$\Box(u \leftarrow q)$$
$$\Box(u \leftarrow p, \bigcirc u) \quad \blacksquare$$

The semantics of a program $P(p_1, \ldots, p_n)$ must naturally be a function of the semantics of $p_1, \ldots, p_n$, that is, of the interpretation of $\{p_1, \ldots, p_n\}$. Let us view $P(p_1, \ldots, p_n)$ as the top layer of a two-layer program whose bottom layer defines $p_1, \ldots, p_n$. More precisely, a program is said to be *layered* if it can be partitioned into sets of clauses (*layers*) $P_1, \ldots, P_k$ such that the definition of each predicate is completely contained within one layer and for every i $(1 \leq i \leq k)$, the predicate symbols appearing in the body of the clauses in P_i are defined in a layer P_j such that $1 \leq j \leq i$. Two-layer programs are sufficient for our purpose here. The fixpoint semantics of a layered program can be reformulated in a way that reflects its layering, somewhat like the iterated fixpoint semantics of the stratified programs of classical logic ([Min88]).

Proposition 6.1 *Consider a two-layer* TEMPLOG *program $P = P_1, P_2$ whose minimal Herbrand model is M_P. Let M_1 denote the minimal Herbrand model of P_1. Let T_2 be the mapping associated with P_2 as defined in Section 3.2, and let T_2' be defined by $T_2'(I) = I \cup T_2(I)$. Then $M_P = \bigcup_{i=0}^{\infty} T_2'^{i}(M_1)$.*

PROOF: The proof is quite straightforward. It involves using the monotonicity and the continuity of T_2 (proved in Section 3.2). ■

In our case, we consider programs $P(p_1, \ldots, p_n)$ whose bottom layer is arbitrary. So we define the semantics of P in terms of interpretations of $\{p_1, \ldots, p_n\}$. Let T_P be the mapping associated with the clauses in P as defined in Section 3.2, and let $T_P'(I) = I \cup T_P(I)$. Then the *semantics of $P(p_1, \ldots, p_n)$ with respect to I*, denoted $M_P(I)$, is given by $M_P(I) = \bigcup_{i=0}^{\infty} T_P'^{i}(I)$.

This sets up the framework for understanding how programs characterize sets of words. The combination of a program $P(p_1, \ldots, p_n)$, defining predicates $u_1, \ldots, u_m$, and a goal $\leftarrow \bigcirc^i u_\ell$ characterizes the collection of interpretations I of $\{p_1, \ldots, p_n\}$ (collection of words on $2^{\{p_1, \ldots, p_n\}}$) such that $\bigcirc^i u_\ell$ holds in the semantics of P considered with respect to I, that is, such that $\models_{M_P(I)} \bigcirc^i u_\ell$. Notice, however, that when $\bigcirc^i u_\ell$ holds in $M_P(I)$, there is a finite prefix I^* of I such that $\bigcirc^i u_\ell$ holds in $M_P(I^*)$.

This last fact partially explains why the expressiveness of TEMPLOG programs can be characterized in terms of finitely regular ω-languages. To prove this characterization, we will show how one can build a finite-acceptance finite automaton on infinite words from a TEMPLOG program and a goal, as well as give the opposite construction. A finite-acceptance automaton accepts an infinite word iff it accepts a finite prefix of that word ([WVS83], [VW88]). Except for the fact that it is applied to prefixes of infinite words, a finite acceptance automaton is identical to a classical finite automaton. Finite acceptance automata thus characterize the finitely regular ω-languages. However, we should note that without further assumptions, the construction of a TEMPLOG program from an automaton yields a program that defines a superset of the set of interpretations characterized by the automaton. The needed additional assumptions will appear clearly once we have given the proofs, and we will discuss them below.

