A Family of Temporal Languages

for the Description of Transition Constraints

by

M.A. Casanova
Centro Científico - IBM do Brasil
Caixa Postal 853
70.000, Brasília, DF
Brasil

A. L. Furtado
Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro
Rua Marquês de São Vicente, 225
22.453, Rio de Janeiro, RJ
Brasil

January 1983
Abstract

A purely declarative formalism for the description of transition constraints is described. The formalism is based on a family of languages that extends Temporal Logic with a mechanism to define new modalities. This mechanism increases the expressive power of Temporal Logic and facilitates the description of transition constraints. Finally, a series of results about the solvability of the decision problem of these languages is presented.

key words: database, Temporal Logic, semantic integrity constraints, grammars, formal languages, semantic tableaux.
1. Introduction

A database description, or database schema, consists of a set of data structure descriptions indicating how data is logically organized in the database and a set of static constraints capturing the semantics of the data by imposing restrictions on the allowed database states.

Considerable effort has been spent on devising formal languages tailored to the description of static constraints and on investigating their decision problem.

However, static constraints do not cover situations where restrictions on sequences of database states must be imposed. An example is the restriction that salaries must never decrease. Restrictions on sequences of database states are called transition constraints. (Observe that, unlike [Gr], we do not confine ourselves to just sequences of length 2).

Despite their importance, almost no formal treatment of transition constraints can be found in the literature. This paper then tries to remedy this neglect by presenting a family of formal languages to describe transition constraints and by investigating their decision problem.

There are in principle several alternative approaches to formalise transition constraints. Perhaps the simplest one would be to use first-order languages with explicit "state variables" acting as indices on the other terms. For example, instead of using a binary predicate symbol EMP, with EMP(n,s) interpreted as employee n has salary s, we would have a ternary predicate symbol EMPl, with EMPl(n,s,t) interpreted as employee n has salary s in state t. With this device and a binary predicate symbol U, with U(t,t') interpreted as t' is the state after t, we could translate "salaries never decrease" as

\[ \forall n \forall s \forall t \forall t' (EMPl(n,s,t) \land EMPl(n,s',t') \land U(t,t') \Rightarrow s \leq s') \]

We discarded this alternative both because having an extra "state variable" is awkward and because it was not clear that many interesting constraints could be thus expressed. However, this approach, casted in the language of classic Temporal Logic [RU], was followed in [CCF] and in [C1]. We also note here that a fragment of branching time Temporal Logic was used in [MWJ] to investigate the possibility of monitoring future events in DB environments.

A second alternative would be to assume that the database is updated via a pre-specified set of built-in operations, and then phrase transition constraints as properties of the operations (described using a suitable programming logic, such as Dynamic Logic [Ha]). The drawback here is exactly
that transition constraints are only implicitly specified as a consequence of assuming built-in operations. As argued elsewhere [CCF], it is advantageous to have an independent description of transition constraints, defined as declaratively as possible, without any reference to how the database will be updated.

The third approach, which we follow here, would be to adopt a formal language that does not explicitly talk about states and yet is able to express restrictions on sequences of states. One such language is Temporal Logic, as described in connection with the specification and verification of concurrent programs [Pn, MP, MW], or network protocols [SM]. Temporal Logic, as considered in [Pn, MP, MW], is Propositional Calculus extended with four modalities: ♦ P ("P is true in the next state"), ♣ P ("eventually P will be true"), □ ♦ P ("henceforth P will always be true") and □ P ♦ Q ("henceforth P will always be true until Q is true"). A first-order-like version of Temporal Logic could also be defined by taking P to be a first-order wff.

Temporal Logic proved to be suitable to express certain general properties of concurrent programs. However, as discussed in [Wo], we can easily imagine properties, particular to the concurrent program under investigation, that cannot be described using Temporal Logic. The solution proposed in [Wo] consisted in expanding the expressive power of Temporal Logic by adding a mechanism to define new modalities. The mechanism is based on right-linear grammars and was developed only for the propositional version of Temporal Logic.

The situation concerning transition constraints is entirely similar. Since we want to express properties intimately related to the enterprise being modelled, and not general properties of enterprises, we cannot expect to cover all situations with a small set of modalities. Therefore, we propose here to express transition constraints using Temporal Logic expanded along the lines proposed in [Wo]. However, unlike [Wo], we consider the full first-order-like version of the language and adopt a much more general mechanism to define modalities. This is essential to cover the wide range of transition constraints we expect to come up with in database modelling.

We close this introduction with a brief description of each section. Section 2 presents the family of temporal languages we propose to use and states the major results of the paper. Section 3 examines the decision problem of the propositional version of the languages. In fact, most of Section 3 is devoted to exhibiting a decision procedure for a certain subfamily of languages. Section 4 addresses the decision problem of the full version of the
2. Extended Temporal Logic

In this section we describe the family of formal languages we adopt to describe transition constraints and state the major results of the paper. We begin with a brief example to set the general scenario.

Let \( p \) stand for "John has now salary 10K", \( q \) stand for "John is now an employee" and \( r \) stand for "John has now salary less than 10K". Then, the constraint "if John has now salary 10K and continues to be an employee, then his salary must be at least 10K" can be rephrased as "there cannot be a sequence \( (S_0, \ldots, S_n) \) of database states such that \( p \) holds in \( S_0, q \) holds in \( S_1 \) through \( S_{n-1} \) and \( r \) holds in \( S_n \)." Or, putting it differently, no sequence of database states should satisfy some sequence of formulas of the form \( pq \ldots qr \) (for 0 or more \( q \)'s).

We now observe that the set of sequences of formulas of the form \( pq \ldots qr \) can be defined either by the grammar

\[
G = \langle \{G,H\}, \{p,q,r\}, \{G \to pH, H \to qH \mid r\}, G \rangle,
\]

or by the regular expression \( p; q^*; r \). Thus, we could succintly express the constraint in question as \( \neg (p; q^*; r) \), which should be understood as "no sequence of database states should satisfy some sequence of formulas in the set denoted by \( (p; q^*; r) \)". When the set is denoted by the grammar, we introduce a new ternary modality symbol \( g \) and express the constraint as \( \neg g(p, q, r) \), which is interpreted exactly as \( \neg (p; q^*; r) \), if we understand \( g(p, q, r) \) as denoting the set of all words generated from \( G \).

To summarize, we defined a transition constraint by matching sequences of database states against sequences of formulas taken from a set denoted by a grammar (or by a regular expression).

We now develop these ideas more precisely by taking formulas either from a given first-order language, or from a given propositional language.

Let \( L \) be a first-order language and \( G_1, \ldots, G_k \) be a set of grammars. The extended temporal language \( TL \) over \( L, G_1, \ldots, G_k \) (or the temporal extension of \( L \) over \( G_1, \ldots, G_k \)) is defined as follows. The symbols of \( TL \) are those of \( L \) plus a unary modality \( \circ \) ("next") and, for each non-terminal \( H \) of each grammar, an \( n_H \)-ary modality \( h \), where \( n_H \) is the number of terminals of
the grammar. The set of terms of $TL$ is exactly the set of terms of $L$, and
the set of well-formed formulas (wffs) of $TL$ is defined inductively as
follows:

(1) all wffs of $L$ are wffs of $TL$;

(2) if $P$ and $Q$ are wffs, then $(\neg P)$ and $(P \land Q)$ are wffs;

(3) if $P$ is a wff and $x$ is a variable of $L$, then $\forall xP$ is a wff;

(4) if $P$ is a wff, then $OP$ is a wff;

(5) if $h$ is an $n$-ary modality and $A_1, \ldots, A_n$ are wffs then $h(A_1, \ldots, A_n)$
is a wff.

