A Logic for the Description of Non-deterministic Programs and Their Properties

S. Graf and J. Sifakis

IMAG–Lab de Genie Informatique, BP 68, 38402 St. Martin d'Hères Cedex, France

We present a logic, called Synchronization Tree Logic (STL), for the specification and proof of programs described in a simple term language obtained from a constant Nil by using a set A of unary operators, a binary operator + and recursion. The elements of A represent names of actions, + represents non-deterministic choice, and Nil is the program performing no action. The language of formulas of the logic proposed, contains the term language used for the description of programs, i.e., programs are formulas of the logic. This provides a uniform frame to deal with programs and their properties as the verification of an assertion \( t \models f \) (\( t \) is a program, \( f \) is a formula) is reduced to the proof of the validity of the formula \( t \models f \). We propose a sound and under some conditions complete deductive system for synchronization tree logics and discuss their relation with modal logics used for the specification of programs.

1. Introduction

This work has been motivated by the following general problem: Find logics for the specification and proof of non-deterministic programs. We suppose that programs belong to the set of terms \( \mathcal{I} \) of an algebra with a congruence relation \( \approx \). The operators of the algebra correspond to program constructors, and the relation \( \approx \) defines a concept of equivalence which is supposed to be satisfactory for the comparison of programs.

One requirement for a logic with set of formulas \( F \), to be an appropriate tool for the specification and proof of such programs is,

\[
(1) \quad \forall t_1, t_2 \in \mathcal{I} \, (t_1 \approx t_2) \iff \forall f \in F \, (t_1 \models f \iff t_2 \models f),
\]
i.e., the congruence \( \approx \) and the equivalence relations induced by the logic on programs, agree.

Consequences of this requirement are,

— formulas represent unions of congruence classes (congruent terms satisfy the same formulas),
— the logic is sufficiently powerful to distinguish non-congruent terms (for each congruence class there exists a formula of the logic representing it).
This requirement, known as adequacy, is satisfied by the logic HML proposed by Hennessey and Milner (1985) for observational equivalence on finite CCS terms. Furthermore, the use of such a logic as a tool for syntax directed proofs requires that to any program constructor \( \text{cons} \) of the logic, such that

\[
(2)(a) \ t_i \models f_i \text{ for } i = 1 \ldots n \text{ implies } \text{cons}(t_1 \ldots t_n) \models \text{cons}(f_1 \ldots f_n),
\]

and

\[
(2)(b) \ \text{cons}(t_1 \ldots t_n) \models \text{cons}(f_1 \ldots f_n) \text{ is the strongest assertion which can be deduced from } t_i \models f_i \text{ for } i = 1 \ldots n, \text{ where } t_i \in \mathcal{T} \text{ and } f_i \in \mathcal{F}.
\]

In this paper we present logics satisfying requirements (1) and (2) for a simple non-deterministic term language with a given congruence relation. Its terms are obtained from a constant \( \text{Nil} \) by using a set \( A \) of unary operators, a binary operator \( + \), and recursion. The elements of \( A \) represent actions, \( + \) represents non-deterministic choice, and \( \text{Nil} \) the program performing no action. Such a term language is at the base of various calculi for communicating systems (Milner, 1980).

The language of formulas is an extension of this term language which generalizes the program language into a language for describing program properties. It contains the term language used for the description of programs, i.e., programs are formulas of the logic. The verification of an assertion \( t \models f \) is reduced to the proof of the validity of the formula \( t \models f \). This provides a uniform frame to deal with programs and their properties.

The language of formulas is obtained from the constants \( \text{Nil}, T \) by using the Boolean Connectives, the set \( 2^A \) of unary operators (\( A \) is the set of actions of programs), the binary operator \( + \) and fixpoint operators. The operator \( + \) of the logic is an extension of the operator \( + \) on programs such that \( t_1 \models f_1 \) and \( t_2 \models f_2 \) implies \( t_1 + t_2 \models f_1 + f_2 \), where \( t_1, t_2 \) are programs and \( f_1, f_2 \) are formulas.

The paper is organized as follows:

In part 2, we present the term language used to describe programs and its operational semantics in terms of \( A \)-labelled trees (synchronization trees).