Theorem 6.2 (From Programs to Automata) *Let $P(p_1, \ldots, p_n)$ be a* TEMPLOG *program defining predicates $u_1, \ldots, u_m$. To this program $P(p_1, \ldots, p_n)$ and any goal $\leftarrow \bigcirc^i u_\ell$ with $1 \leq \ell \leq m$, one can associate a finite automaton A such that for every interpretation I of $\{p_1, \ldots, p_n\}$, A accepts a finite prefix of I if and only if $\models_{M_P(I)} \bigcirc^i u_\ell$.*

PROOF: This theorem is proved by techniques similar to those used in [Var88], [WVS83] and [VW88]. The proof will be given in the full paper. ■

Let us now consider the other direction, that is, the construction of a TEMPLOG program corresponding to

a finite automaton. We first give the theorem and its proof. Notice that in the statement of the theorem, every sequence accepted by the automaton produces a model of the program that satisfies the goal, but the converse does not always hold. We will see how the correspondence can be made exact after giving the proof.

Theorem 6.3 (From Automata to Programs) *Let A be a finite automaton. There is a* TEMPLOG *program $P(p_1, \ldots, p_n)$ defining a predicate u such that for every interpretation I of $\{p_1, \ldots, p_n\}$, if A finitely accepts I then $\models_{M_P(I)} u$.*

PROOF: Let $A = (\mathcal{A}, S, \rho, \{\mathbf{s}_0\}, F)$, where $\mathcal{A} = \{a_1, \ldots, a_n\}$ is the alphabet, $S = \{\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_k\}$ is the set of states, $\rho : S \times \mathcal{A} \rightarrow 2^S$ is the transition relation, $\mathbf{s}_0$ is the initial state, and $F \subseteq S$ is the set of final states. We encode the automaton's alphabet with predicate symbols. So to each a_j corresponds a predicate symbol p_j $(1 \leq j \leq n)$. We now construct a TEMPLOG program $P(p_1, \ldots, p_n)$ defining a predicate u. The program will have to encode the transition relation of the automaton. For this, we introduce an auxiliary predicate s_j for each state $\mathbf{s}_j$ of the automaton $(0 \leq j \leq k)$. The clauses of $P(p_1, \ldots, p_n)$ are obtained as follows.

- For the initial state $\mathbf{s}_0$, we introduce in P the clause

  $$\Box(u \leftarrow s_0).$$

- For every alphabet symbol $a_j \in \mathcal{A}$ and every pair of automaton states $\mathbf{s}_v, \mathbf{s}_w \in S$ such that $\mathbf{s}_w \in \rho(\mathbf{s}_v, a_j)$, we introduce in P the clause

  $$\Box(s_v \leftarrow p_j, \bigcirc s_w).$$

- For every final state $\mathbf{s}_v \in F$, we introduce in P the clause

  $$\Box\, s_v \leftarrow .$$

To prove that if A accepts I then $\models_{M_P(I)} u$, we establish the following intermediate result.

> *Let $i \in \omega$, $j \geq 1$, and $\mathbf{s}_v \in S$. If $A_{\mathbf{s}_v}$ has an accepting run of length at most j over $I^{(i)}$, then $\bigcirc^i s_v \in T_P'^{\,j}(I)$, where $A_{\mathbf{s}_v}$ is the automaton that is identical to A except for its initial state which is $\mathbf{s}_v$ instead of $\mathbf{s}_0$.*

This lemma is proved by induction on j . The correctness of the construction of P follows immediately from the lemma (take $\mathbf{s}_v$ to be $\mathbf{s}_0$). ■

In the construction of a TEMPLOG program corresponding to a finite automaton, we had to encode the alphabet of the automaton with predicate symbols. One problem with the encoding we have used is that the predicate symbols are not mutually exclusive: the fact that one of them holds at a certain time does not prevent another one from holding at that same time. Let us illustrate this with an example.

Example 6.3 Suppose that we try to encode in TEMPLOG the automaton with alphabet $\{a, b, c\}$ that accepts the regular language $(ab)^*c$. We associate predicate symbols p, q, and r to a, b and c, respectively. Then we construct the program as described in the proof of Theorem 6.3, and obtain the following.

$$\Box(u \leftarrow r)$$
$$\Box(u \leftarrow p, \bigcirc q, \bigcirc\bigcirc u)$$