We can also define variations of these languages as follows. If we
start with a propositional language $L$ and drop rule (3), we obtain a proposi-
tional temporal language. If we drop rule (3), but start with a first-
order language $L$, we obtain a restricted temporal language. Orthogonally, by
restricting the type of grammars to be right-linear, we obtain a right-linear
temporal language, and so on according to Chomsky's hierarchy (sec, e.g.,[AU]).
For example, the languages considered in [Wo] are what we would call right-
linear propositional temporal languages.

We might as well consider regular expressions instead of right-linear
grammars, which does not give us more powerful languages of course, but seems
to produce more readable formulas. Thus, a regular temporal extension $TL$ of
a first-order language $L$ is defined as follows. The symbols of $TL$ are those
of $L$, plus "o", ";", "u", "*". The set of wffs of $TL$ is defined as before,
except that rule (5) is replaced by:

(5') if $P$ and $Q$ are wffs of $TL$, then $(P;Q),(PuQ)$ and $P*$ are wffs of $TL$.

Let $L$ be a first-order language and $TL$ be the temporal extension of $L$
over a given set of grammars. A structure of $TL$ is a sequence $I=(I_0,I_1,\ldots)$
of structures of $L$ (the "database states") with the same domain (this re-
striction is somewhat important). An assignment of values to the variables of
$TL$ is a function $v$ that assigns to each variable of $TL$ a value taken from
the (common) domain. We extend $v$ to the terms of $TL$ as for first-order lan-
guages.
Given a structure $T = (I, I_1, \ldots)$ of $T_L$ and an assignment $v$ of values to the variables of $T_L$ from the common domain of $I$, we define $\models I P[v]$ (P is valid in $I$ for $v$) as follows:

1. If $P$ is first-order, then $\models I P[v]$ if $\models I_0 P[v]$
2. If $P$ is of the form $\neg Q$, then $\models I P[v]$ if not $\models I Q[v]$
3. If $P$ is of the form $Q \land R$ then $\models I P[v]$ if $\models I Q[v]$ and $\models I R[v]$
4. If $P$ is of the form $\forall x Q$, then $\models I P[v]$ if $\models I Q[u]$, for every assignment $u$ that differs from $v$ only on the value of $x$
5. If $P$ is of the form $\exists Q$, then $\models I P$ if $\models I^1 Q$, where $I^1 = (I_1, I_2, \ldots)$
6. If $P$ is of the form $h(Q_1, \ldots, Q_n)$, then $\models I P[v]$ if there is a word $w_{i_0} \ldots w_{i_k}$ generated from $H$ such that $\models I^j Q_{i_j}[v]$, for every $j \in [0, k]$, where $I^j = (I_{i_j}, I_{i_{j+1}}, \ldots)$

Note: The sentence $h(Q_1, \ldots, Q_n)$ implicitly establishes a one-to-one correspondence between formulas and terminals so that $Q_i$ corresponds to the $i$th terminal $w_i$. Thus, each word $w_{i_0} \ldots w_{i_k}$ corresponds to a sequence $Q_{i_0} \ldots Q_{i_k}$ of wffs. 

The notion of structure for propositional temporal languages is similarly defined, by making the necessary simplifications.
If $TL$ is a regular temporal language, then we have to adapt rule (6) appropriately. We then introduce the following definitions. If $P$ is a wff of the form $(R;Q)$, $(R u Q)$ of $R^*$, we say that $P$ is regular; otherwise, we say that $P$ is not regular (i.e., when $P$ is of the form $(-R)$, $(R \wedge Q)$, $\forall x R$, or $\exists R$). We say that $R$ is a component of $P$ iff (i) $R$ is $P$, or (ii) $P$ is of the form $(R;Q)$, $(Q;R)$, $(R u Q)$, $(Q u R)$ or $R^*$, or (iii) $R$ is a component of a component of $P$. Note that if $P$ is not regular, then it has just one component, which is itself.

Let $P$ be a regular wff. Define $A(P)$ as the set of all components of $P$ that are non-regular wffs. Then, we may view $P$ as a regular expression over the alphabet $A(P)$. Hence, $P$ defines a language over $A(P)$ (i.e., a set of finite sequences of elements of $A(P)$), that we denote by $L(P)$.

**Example 2.1:** Let $P$ be the following regular wff:

1. $(R;Q u (\forall x(T u U))^*)$

Then, the (immediate) components of $P$ are

2. $R;Q$ and $(\forall x(T u U))^*$

and the components of components of $P$ are

3. $R$, $Q$ and $(\forall x(T u U))$

Since $(\forall x(T u U))$ is not regular, $T$ and $U$ are not components of $P$. Therefore, we have that the alphabet associated with $P$ and the language generated by $P$ are:

4. $A(P) = \{R, Q, (\forall x(T u U))\}$

5. $L(P) = \{\lambda, RQ, (\forall x(T u U)), (\forall x(T u U))(\forall x(T u U)), \ldots\}$

With the help of these definitions, we define the semantics of a regular temporal language $TL$ just as before, except that rule (6) is replaced by:

6. If $P$ is a regular formula, then $\models_I P[v]$ iff there is a finite sequence $Q_0...Q_K$ in $L(P)$ such that $\models_I Q_j[v], \text{ for any } j \in [0,K]$.

Finally, we say that $P$ is valid ($\models P$) iff $\models_I P[v]$, for every $I$ and $v$. We say that $P$ is satisfiable iff there is $I$ and $v$ such that $\models_I P[v]$, in which case $I$ is said to be a model for $P$. The validity problem for a class $C$ of languages is the following problem: "Does there exist an algorithm (Turing Machine) that takes any wff $P$ of any language in $C$ as input and always halts with a correct yes ($P$ is valid) or no ($P$ is not valid) answer?". The satisfiability problem is similarly defined.
At this point we observe that our languages do not contain the standard modalities \( \Diamond Q \) ("eventually Q will be true"), \( \Box Q \) ("henceforth Q will always be true") and \( PUQ \) ("henceforth P will always be true until Q is true"). They can be introduced by definition as follows:

1. \( \Diamond Q \equiv g(\text{true}, Q) \)
2. \( \Box Q \equiv \neg g(\text{true}, \neg Q) \)
3. \( PUQ \equiv g(P, Q) \)

where \( g \) is induced by the following grammar

\[
G = (\{G\}, \{a_1, a_2\}, \{G \rightarrow a_1G, G \rightarrow a_2\}, G)
\]

In fact, the modality \( \diamond \) (next) could also be introduced by definition. But, in view of Section 3, it is not convenient to do it so.

To see that these definitions agree with the intuition, consider (3), for example. Given a structure \( I = (I_0, I_1, \ldots) \) it says that there is a sequence \( P \ldots PQ \) such that \( P \) is valid in \( I^0 \) through \( I^3 \) and \( Q \) is valid in \( I^4 \). That is, \( P \) is valid until \( Q \) is valid. Likewise, (1) says that there is a sequence \( \text{true} \ldots \text{true} Q \) such that \( \text{true} \) is valid in \( I^0 \) through \( I^3 \) and \( Q \) is valid in \( I^4 \). That is, there is some \( I^k \) where \( Q \) is valid, or eventually \( Q \) is valid.

Using regular temporal logic, these definitions would go as follows:

1. \( \Diamond Q \equiv \text{true}^*; Q \)
2. \( \Box Q \equiv \neg(\text{true}^*; \neg Q) \)
3. \( PUQ \equiv P^*; Q \)

**Note:** \( a^* \) denotes the set of finite words consisting of just \( a \)'s; the infinite word \( aa \ldots \) is not in the set denoted by \( a^* \) (see [AU]).