In part 3, we present the logic used to describe programs and their specifications (properties). We call this logic \textit{Synchronization Tree Logic} as its non-logical operators correspond to operations on classes of trees. We first study a synchronization tree logic for the sublanguage of non-recursive programs and give a sound and relatively complete deductive system. Then, we introduce the synchronization tree logic for recursive programs, show its adequacy and propose a sound deductive system. Finally, we show how to define modalities used in branching time logics and discuss the relation of synchronization tree logics with the \( \mu \)-calculus (Kozen, 1982).
2. PRELIMINARY RESULTS

Consider the term language $\mathcal{Z}$ built from the constants $\text{Nil}$, $T$, and a set of variables $X$ by using a set $A$ of unary operators, a binary operator $+$, and recursion.

- $\text{Nil}, T \in \mathcal{Z}, X \subseteq \mathcal{Z}$,
- at, $t + t'$, $\text{rec } x \cdot t \in \mathcal{Z}$ if $x \in X$, $a \in A$, $t, t' \in \mathcal{Z}$.

We denote by $\Sigma$ the signature $\Sigma = \{\text{Nil}\} \cup A \cup \{+\}$ and represent by,

- $\mathcal{Z}[\Sigma, X]$ the sublanguage of the well-guarded and closed terms obtained from $\text{Nil}$ by using elements of $A$, $+$, and recursion, where $\text{rec } x \cdot t$ is well-guarded means that any occurrence of the variable $x$ is under the scope of an operator of $A$.
- $\mathcal{Z}[\Sigma, X]$ the subset of $\mathcal{Z}[\Sigma, X]$ without recursive terms.
- $\mathcal{Z}[\Sigma \cup \{T\}]$ the subset of the non-recursive and closed terms of $\mathcal{Z}$.

We consider that elements of $\mathcal{Z}[\Sigma, X]$ represent programs; $\text{at}$ represents a set of actions, $+$ represents non-deterministic choice, and $\text{Nil}$ the program performing no action. The meaning of $T$ will be given later.

With a term $t$ of $\mathcal{Z}[\Sigma, X] \cup \mathcal{Z}[\Sigma \cup \{T\}]$ can in an obvious manner be associated an $A$-labelled tree $[t]$ (transition system) whose nodes correspond to subterms and labelled edges are defined as the least relations $\rightarrow a$ for $a \in A$, satisfying,

- at $\rightarrow^a t$ for $a \in A$, $t \in \mathcal{Z}[\Sigma, X] \cup \mathcal{Z}[\Sigma \cup \{T\}]$,
- $t_1 \rightarrow^a t$ implies $t_1 + t_2 \rightarrow^a t$

and

$t_2 + t_1 \rightarrow^a t$ for $t_1, t_2, t \in \mathcal{Z}[\Sigma, X] \cup \mathcal{Z}[\Sigma \cup \{T\}]$,

- $t[\text{rec } x \cdot t/x] \rightarrow^a t'$ implies $\text{rec } x \cdot t \rightarrow^a t'$ for $\text{rec } x \cdot t, t' \in \mathcal{Z}[\Sigma, X] \cup \mathcal{Z}[\Sigma \cup \{T\}]$.

To obtain the tree $[t]$ from $t$, take $t$ as root of $[t]$ and add edges and nodes following the definition of the relations $\rightarrow a$.

**Example.** The trees $[t_1], [t_2]$ representing the terms $t_1 = a(bT + c\text{Nil})$, $t_2 = \text{rec } x \cdot (ax + b\text{Nil})$ are,

\[
[t_1] = \begin{array}{c}
\text{Nil} \\
T \\
\text{Nil}
\end{array}
\]

\[
[t_2] = \begin{array}{c}
\text{Nil} \\
\text{Nil}
\end{array}
\]

**Remark.** The trees representing terms of $\mathcal{Z}[\Sigma, X]$ are finite branching as only well-guarded terms are admitted.
We define the equivalence relation \( \approx \) on \( \mathcal{X}[\Sigma, X] \) by taking
\[
\approx = \bigcap_{k \in \mathbb{N}} \approx_k,
\]
where,
1. \( t_1 \approx_0 t_2 \ \forall t_1, t_2 \in \mathcal{X}[\Sigma, X] \),
2. \( t_1 \approx_{k+1} t_2 \iff \forall t'_1 (t_1 \rightarrow^a t'_1 \implies \exists t'_2 (t_2 \rightarrow^a t'_2 \text{ and } t'_1 \approx_k t'_2)) \)
and \( \forall t'_2 (t_2 \rightarrow^a t'_2 \implies \exists t'_1 (t_1 \rightarrow^a t'_1 \text{ and } t'_1 \approx_k t'_2)) \).  