Let us consider the goal $\leftarrow u$. The collection of interpretations I of $\{p, q, r\}$ such that $\models_{M_P(I)} u$ contains not only the interpretations that have a finite prefix I_k of the form $\{p, \bigcirc q, \bigcirc^2 p, \bigcirc^3 q, \ldots, \bigcirc^{2k-2} p, \bigcirc^{2k-1} q, \bigcirc^{2k} r\}$, but also all those that have a finite prefix *containing* I_k, like for example the interpretation in which p and q are true at every time instant and r is true at some time instant. If we could instead encode the alphabet symbols a, b, c respectively with $(\neg p \wedge \neg q)$, $(p \wedge \neg q)$, and $(\neg p \wedge q)$, which are mutually exclusive formulas, this problem would disappear. ■

Thus what is missing to obtain an exact correspondence between TEMPLOG and finitely regular ω-languages is the possibility of allowing the predicate symbols $p_1, \ldots, p_n$ to occur negated in the body of the clauses of a program $P(p_1, \ldots, p_n)$. This is necessary for unambiguously representing the alphabet of an automaton. Notice that we do not need to allow the defined predicates $u_1, \ldots, u_m$ to appear negated in P, only the bottom-layer predicates. It is straightforward to adapt our proofs to show that, with this extension, the correspondence between the expressiveness of TEMPLOG and finitely regular ω-languages is exact.

One could imagine extending TEMPLOG further to allow full stratified negation, that is, no predicate is defined in terms of its own negation, but can be defined in terms of the negation of the predicates defined in a lower layer. In that case, the expressiveness of the extended language would be that of the ω-regular expressions. Indeed, such a use of negation would make it possible to obtain the alternation of a greatest and a least fixpoint sufficient to define all ω-regular languages. This last result is essentially only of theoretical interest, since the natural procedural semantics of stratified programs based on the TSLD-

resolution method would not constitute a complete proof system for this extended language.

Interestingly, stratified programs were first introduced by Chandra and Harel in a paper in which they study the expressiveness of DATALOG queries, that is, queries of Horn logic programming without function symbols, and compare it with fixpoint logic on finite structures ([CH85]). In this paper, they first show that DATALOG queries are equivalent to a fragment of fixpoint logic, namely, the one in which formulas consist of a least-fixpoint operator applied to a positive existential formula. It was hoped that extending DATALOG with stratified negation would extend the expressiveness of the queries to that of the full fixpoint logic on finite structures. However, Kolaitis proved in [Kol88] that stratified programs have a strictly weaker expressive power than fixpoint logic on finite structures. So the similarity does not carry over: although adding stratified negation to TEMPLOG yields the expressiveness of the temporal fixpoint calculus, adding stratified negation to DATALOG does not yield the full expressiveness of fixpoint logic on finite structures.

7 Conclusion and Related Work

We have developed the declarative (logical) semantics of TEMPLOG programs and expressed it in two equivalent ways: as a minimal temporal Herbrand model and as the least fixpoint of a mapping. We proved a correspondence between the least fixpoint semantics and the existence of refutations, hence proving a completeness theorem for strictly ground formulas. From this theorem and lifting lemmas, we established the completeness of TSLD-resolution.

In classical logic, the proof of the completeness of resolution relies on the Herbrand's theorem, which is an immediate consequence of the compactness of first-order logic ([Rob65], [Hil74], [Lov78]). Compactness can be derived from the completeness of first-order logic ([End72], [Lov78]). First-order temporal logic is neither complete nor compact, so we could not rely a priori on such results for TEMPLOG. However, we were able to establish completeness for the subset of temporal logic that constitutes TEMPLOG. So, it is natural at this point to wonder whether results such as compactness and Herbrand's theorem also hold for this subset of temporal logic. To derive compactness, we have to begin by extending the completeness theorem proved in this paper to the case of programs that can have infinitely many clauses. This can be done without difficulty. Then, compactness follows from such a (stronger) completeness theorem, and a Herbrand-like theorem can be stated.

Related work on the semantics of programming in non-classical logics includes that of [OW88a] and [OW88b] which was developed independently. There, Orgun and Wadge study the declarative semantics of "intensional" (modal) extensions of Horn clause programs. One such extension that they consider is the THLP language we discussed in the previous section. They give declarative semantics similar to ours, but as they do not consider proof systems in conjunction with their language, they have no completeness results. Also, as far as temporal programming, their results are only given for a language equivalent to our TL1. In the conclusions of [OW88a] and [OW88b], it is mentioned that one of their results, namely the minimal model semantics, also holds for full TEMPLOG.