The reader is invited to verify that these definitions agree with the intuition.

We now discuss in detail a few transition constraints. Our examples will be based on a first-order language \( L \), which supposedly describes a small company. The language \( L \) has three binary predicate symbols, EMP, ASSIGN and \(<\).
A wff \( \text{EMP}(n,s) \) indicates that employee \( n \) has salary \( s \); \( \text{ASSIGN}(n,p) \) indicates that employee \( n \) is assigned to project \( p \); and \( s \succ s' \) indicates that \( s \) is greater than \( s' \), as usual. To describe transition constraints, we will use the regular extension \( TL \) of \( L \).

Consider first the constraint "salaries never decrease". Let \( S=(S_0, S_1, \ldots) \) be a sequence of database states, where \( S_0 \) is the initial database state. The sequence \( S \) is unacceptable if there is \( i \geq 0 \) such that \( \text{EMP}(n,s) \) holds in \( S_i \) and \( (\text{EMP}(n,s') \land s \succ s') \) holds in \( S_{i+1} \). Thus, the constraint can be expressed as

\[
(1) \quad \neg \exists n \exists s (\Diamond (\text{EMP}(n,s) \land \exists s' ((\text{EMP}(n,s') \land s \succ s'))) )
\]

The wff in (1), when translated back into English, reads "it is false that there is an employee \( n \) and salaries \( s \) and \( s' \) such that eventually \( n \) has salary \( s \) in one state and salary \( s' \) less than \( s \) in the next state".

Note that, in our formalisation, an employee can be fired and re-admitted with a lower salary. If we understand "salaries never decrease" as ruling out this situation, then we have to give an alternative formalization. Let \( S=(S_0, S_1, \ldots) \) again be a sequence of database states. The sequence \( S \) is now unacceptable if there are \( i \geq 0 \) and \( j > i \) such that \( \text{EMP}(n,s) \) holds in \( S_i \) and \( (\text{EMP}(n,s') \land s \succ s') \) holds in \( S_j \). Thus, the formalisation now is:

\[
(2) \quad \neg \exists n \exists s (\Diamond (\text{EMP}(n,s) \land \Diamond (\exists s' ((\text{EMP}(n,s') \land s \succ s')) ))
\]

We can also give a third interpretation to "salaries never decrease". We can take it to mean that "once an employee is admitted and as long as he is continuously working for the company, his current salary is never below his salary at the time he was admitted". (This formulation perhaps takes into account seasonal commissions that are added to the salary). The formalisation of "salaries never decrease" would then forbid a sequence \( S=(S_0, S_1, \ldots) \) where there are \( i > 0 \) and \( j > i \) such that \( \text{EMP}(n,s) \) holds in \( S_i \), but \( \neg \exists s '' \text{EMP}(n,s'') \) holds in \( S_{i-1} \) (i.e., \( n \) was admitted at time \( i \), \( \text{EMP}(n,s'') \) holds in \( S_{i+1} \) till \( S_j \) (i.e., \( n \) was an employee from \( i+1 \) till \( j \)) and \( (\text{EMP}(n,s') \land s \succ s') \) holds in \( S_j \) (i.e., \( n \) has salary less than \( s \) at time \( j \)).

Thus, the formalisation is:

\[
(3) \quad \neg \exists n \exists s (\Diamond (\neg \exists s '' \text{EMP}(n,s'') ; \text{EMP}(n,s) ; (\exists s '' \text{EMP}(n,s'') ) \land \exists s' ((\text{EMP}(n,s') \land s \succ s')) )
\]
As another example, consider the constraint "employees that are assigned to a project cannot be fired". It can be expressed as:

\[(4) \quad \neg \exists n (\exists s \text{ EMP}(n,s) \land \exists p \text{ ASSIGN}(n,p)) \land \neg \exists s' \text{ EMP}(n,s'))\]

Again, note that the ambiguity of natural languages is avoided. In (4) we do not say that an employee who is presently assigned to a project can never be fired, but that he cannot be fired without previously cancelling all his assignments to projects.

To conclude the examples, we observe that triggers [ES] can also be specified as transition constraints of the form

\[(5) \quad \Box (P \Rightarrow Q)\]

which says that whenever P becomes true, Q must be true in the next state. In an implementation-oriented context, triggers indicate that some action must take place when a condition P holds, the goal of the action being to make Q hold. In the present discussion, however, we are not concerned with operational aspects and even less with the mechanisms (e.g. monitors) involved. Thus, triggers are here specified as transition constraints.

We close this section with a list of results about the decision problem of extended temporal languages and their implications. The results will be proved in later sections.

For the class of propositional temporal languages, we offer the following results.

**THEOREM 1:** The validity problem for the class of propositional temporal languages with one right-linear grammar and one context-free grammar is undecidable. \(\Box\)

**THEOREM 2:** The validity problem for the class of right-linear propositional temporal languages is decidable in exponential time. \(\Box\)

Thus, there is a sharp contrast when we move from extensions using only right-linear grammars to extensions allowing context-free grammars. This fact is important in so far as it limits the expressiveness of the languages, if decidability must be retained. It should be noted that Theorem 2 is essentially contained in [Wo], except that the extensions considered there allow infinite words in the language generated by grammars, whereas we stay with
[AU] and allow only finite words. This creates an interesting duality in the metatheory of the languages, which we do not explore here.

For regular temporal logic (i.e., the extension of first-order languages using regular expressions), we have:

**THEOREM 3**: The validity problem for regular temporal languages is not partially solvable. □

Theorem 3 has an important consequence, that we state as a corollary.

**COROLLARY 1**: There is no consistent and complete axiom system for regular temporal languages. □

A second rather obvious consequence is that the validity problem for extended temporal languages using any type of grammars is not partially solvable, as long as the grammars are at least right linear.

3. The Decision Problem for Extended Propositional Temporal Languages

We prove in this section two results about extended propositional temporal languages. Together they imply that decidability is retained if and only if at most right-linear grammars are allowed. The proof of the first result goes as follows.

**THEOREM 1**: The validity problem for the class of propositional temporal languages extended with one right-linear grammar and one context-free grammar is undecidable.

**Proof**

We reduce the problem of deciding if a context-free grammar and a right-linear grammar generate the same language to the problem in question. Since the former problem is undecidable [AU], so is ours.

Let $G'_1$ be a context-free grammar and $G'_2$ be a right-linear grammar. Without loss of generality, we may assume that $G'_1$ and $G'_2$ have the same set $\Sigma' = \{v_1, \ldots, v_n\}$ of terminals. Construct now two other grammars $G_i$ (i=1,2) such that: (i) the start symbol of $G_i$ is $S_i$; (ii) the set of terminals of
G_1 is \( \Sigma = \Sigma' \cup \{v_0\} \) (we assume that \( v_0 \not\in \Sigma' \)); (iii) \( w \in L(G_1') \) iff \( wv_0 \in L(G_1) \); (iv) \( G_1 \) is context-free and \( G_2 \) is right-linear. (Note that conditions (iii) and (iv) are not contradictory). Then, we trivially have:

\[
(1) \quad L(G_1') = L(G_1') \quad \text{iff} \quad L(G_1) = L(G_2)
\]

Let \( L \) be a propositional language with \( n+1 \) propositional symbols \( p_0, \ldots, p_n \). Let \( TL \) be the extension of \( L \) via \( G_1 \) and \( G_2 \). Let \( s_i (i=1,2) \) be the \((n+1)\)-ary modality corresponding to \( S_i \). We show that

\[
(2) \quad \models s_1(p_0, \ldots, p_n) \equiv s_2(p_0, \ldots, p_n) \quad \text{iff} \quad L(G_1) = L(G_2)
\]

This suffices to establish our result since, by (1) and (2), we reduced the problem of testing if a context-free grammar \( G_1' \) and a right-linear grammar \( G_2' \) generate the same language to testing the validity of a wff of a language in the class of propositional temporal languages extended with a right-linear and a context-free grammar.

(\( \Rightarrow \)) Assume that \( L(G_1) = L(G_2) \). Then, the result follows trivially, by definition of validity.

(\( \Rightarrow \)) Assume that \( \models s_1(p_0, \ldots, p_n) \equiv s_2(p_0, \ldots, p_n) \)

that is, assume that

\[
(3) \quad \text{for any structure } I \text{ of } TL, \models_I s_1(p_0, \ldots, p_n) \quad \text{iff} \quad \models_I s_2(p_0, \ldots, p_n)
\]

We first show that \( L(G_1) \subseteq L(G_2) \). Let \( w = v_0 v_1 \cdots v_n v_0 \in L(G_1) \). Let \( \bar{w} = p_0 p_1 \cdots p_n p_0 \) be the sequence of propositional symbols corresponding to \( w \). Construct a structure \( I = (I_0, I_1, \ldots) \) of \( TL \) as follows:

(i) for each \( j \in [0, \ell] \), \( I_j(p_{ij}) = \text{true} \) and \( I_j(q) = \text{false} \), for any propositional symbol \( q \) other than \( p_{ij} \);

(ii) \( I_{\ell+1}(p_0) = \text{true} \) and \( I_{\ell+1}(q) = \text{false} \), for any propositional symbol \( q \) other than \( p_0 \);

(iii) for each \( j > \ell+1 \), \( I_j(q) = \text{false} \), for any propositional symbol \( q \).
Now, by construction of $I$, $I^e_{s_1}(p_0, \ldots, p_n)$. Hence, by (3), $I^e_{s_2}(p_0, \ldots, p_n)$. Therefore, there is $u = v_{k_0} \ldots v_{k_m} v_0 \in L(G_2)$ such that $I^e_{j}(p_{k_j}) = \text{true}$, for any $j \in [0, m]$, and $I^e_{m+1}(p_0) = \text{true}$. But, by construction of $I$, $I^e_{j}(p_{k_j}) = \text{true}$ iff $k_j = i_j$, for any $j \in [0, \min(\ell, m)]$, and $I^e_{m+1}(p_0) = \text{true}$ iff $\ell = m$. Hence, we may conclude that $w = u$. Therefore, we have that $w \in L(G_2)$. Thus, $L(G_1) \subseteq L(G_2)$. Likewise, we can prove that $\overline{L(G_2)} \subseteq \overline{L(G_1)}$, which permits us to conclude that $L(G_1) = L(G_2)$, as was to be shown. □

We now turn to the second result. We will prove that the satisfiability problem for right-linear propositional temporal languages is decidable in exponential time. Hence, the validity problem is also decidable in exponential time.

Throughout the rest of this section we will use only right-linear grammars, so we may assume that all productions are of the form $G \rightarrow aH$ or $G \rightarrow a$.

The procedure we will describe to decide satisfiability for right-linear propositional temporal logic is a generalization of the analytic tableau method for propositional calculus [Sm] (for a brief description of the method see Appendix A). Given a wff $P$, the procedure starts a systematic search for a model of $P$ by decomposing $P$ into its atomic components. The procedure then either reaches contradictions, in which case $P$ is unsatisfiable, or constructs a sequence of sets of wffs that can be used to build a model for $P$. The sequence must have a special property, which we now define.

**DEFINITION 3.1:** A sequence $H = (H_0, H_1, \ldots)$ of sets of wffs of a right-linear propositional temporal language $TL$ is a Hintikka sequence for $TL$ iff for each $i \geq 0$, we have

(i) $H_i$ contains no wff and its negation;

(ii) if $Q$ is in $H_i$ and if there is a rule $R_j$ in Figure 3.1 with antecedent $P_j$ and consequents $Q_{j1}, \ldots, Q_{jn_j}$ such that $P_j$ matches $Q$, then all wffs in $Q_{jk}$ occur in $H_i$, for some $k \in [1, n_j]$;

(iii) if $\neg Q$ is in $H_i$, then $Q$ is in $H_{i+1}$;

(iv) for each wff in $H_i$ of the form $g(A_1, \ldots, A_n)$, there is a finite sequence $A_{\ell_0} \ldots A_{\ell_m}$ such that $A_{\ell_r}$ is in $H_{i+r}$, $0 \leq r \leq m$, and $A_{\ell_0} \ldots A_{\ell_m}$ is a word generated from $G$, the non-terminal corresponding to $g$. □
Rules for Right-Linear Propositional Temporal Logic

\[ \text{A-rules: } \frac{A}{A_1, A_2} \quad \text{B-rules: } \frac{B}{B_1 | B_2} \]

(as for Propositional Calculus - see Appendix A)

\[ \text{\( o \)-rule: } \frac{\neg o \, P}{o \, \neg P} \]

\[ \text{g-rules: } \frac{g_i(A_1, \ldots, A_n)}{A_{k_1}, o \, g_{\ell_1}(A_1, \ldots, A_n) | \ldots | A_{k_m}, o \, g_{\ell_m}(A_1, \ldots, A_n)} \]

\[ \frac{\neg g_i(A_1, \ldots, A_n)}{\neg A_{k_1} \lor \neg \, o \, g_{\ell_1}(A_1, \ldots, A_n), \ldots, \neg A_{k_m} \lor \neg \, o \, g_{\ell_m}(A_1, \ldots, A_n)} \]

where \( g_{i_j} (j = 1, \ldots, m) \) are all productions whose left-hand side

is \( g_i \), the non-terminal corresponding to the modality \( g_i \)

FIGURE 3.1
EXAMPLE 3.1:

Let $TL$ be a right-linear propositional temporal language whose only grammar is $G = \{(G_0, G_1), \{a, b, c\}, \{G_0 \rightarrow aG_1, \; G_1 \rightarrow bG_1 \mid c\}, \emptyset\).

Then, the rules for the modality $g_0$ corresponding to $G_0$ are:

\[
\frac{g_0(p, q, r)}{\neg g_0(p, q, r)}
\]

and the rules for the modality $g_1$ corresponding to $G_1$ are:

\[
\frac{g_1(p, q, r)}{q, \neg g_1(p, q, r) \mid r}
\]

\[
\frac{\neg g_1(p, q, r)}{\neg q \vee \neg g_1(p, q, r), \neg r}
\]

A Hintikka sequence over $TL$ would then be: $H = (H_0, H_1, H_2)$, where

$H_0 = \{p \vee q, \; g_0(p, q, r), \; p, \; \neg g_1(p, q, r)\}$

$H_1 = \{g_1(p, q, r), \; q, \; \neg g_1(p, q, r)\}$

$H_2 = \{g_1(p, q, r), \; r\}$

Note that, for example, $g_0(p, q, r)$ in $H_0$ satisfies condition (iv) since $p$ occurs in $H_0$, $q$ occurs in $H_1$ and $r$ occurs in $H_2$ and, moreover, the word $abc$ is generated from $G_0$ in $G$ (the word $abc$ corresponds to the sequence of wffs $pqr$). [\]

Let $H = (H_0, H_1, \ldots)$ be a Hintikka sequence over a language $TL$. We say that $H$ is satisfiable iff there is a structure $I = (I_0, I_1, \ldots)$ of $TL$ such that $I^i$ satisfies all wffs in $H_i$, for each $i \geq 0$, where $I^i$ is the sub-sequence of $I$ starting on $I_i$. The fundamental property of Hintikka sequences is stated in the following lemma.