In [Far86], Fariñas del Cerro defines a framework, called MOLOG, for programming in modal logics. This framework is based on resolution proof methods for such logics. In a recent paper ([BFH88]) Balbiani et al. provide declarative and operational semantics for one language in the MOLOG family and prove the equivalence of these semantics.

Gabbay has proposed an extension of classical logic programming distinct from TEMPLOG ([Gab87]). His TEMPORAL PROLOG is based on a different subset of temporal logic: $\Box$ can only be applied to entire clauses and the only operators allowed in the body and in the head of clauses are $\Diamond$ and the corresponding operator for the past. A proof method is sketched for this language, but it is unclear how it could be used as the basis of an execution mechanism and of operational semantics for the language. The only semantics defined for this language is its logical semantics.

For temporal languages like Moszkowski's TEMPURA ([Mos86]) and TOKIO ([FKTMo86]), which view executing a program as constructing a model for the program, the semantic issues are completely different. In fact, in the case of TEMPURA that imperatively executes a temporal logic formula, the states of the computation are exactly the states of the model of the formula, and the operational semantics of a program corresponds to its logical semantics. TOKIO extends PROLOG with temporal constructs that are interpreted as control features. To give its formal semantics one would need to combine a semantics of temporal logic with a semantics of PROLOG that explicitly represents the execution mechanism. Such a semantics could, for instance, be based on that of [JM84], [DM88] or [Bau88b].

Acknowledgements

I wish to thank Martín Abadi, Rajeev Alur, Phokion Kolaitis, Zohar Manna, Amir Pnueli, Carolyn Talcott, Moshe Vardi and Pierre Wolper for related discussions and/or valuable comments on drafts of this paper.

References

[Aba87] Martín Abadi. *Temporal-Logic Theorem Proving.* PhD thesis, Computer Science Department, Stanford University, Stanford, CA, March 1987.

[AM87] Martín Abadi and Zohar Manna. Temporal logic programming. In *International Symposium on Logic Programming*, pages 4–16, San Francisco, CA, September 1987. IEEE. An extended version will appear in the Journal of Symbolic Computation.

[Apt87] Krzysztof R. Apt. Introduction to logic programming. Technical Report TR-87-35, Department of Computer Science, The University of Texas at Austin, September 1987.

[AvE82] Krzysztof R. Apt and M.H. van Emden. Contributions to the theory of logic programming. *Journal of the ACM*, 29(3):841–862, July 1982.

[Bau88a] Marianne Baudinet. On the semantics of temporal logic programming. Technical Report STAN-CS-88-1203, Computer Science Department, Stanford University, Stanford, CA, May 1988.

[Bau88b] Marianne Baudinet. Proving termination properties of PROLOG programs: A semantic approach. In *Symposium on Logic in Computer Science*, pages 336–347, Edinburgh, Scotland, July 1988. IEEE.

[BB86] Behnam Banieqbal and Howard Barringer. A study of an extended temporal logic and a temporal fixed point calculus. Technical Report UMCS86-10-2, Department of Computer Science, University of Manchester, 1986.

[BFH88] Ph. Balbiani, L. Fariñas del Cerro, and A. Herzig. Declarative semantics for modal logic programs. In *International Conference on Fifth Generation Computer Systems 1988*, Tokyo, Japan, December 1988.

[CH85] Ashok K. Chandra and David Harel. Horn clause queries and generalizations. *Journal of Logic Programming*, 2(1):1–15, 1985.

[Cla79] K.L. Clark. Predicate logic as a computational formalism. Technical Report 79/59 TOC, Department of Computing and Control, Imperial College, London, December 1979.

[CM84] W.F. Clocksin and C.S. Mellish. *Programming in Prolog.* Springer-Verlag, Berlin, second edition, 1984.

[DM88] Saumya K. Debray and Prateek Mishra. Denotational and operational semantics for Prolog. *Journal of Logic Programming*, 5(1):61–91, March 1988.

[End72] Herbert B. Enderton. *A Mathematical Introduction to Logic.* Academic Press, 1972.