**Lemma 1:** Any Hintikka sequence is satisfiable.
Sketch of Proof

Let $H = (H_0, H_1, \ldots)$ be a Hintikka sequence over $TL$. Construct a structure $I = (I_0, I_1, \ldots)$ of $TL$ such that, for each $i \geq 0$, for each propositional symbol $p$ of $TL$, $I_i(p) = \text{true}$ iff $p$ occurs in $H_i$. Then, we can prove by induction on the height of a wff that $I$ satisfies $H$. □

We now turn to another important notion. Recall that our decision procedure is a systematic search for a model of a wff $P$. During the search wffs are decomposed into atomic components so that either contradictions are identified or a Hintikka sequence is constructed. The wffs obtained during the search are organized as a data structure called a full tableau for $P$, which we now define.

Given a set $Q$ of wffs, the $\circ$-reduction of $Q$ is the set of all wffs $Q'$ such that $\circ Q$ is in $Q$. The $\circ$-reduction of a sequence of sets of wffs is the union of the $\circ$-reductions of the sets. A sequence of sets of wffs is open iff it contains no wff and its negation.

DEFINITION 3.2:

(a) The set of partial tableaux for a wff $P$ of right-linear propositional temporal logic consists of trees whose nodes are sets of wffs. It is defined as for propositional calculus (see Definition A.1), but using the rules in Figure 3.1. Whenever a rule is applied to add new leaves to a tableau we say that the tableau was extended; if the new leaves contain some wff not in their ancestors, we say that the tableau was extended without repetition.

(b) The set of full tableaux for a wff $P$ of right-linear propositional temporal logic consists of pairs $FT = (N, E)$ where $N$ is a set of partial tableaux and $E$ is a set of pairs of sets of wffs. It is defined inductively as follows:

(i) $([T], \emptyset)$ is a full tableau for $P$, where $T$ is a tree whose only node is the root, which is $\{P\}$.

(ii) Let $FT = (N, E)$ be a full tableau for $P$. Suppose that some partial tableau $T$ in $N$ can be extended without repetition into a partial tableau $T'$. 
Then, $FT' = (N',E)$ is a full tableau for $P$, where $N'$ is $N$ with $T$ replaced by $T'$.

(iii) Let $FT = (N,E)$ be a full tableau for $P$. Suppose that no partial tableau in $N$ can be extended further without repetition, but there is a partial tableau $T \in N$ with an open branch $\beta$ whose $o$-reduction $P'$ is not empty. Let $\lambda$ be the last node of $\beta$. Then $FT' = (N',E')$ is a full tableau for $P$, where $E' = E \cup \{(\lambda,P')\}$ and $N' = N$, if $P'$ is the root of a tableau in $N$, or $N' = N \cup \{(P')\}$, otherwise. In both cases, $\beta$ is said to generate the partial tableau whose root is $P'$.

(c) A full tableau is finished iff it cannot be extended further by the above process. □

A full tableau $FT = (N,E)$ can be viewed as a set of partial tableaux with some added structure given by $E$. Thus, we will use the following terminology. A set $\eta$ of wffs is a root (leaf) of $FT$ iff $\eta$ is the root (leaf) of some tableau in $N$; a sequence $\bar{\eta} = (\eta_0, \ldots, \eta_n)$ of sets of wffs is a branch of $FT$ iff $\bar{\eta}$ is a branch of some tableau in $N$.

We may also view a tableau $FT = (N,E)$ as a graph with special arcs (those in $E$) and with distinguished subgraphs (the partial tableau in $N$). We refrained from defining full tableaux as graphs, though, simply because it is useful to regard the partial tableaux in $N$ as existing independently of $FT$. However, we will use the following graph-theoretical terminology from now on. An arc of $FT$ is either an arc of a tableau in $N$ or a pair in $E$. We say that a set $\eta$ of wffs is a successor of a set $\eta'$ of wffs in $FT$ iff $(\eta', \eta)$ is an arc of $FT$. A sequence $(\eta_0, \eta_1, \ldots)$ of sets of wffs is a path in $FT$ iff $\eta_i$ is a successor of $\eta_{i-1}$ in $FT$, for each $i > 0$.

A full tableau $FT=(N,E)$ for $P$ organizes the search for a model for $P$ in stages, in the following sense. The first stage of the search consists of constructing a partial tableau $T_0$ whose root contains just $P$. This stage ends when $T_0$ cannot be extended further without repetition. Each open branch $\beta$ of $T_0$ with $o$-wffs gives rise to a tableau belonging to the second stage, and so on. The relationship between tableaux at the $i$th stage and branches of tableaux at the $(i-1)$th stage is indicated by $E$. This is illustrated by the following example.
EXAMPLE 3.2: An example of a full tableau.

Let \( P = (A \Rightarrow B) \land g_0(A, B, \neg C) \land \Box (B \Rightarrow \Box C) \) be a wff of \( T^I \), the language used in Example 3.1. A full tableau \( FT = (N, E) \) for \( P \) is given below:

\[
\begin{align*}
(A \Rightarrow B) \land g_0(A, B, \neg C) \land \Box (B \Rightarrow \Box C) \\
(A \Rightarrow B), \ g_0(A, B, \neg C), \ \Box (B \Rightarrow \Box C) \\
A, \ \Box g_1(A, B, \neg C) \\
\neg A, \ B \\
B \Rightarrow \Box C, \ \Box \Box (B \Rightarrow \Box C) \\
\neg B, \ \Box C \\
g_1(A, B, \neg C), \ \Box (B \Rightarrow \Box C), \ C \\
\neg C, \ B, \ \Box g_1(A, B, \neg C) \\
B \Rightarrow \Box C, \ \Box \Box (B \Rightarrow \Box C) \\
\neg B, \ \Box C
\end{align*}
\]

Note: The arcs denoted by double arrows correspond to pairs in \( E \).

Given a wff \( P \), Definition 3.2 always induces a finished full tableau for \( P \) with at most an exponential number of nodes. However, it is not entirely obvious how one can tell if \( P \) is satisfiable or not by looking at a finished full tableau for \( P \). For instance, the full tableau in Example 3.1 contains a loop involving no wff and its negation. So, apparently, we could use this fact to build a model for \( P \). However, the loop contains a wff, \( g_1(A, B, \neg C) \), which is unsatisfiable in the presence of the other wffs in the loop. This observation is not obvious and depends on a careful analysis of the semantics of \( g_1(A, B, \neg C) \). In fact, most of the rest of the section is devoted to a clarification of this point.
DEFINITION 3.3: Let $FT = (N,E)$ be a full tableau for a wff $P$. Let $Q$ and $Q'$ be occurrences of wffs in $FT$. Then, we say that $Q'$ is derived from $Q$ iff one of the two conditions below hold:

(i) $Q'$ belongs to a node of $FT$ that was created as a result of applying some rule in Figure 3.1 to $Q$;

(ii) $Q$ is $\neg Q'$, and $Q$ occurs in an open branch $\beta$, and $Q'$ belongs to a root $\omega$ of $FT$ that was created as a result of applying step (iii) of Definition 3.2 to $\beta$. □

DEFINITION 3.4: Let $FT = (N,E)$ be a full tableau for a wff $P$.

(a) the primary root of $FT$ is the node $\{P\}$ that started the construction of $FT$.

(b) an extended branch of $FT$ is either a simple path (a path with no repeated nodes) from the primary root of $FT$ to a leaf of $FT$, or a simple path $\pi$ from the primary root of $FT$ to a node $\eta$ of $FT$ such that some node in $\pi$ is a successor of $\eta$ (i.e., $\pi$ ends "in a loop"). □

In Example 3.1, the simple paths from the primary root ending on $\neg A$ or $\neg B$ or $\neg C$ are extended branches, as is the path ending on $\neg C$.

Recall that, if we write $g_i(A_1, \ldots, A_n)$, then the modality $g_i$ corresponds to the non-terminal $G_i$ and the wff $A_j$ corresponds to the $j$th terminal, which we denote here by $w_j$, for $j=1, \ldots, n$.

DEFINITION 3.5: Let $\pi$ be an extended branch of $FT$.

An occurrence $Q$ in $\pi$ of a wff of the form $g_i(A_1, \ldots, A_n)$ is accomplished in $\pi$ iff:

(i) either there is a production rule of the form $G_i \rightarrow w_j$, and $\pi$ has an occurrence of $A_j$ which is derived from $Q$,

(ii) or there is a rule of the form $G_i \rightarrow w_j G_k$ and $\pi$ has occurrences $Q'$ and $Q''$ of wffs of the form $A_j$ and $\neg g_k(A_1, \ldots, A_n)$, respectively, which are derived from $Q$ and, moreover, there is an occurrence $Q'''$ in $\pi$ of a wff of the form $g_k(A_1, \ldots, A_n)$ such that $Q'''$ is derived from $Q''$ and $Q'''$ is accomplished in $\pi$. □

Note then that if $g_i(A_1, \ldots, A_n)$ is accomplished in $\pi$, then $\pi$ induces a word $w_i w_{i_1} \ldots w_{i_m}$ generated from $G_i$. 


DEFINITION 3.6: Let $FT = (N,E)$ be a full tableau for $P$.

(a) An extended branch $\pi$ of $FT$ is fulfilled iff

(i) $\pi$ contains no wff and its negation;
(ii) if there is an occurrence $Q$ in $\pi$ of a wff of the form $g_1(A_1, \ldots, A_n)$ then $Q$ is accomplished in $\pi$.

Otherwise, we say that $\pi$ is unfulfilled.

(b) We say that $FT$ is closed iff every extended branch of $FT$ is unfulfilled.

EXAMPLE 3.3: The full tableau of Example 3.2 is closed for the following reasons:

(i) there are five extended branches, each ending on $\neg A$ or $\neg B$ or $\neg C$ or $\neg \neg C$;
(ii) those ending on $\neg A$ or $\neg B$ or $\neg C$ are all unfulfilled since they contain contradictions, that is, a wff and its negation;
(iii) the extended branch ending on $\neg \neg C$ is also unfulfilled since the last occurrence of $g_1(A,B,\neg C)$ is not accomplished. To see why, observe that the last occurrence of $g_1(A,B,\neg C)$ does not satisfy either clause (i) or clause (ii) of Definition 3.5.

We can now give a concise description of the procedure $\mathcal{P}$ deciding satisfiability for right-linear propositional temporal logic. The procedure $\mathcal{P}$ starts with a wff $P$ and nondeterministically constructs a full tableau for $P$. The procedure stops when the full tableau is finished or closed, and outputs "NO", if the tableau is closed, and "YES", otherwise. Since a full tableau for $P$ has at most an exponential number of nodes, the procedure takes at most exponential time.

We now show that $\mathcal{P}$ works correctly.

LEMMA 2: If $\mathcal{P}$ stops with 'NO', when started with $P$, then $P$ is unsatisfiable.

Sketch of Proof

If suffices to show that if $P$ is satisfiable no full tableau for $P$ is closed. This can be done by induction on the number of nodes of the full tableau, using previous definitions.
LEMMA 3: Iff \( P \) stops with 'YES', when started with \( P \), then \( P \) is satisfiable.

**Sketch of Proof**

Assume that \( P \) stops with 'YES' when started with \( P \). Let \( FT = (N,E) \) be the full tableau for \( P \) constructed by \( P \). We first observe that, since \( P \) does not stop with 'NO', \( FT \) is not closed and, hence, it has an extended branch \( \pi \) which is fulfilled. We can then show that \( \pi \) induces a Hintikka sequence \( H = (H_0, H_1, \ldots) \) such that \( P \in H_0 \). Hence, by Lemma 1, \( H \) and so \( P \) are satisfiable. \( \square \)

**THEOREM 2:** The satisfiability problem for right-linear propositional temporal languages is decidable in exponential time.

**Sketch of Proof**

Follows from Lemmas 2 and 3 and the definition of full tableaux \( \square \)

To summarize the results in this section, we proved that satisfiability and, so, validity for right-linear propositional temporal languages is decidable in exponential time. However, validity is undecidable, if we allow extensions that use at least one right-linear and one context-free grammar.

This concludes our discussion on extended propositional temporal languages.

4. The Decision Problem for Regular Temporal Languages

We prove in this section that the validity problem of regular temporal languages is not partially solvable. To prove this result, we show that, for any fixed regular program schema \( r \), there is a wff \( P_r \) such that \( P_r \) is valid iff \( r \) never halts for any interpretation and any initial state. Since the diverge problem for regular program schemes is not partially solvable, then so is the validity problem of regular temporal languages. We also show that, as a consequence, there is no consistent and complete axiom
system for regular temporal languages.

We begin with a brief discussion about regular program schemes. Let \( L \) be a first-order language. The set \( R \) of regular program schemes over \( L \) (or, simply, programs) is defined inductively as follows:

1. If \( x \) is a variable, \( t \) a term and \( B \) an atomic wff of \( L \), then \( x := t \) and \( B? \) are programs called, respectively, an assignment and a test;
2. If \( r \) and \( s \) are programs, then \( r^* \), \( r \cup s \) and \( r; s \) are also programs.

An interpretation for a program \( r \) over \( L \) is simply an interpretation \( \mathcal{A} \) for \( L \). A state for \( L \) and \( \mathcal{A} \) is an assignment of values from the domain of \( \mathcal{A} \) to the variables of \( L \). The universe \( U \) of \( L \) and \( \mathcal{A} \) is the set of all states of \( L \) and \( \mathcal{A} \).

Given the universe \( U \) of \( L \) and \( \mathcal{A} \), we define a function \( m^r_\mathcal{A} : R \rightarrow 2^U \) associating a relation \( m^r_\mathcal{A}(r) \subseteq U^2 \) with each program \( r \in R \). The function \( m^r_\mathcal{A} \) is defined inductively as follows:

1. \( m^x_\mathcal{A}(x := t) = \{(v, v') \in U^2 / v'(x) = \mathcal{A}(t) \} \) and \( v'(y) = v(y) \), for any variable other than \( x \)

Note: \( v \) denotes the extension of \( v \) to the terms of \( L \), using \( \mathcal{A} \).

2. \( m^B_\mathcal{A}(B?) = \{(v, v) \in U^2 / \mathcal{A} B[v] \} \)

3. \( m^{r^*}_\mathcal{A}(r^*) = (m^r_\mathcal{A}(r))^* \) - the reflexive and transitive closure of \( m^r_\mathcal{A}(r) \)

4. \( m^{r \cup s}_\mathcal{A}(r \cup s) = m^r_\mathcal{A}(r) \cup m^s_\mathcal{A}(s) \) - the union of \( m^r_\mathcal{A}(r) \) and \( m^s_\mathcal{A}(s) \)

5. \( m^{r; s}_\mathcal{A}(r; s) = m^r_\mathcal{A}(r) \cdot m^s_\mathcal{A}(s) \) - the composition of \( m^r_\mathcal{A}(r) \) and \( m^s_\mathcal{A}(s) \)

Now we say that a program \( r \) over \( L \) diverges under interpretation \( \mathcal{A} \) iff \( m^r_\mathcal{A}(r) = \emptyset \). The divergence problem for regular program schemes is: "Does there exist an algorithm that takes any program \( r \) as input and always halts with a correct "YES" (\( r \) diverges for every interpretation) or "NO" (\( r \) does not diverge for every interpretation) answer".
LEMMA 4: The divergence problem for regular program schemes is not partially solvable.