[Far86] Luis Fariñas del Cerro. Molog: A system that extends Prolog with modal logic. *New Generation Computing*, 4:35–50, 1986.

[FKTMo86] M. Fujita, S. Kono, H. Tanaka, and T. Moto-oka. Tokio: Logic programming language based on temporal logic and its compilation to PROLOG. In *Third International Conference on Logic Programming*, pages 695–709, London, July 1986. LNCS 225, Springer-Verlag.

[Gab86] Dov Gabbay. Modal and temporal logic programming. Technical Report 86/15, Department of Computing, Imperial College, London, June 1986.

[Gab87] Dov Gabbay. Modal and temporal logic programming. In Antony Galton, editor, *Temporal Logics and Their Applications*, chapter 6, pages 197–237. Academic Press, London, 1987.

[Hil74] Robert Hill. LUSH-resolution and its completeness. Technical Report 78, Department of Artificial Intelligence, University of Edinburgh, Edinburgh, Scotland, 1974.

[JM84] Neil D. Jones and Alan Mycroft. Stepwise development of operational and denotational semantics for Prolog. In *International Symposium on Logic Programming*, pages 281–288, Atlantic City, NJ, February 1984. IEEE.

[Kol88] Phokion Kolaitis. The expressive power of stratified logic programs. Manuscript, Submitted for Publication, 1988.

[Llo84] J.W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, Berlin, 1984.

[LMM88] J-L. Lassez, M.J. Maher, and K. Marriot. Unification revisited. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, chapter 15, pages 587–625. Morgan Kaufmann, Los Altos, CA, 1988.

[Lov78] Donald W. Loveland. *Automated Theorem Proving: A Logical Basis.* North-Holland, Amsterdam, 1978.

[Min88] Jack Minker, editor. *Foundations of Deductive Databases and Logic Programming.* Morgan Kaufmann, Los Altos, CA, 1988.

[Mos84] Ben Moszkowski. Executing temporal logic programs. Technical Report No. 55, Computer Laboratory, University of Cambridge, Cambridge, England, 1984.

[Mos86] Ben Moszkowski. *Executing Temporal Logic Programs.* Cambridge University Press, Cambridge, England, 1986.

[MW89] Zohar Manna and Richard Waldinger. *The Logical Basis for Computer Programming. Volume II: Deductive Systems.* Addison-Wesley, 1989. Forthcoming.

[OW88a] Mehmet A. Orgun and William W. Wadge. Chronolog: A temporal logic programming language and its formal semantics. Unpublished Manuscript, 1988.

[OW88b] Mehmet A. Orgun and William W. Wadge. A theoretical basis for intensional logic programming. In *Symposium on Lucid and Intensional Programming*, pages 33–49, Sidney, B.C., Canada, April 1988.

[Par81] David Park. Concurrency and automata on infinite sequences. In Peter Deussen, editor, *Theoretical Computer Science.* LNCS 104, Springer-Verlag, March 1981.

[Rob65] J.A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, January 1965.

[Sak] Takashi Sakuragawa. Temporal Prolog. To appear in *RIMS Conference on Software Science and Engineering.* LNCS, Springer-Verlag.

[Tho81] Wolfgang Thomas. A combinatorial approach to the theory of ω-automata. *Information and Control*, 48(3):261–283, March 1981.

[Var88] Moshe Y. Vardi. A temporal fixpoint calculus. In *Fifteenth ACM Symposium on Principles of Programming Languages*, pages 250–259, San Diego, CA, January 1988.

[vEK76] M.H. van Emden and R.A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, October 1976.

[VW88] Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computation paths. Manuscript, Submitted for Publication, 1988.

[Wad85] William W. Wadge. Tense logic programming: a sane alternative. Unpublished Manuscript, 1985.

[Wad88] William W. Wadge. Tense logic programming: a respectable alternative. In *Symposium on Lucid and Intensional Programming*, pages 26–32, Sidney, B.C., Canada, April 1988.

[Wol83] Pierre Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1–2):72–99, 1983.

[WVS83] Pierre Wolper, Moshe Y. Vardi, and A. Prasad Sistla. Reasoning about infinite computation paths. In *24th Symposium on Foundations of Computer Science*, pages 185–194, Tucson, Arizona, November 1983.