Sketch of Proof

There is a straightforward reduction of the divergence problem in question to the divergence problem for program schemes, which is not partially solvable [Ma]. □

We now show that the validity problem for regular temporal languages is also not partially solvable by reducing the divergence problem to it. The validity problem is: "Does there exist an algorithm that takes any wff $P$ of any regular temporal language as input and always halts with a correct "YES". $(P$ is valid for every interpretation) or "NO" $(P$ is not valid for every interpretation) answer".

THEOREM 3: The validity problem for regular temporal languages is nor partially solvable.

Proof

We will reduce the divergence problem for regular program schemes to the validity problem in question. Since, by Lemma 4, the former problem is not partially solvable, so is the latter problem.

Given a regular program scheme $r$, we construct a wff $\neg P_r$ of a regular temporal language such that $\neg P_r$ is valid iff $r$ diverges for any interpretation or, equivalently, $P_r$ is satisfiable iff there is an interpretation $A$ of $r$ such that $m_A(r) \neq \emptyset$.

Let $x_1, \ldots, x_k$ be the variables occurring in $r$, let $f_1, \ldots, f_\ell$ be the function symbols occurring in $r$ and let $p_1, \ldots, p_m$ be the predicate symbols occurring in $r$. Let $L$ be the first-order language whose non-logical symbols are exactly $f_1, \ldots, f_\ell$, $p_1, \ldots, p_m$ and whose variables are $x_1, \ldots, x_k$. Then $r$ can be considered as a program over $L$. Let $L'$ be another first-order language whose non-logical symbols are $f_1, \ldots, f_\ell$, $p_1, \ldots, p_m$, plus a set of constants $c_1, \ldots, c_k$. We interpret $c_i$ as corresponding to $x_i$ in the following sense. Given a structure $A$ of $L$ and an assignment $v$ of values from the domain of $A$ to $x_1, \ldots, x_k$, we denote by $A_v$ the structure of $L'$ such that $A_v(f_i) = A(f_i)$, $1 \leq i \leq \ell$, $A_v(p_i) = A(p_i)$, $1 \leq i \leq m$, and $A_v(c_i) = v(x_i)$, $1 \leq i \leq k$. The wff $P_r$ correspond-
ing to \( r \) will be a wff in the regular temporal language \( TL' \) extending \( L' \).

Before constructing \( P_\_r \), we introduce some auxiliary wffs. Let \( A \) be the following wff

\[
\begin{align*}
\wedge_{j=1}^{k} & \exists y_j \ (c_j = y_j \wedge \circ c_j = y_j) \\
\wedge_{j=1}^{j \neq i} & \exists y_j \ (c_j = y_j \wedge \circ c_j = y_j)
\end{align*}
\]

This formula is satisfiable by a structure \( I=(I_0, I_1, \ldots) \) of \( TL' \) iff the value of \( c_j \), \( 1 \leq j \leq k \), is the same in \( I_0 \) and \( I_1 \). Likewise, let \( A_i \) be the wff below

\[
\begin{align*}
\wedge_{j=1}^{i \neq k} & \exists y_j \ (c_j = y_j \wedge \circ c_j = y_j) \\
\end{align*}
\]

which is satisfiable by \( I \) iff the value of \( c_j \), \( 1 \leq j \leq \ell \), is the same in \( I_0 \) and \( I_1 \). Let \( B \) be the wff

\[
\begin{align*}
\wedge_{j=1}^{\ell} & \forall z_j \forall y_j \ (f_j(z_j) = y_j \iff \circ f_j(z_j) = y_j)
\end{align*}
\]

which is satisfiable by \( I \) iff the value of \( f_j \), \( 1 \leq j \leq \ell \), is the same in \( I_0 \) and \( I_1 \) (note that \( I_0 \) and \( I_1 \) have the same domain, by definition). Finally, let \( C \) be the wff

\[
\begin{align*}
\wedge_{i=1}^{m} & \forall \bar{z}_i \ (p_i(\bar{z}_i) \iff \circ p_i(\bar{z}_i))
\end{align*}
\]

which is satisfiable by \( I \) iff the value of \( p_j \), \( 1 \leq j \leq m \), is the same in \( I_0 \) and \( I_1 \) (again the fact that \( I_0 \) and \( I_1 \) have the same domain is important).

We now define \( P_\_r \) by induction on the structure of \( r \):

(a) if \( r \) is \( x_i : = t \), then \( P_\_r \) is

\[
A_i \land B \land C \land \exists y_i \ (y_i = t[c_i/x_i, \ldots, c_k/x_k] \land \circ y_i = c_i)
\]

which is satisfiable in \( I=(I_0, I_1, \ldots) \) iff the value of \( c_i \) in \( I_1 \) is equal to the value of \( t[c_i/x_i, \ldots, c_k/x_k] \) in \( I_0 \), and the value of all other symbols are the same in \( I_0 \) and \( I_1 \).
(b) if \( r \) is \( Q \), then \( P_r \) is \( A \land B \land C \land Q[c_1/x_1, \ldots, c_k/x_k] \)

which is satisfiable by \( I \) iff the value of all symbols are the same in \( I_0 \)
and \( I_1 \) and \( Q \) is satisfiable in \( I_0 \).

(c) if \( r \) is \( p \lor q \), \( p \land q \) or \( p^* \), then \( P_r \) is \( P_p \cup P_q \), \( P_p \cap P_q \) or \( P_p^* \), respectively.

We then have to prove that

(*) \( P_r \) is satisfiable iff \( m_A(r) \neq \emptyset \), for some structure \( A \) of \( L \).

Before proceeding to prove (*), we observe that a regular program scheme \( r \)
can be viewed as a regular expression over the alphabet of tests and assignments. Likewise, \( P_r \) can be viewed as a regular expression over the alphabet of wffs of the form given by (a) and (b) above. Thus, \( r \) and \( P_r \) can be viewed as denoting sets of finite words in the appriate alphabet.

(\textcircled{\textasteriskcentered}) Suppose that there is a structure \( A \) of \( L \) such that \( m_A(r) \neq \emptyset \). Let \( U \) be
the universe of \( A \) and \( L \). Then, since \( m_A(r) \neq \emptyset \), there is a word \( s_0 \ldots s_n \) in
the set denoted by \( r \) such that \( s_i \) is either an assignment or a test. Moreover, there is a sequence \( \bar{v}=(v_0, \ldots, v_{n+1}) \) in \( U \) such that \( (v_i, v_{i+1}) \in m_A(s_i) \).

0\leq i \leq n. We also have, by construction of \( P_r \), that \( P_{s_0}, \ldots, P_{s_n} \) is in the set
denoted by \( P_r \). Now, let \( I=(A_{v_0}, \ldots, A_{v_{n+1}}) \). Then, by construction of \( P_r \)
and the basic property of \( \bar{v} \), we have that \( \models_I P_{s_i}[u] \), where \( I_i \) is the sub-
sequence of \( I \) starting on \( I_i \) and \( u \) is any fixed assignment of values to the
variables of \( L' \) (\( u \) is actually irrelevant since \( P_{s_i} \) is closed). Hence, \( P_r \)
is satisfiable.

(\Rightarrow) Suppose that \( P_r \) is satisfiable.

Then, there is a word \( P_0, \ldots, P_n \) in the set denoted by \( P_r \), a sequence
\( I=(I_0, I_1, \ldots) \) of structures of \( L' \) and an assignment \( u \) of values to the
variables of \( L' \) such that, for each \( i \in [0, n] \), \( \models_{I_i} P_i[u] \). By construction of
\( P_r \), for any \( i,j \in [0, n] \), the structures \( I_i \) and \( I_j \) are equal on the values
of \( f_1, \ldots, f_m \), \( p_1, \ldots, p_m \). Thus, \( I \) induces a structure \( A \) of \( L \). Moreover, \( I \)
also induces a sequence \( \bar{v}=(v_0, v_1, \ldots) \) of assignments of values to the
variables \( x_1, \ldots, x_k \) of \( L \), where \( v_i(x_j)=I_i(c_j) \), for \( i \geq 0 \) and \( 1 \leq j \leq k \).
But, by construction of $P_r$, the word $P_0 \ldots P_n$ induces a word $s_0 \ldots s_n$ in the set denoted by $r$ such that $(v_i, v_{i+1}) \in m_A(s_i)$, $0 \leq i \leq n$. Hence, we have that $(v_0, v_{n+1}) \in m_A(r)$ and, so, $m_A(r) \neq \emptyset$, as was to be shown. Thus, we may conclude that for any given regular program scheme $r$, we may construct a wff $P_r$ of a regular temporal language such that $P_r$ is satisfiable iff there is an interpretation $A$ of $r$ such that $m_A(r) \neq \emptyset$. Or, equivalently, $\neg P_r$ is valid iff $r$ diverges for any interpretation $A$. This concludes the proof. □

Theorem 3 has one important consequence that we state as a corollary.

COROLLARY 1: There is no consistent and complete axiom system for regular temporal languages.

Proof

If there were a consistent and complete axiom system, then the set of valid wffs would be r.e., but this contradicts Theorem 3. □

This concludes the discussion about the decision problem of regular temporal languages.

5. Conclusions

We defined a family of temporal languages flexible enough to express complex transition constraints in a natural way. The expressive power of the languages is largely due to the avoidance of a fixed set of modalities in favour of a mechanism to define new modalities. Since the mechanism is based on grammars, Chomsky's hierarchy induces directly a classification of the languages into distinct subfamilies.

We used this classification to obtain results about the decision problem of these languages. In particular, we have shown that the validity problem for right-linear (or regular) temporal languages is decidable, if we consider just the propositional case, but it is not partially decidable, is we consider the general case. We conjecture that there is an intermediate case, viz., that the validity problem for right-linear restricted temporal
languages (i.e., those that do not permit quantification across modalities - see Section 2) is undecidable, but partially decidable.

We have also shown that the validity problem is undecidable if we go beyond right-linear extensions, even in the propositional case.

We observe that the classification of temporal languages according to Chomsky's hierarchy could also be used to map results in formal language theory into results about the expressiveness of each subfamily of languages. For example, we could show that right-linear temporal languages are less expressive than context-free temporal languages by devising a transition constraint that mirrors a context-free language which cannot be expressed by a right-linear grammar.

Finally, we observe that the family of languages defined here would also be useful in contexts other than database specification, such as protocol specification, program verification and office automation.
APPENDIX A

The Analytic Tableau Method for Propositional Calculus

The analytic tableau method for propositional calculus [Sm] can be viewed as a systematic search of a model for a wff $P$ based on the idea of case analysis. The search procedure generates a tree, or analytic tableau, whose nodes are sets of wffs and where the sons of a node correspond to branching cases. A search terminates when each branch either contains a contradiction or cannot be extended further without repetition. Reasoning by cases is captured by using rules of the following type

\[
R_i 
\begin{array}{ll}
\text{P}_i \\
Q_{ij}1_1, \ldots, Q_{in}
\end{array}
\]

where $P_i$ is a wff and $Q_{ij}(1 \leq j \leq n_i)$ are finite sets of wffs. We call $P_i$ the antecedent of $R_i$ and $Q_{ij}(1 \leq j \leq n_i)$ a set of consequents of $R_i$.

These observations are formalized as follows.

DEFINITION A1:

(a) The set of analytic tableaux for a finite set $P$ of wffs of Propositional Calculus consists of trees whose nodes are sets of wffs. It is defined inductively as follows:

(i) the tree whose only node is $P$ is an analytic tableau for $P$;

(ii) suppose that $T$ is an analytic tableau for $P$ and let $\lambda$ be a leaf of $T$. Then, any tree obtained by extending $T$ by the following operation is also an analytic tableau for $P$: if there is a rule in Figure A.1 with antecedent $P_i$ and sets of consequents $Q_{ij1}, \ldots, Q_{ijn}$ such that $P_i$ occurs in the branch $\beta$ ending in $\lambda$, then $n_i$ distinct sons $\lambda_1, \ldots, \lambda_n$ may simultaneously be adjoined to $\lambda$, where $\lambda_j \in Q_{ij}(1 \leq j \leq n_i)$;

If some $\lambda_j$ is not a subset of the union of the nodes in $\beta$, then we say that $T$ was extended without repetition.
(b) A set $H$ of wffs is a Hintikka set iff no wff and its negation occur in $H$ and, for any rule $R_i$ in Figure A.1 with antecedent $P_i$, and sets of consequents $Q_{i_1},...,Q_{i_{n_i}}$, if $P_i \in H$, then $Q_{i_j} \in H$, for some $j \in [1,n_i]$.

(c) A branch of a tableau is closed iff it contains a wff and its negation; otherwise it is open.

(d) A branch of a tableau is complete iff the union of all its nodes is a Hintikka set.

(e) A tableau is closed iff every branch is closed.

(f) A tableau is complete iff every branch is closed or some branch is complete.

Rules for Propositional Calculus

A-rules: \[
\frac{A}{A_1, A_2}
\]

B-rules: \[
\frac{B}{B_1 | B_2}
\]

where $A,A_1,A_2$ are given in Table 1 and $B,B_1,B_2$ in Table 2.

<table>
<thead>
<tr>
<th>$A$</th>
<th>$A_1$</th>
<th>$A_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P \land Q$</td>
<td>$P$</td>
<td>$Q$</td>
</tr>
<tr>
<td>$\neg(P \lor Q)$</td>
<td>$\neg P$</td>
<td>$\neg Q$</td>
</tr>
<tr>
<td>$\neg(P \Rightarrow Q)$</td>
<td>$P$</td>
<td>$\neg Q$</td>
</tr>
<tr>
<td>$\neg \neg P$</td>
<td>$P$</td>
<td>$P$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$B$</th>
<th>$B_1$</th>
<th>$B_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\neg(P \land Q)$</td>
<td>$\neg P$</td>
<td>$\neg Q$</td>
</tr>
<tr>
<td>$P \lor Q$</td>
<td>$P$</td>
<td>$Q$</td>
</tr>
<tr>
<td>$P \Rightarrow Q$</td>
<td>$\neg P$</td>
<td>$Q$</td>
</tr>
</tbody>
</table>

Figure A.1
We can now define a procedure to decide if a set $P$ of wffs of Propositional Calculus is satisfiable. The procedure simply constructs a tableau for $P$ taking care to extend the tableau without repetition. The procedure stops extending the tableau when it becomes complete, which always happens after a finite number of steps since, in each step, the tableau is extended without repetition. Finally, the procedure answers 'NO' if the tableau is closed and 'YES', otherwise. The proof that this procedure is correct can be found in [Sm].
References