It is shown in (Hennessy and Milner, 1985; Brookes and Rounds, 1983) that \( \approx \) is a congruence and its restriction to \( \mathcal{X}[\Sigma] \) is the least congruence satisfying the following axioms:

\begin{align*}
T1 & \quad t_1 + (t_2 + t_3) = (t_1 + t_2) + t_3, \\
T2 & \quad t_1 + t_2 = t_2 + t_1, \\
T3 & \quad t + t = t, \\
T4 & \quad t + \text{Nil} = t.
\end{align*}

We propose a logic satisfying the adequacy criterion for programs of \( \mathcal{X}[\Sigma, X]/\approx \).

3. Synchronization Tree Logics

3.1. The Synchronization Tree Logic STL(A)

3.1.1. Syntax and Semantics

Let \( A \) be a set of actions as defined in Section 2. Consider the language of formulas \( F(A) \) defined by
\begin{align*}
- \text{Nil}, T & \in F(A) \\
- bf & \in F(A) \text{ for } b \subseteq A \text{ and } f \in F(A), \\
- f + f', f \lor f', \neg f & \in F(A) \text{ for } f, f' \in F(A).
\end{align*}

Notice that \( \mathcal{X}[\Sigma \cup \{ T \}] \subseteq F(A) \), by interpreting \( a \in \mathcal{X}[\Sigma \cup \{ T \}] \) as \( \{a\} t \in F(A) \) for \( a \in A \). We give the semantics of \( F(A) \) by defining a function \( \| \| \) associating with a formula \( f \) a set \( \| f \| \) of terms of \( \mathcal{X}[\Sigma, X] \) equal to a union of congruence classes of \( \approx \) on \( \mathcal{X}[\Sigma, X] \).

**Definition of \( \| \| \).** For \( b \subseteq A, f, f_1, f_2 \in F(A) \),
\begin{align*}
- \| T \| & = \mathcal{X}[\Sigma, X], \\
- \| \text{Nil} \| = \{ t \in \mathcal{X}[\Sigma, X] | t \approx \text{Nil} \}, \\
- \| bf \| & = \{ t \in \mathcal{X}[\Sigma, X] | \exists I \subseteq \mathbb{N} \text{ finite, } I \neq \emptyset, \forall i \in I \ (\exists a_i \in b \text{ and } \exists t_i \in \| f \|), t \approx \sum a_i t_i \}, \\
- \| f_1 + f_2 \| & = \{ t \in \mathcal{X}[\Sigma, X] | \exists t_1 \in \| f_1 \| \exists t_2 \in \| f_2 \|, t \approx t_1 + t_2 \}, \\
- \| f_1 \lor f_2 \| & = \| f_1 \| \cup \| f_2 \|, \\
- \| \neg f \| & = \mathcal{X}[\Sigma, X] - \| f \|. 
\end{align*}
Abbreviations and notations. 1. Having already defined \( \lor, \neg \) and \( T \), we can derive the definitions corresponding to the other Boolean connectives \( \land, \Rightarrow, \equiv \) and the constant \( \bot \) in the standard way. Furthermore, for \( t \in \mathcal{I}[\Sigma, X], f \in F(A) \), we define \( t \models f \) and \( \models f \) to mean \( t \in \{f\} \) and \( \{f\} = \mathcal{I}[\Sigma, X] \), respectively.

2. We use the convention that the following operators have progressively increasing priority: \( \neg, =, \lor, \land, + \). All unary operators have higher priority than binary operators and are right associative. For example \( \neg bf = \neg(bf) \) and \( b \neg f = b(\neg f) \).

3. For \( a \in A \), we write \( af \) instead of \( \{a\}f \).

4. For \( b \subseteq A \), we write \( \neg b \) to denote its complement \( A - b \).

PROPERTY. Note that for any \( f \in F(A) \) \( \{f\} \) is a union of classes of \( \equiv \), i.e., \( \{f\} \) is a set of terms closed with respect to \( \equiv \).

**Proposition 1.** \( \{t\} = \{t' \in \mathcal{I}[\Sigma, X] | t' \equiv t\} \) for \( t \in \mathcal{I}[\Sigma] \).

**Proof.** The proof is done by structural induction on \( \mathcal{I}[\Sigma] \).

— For \( t = \text{Nil} \) this is true by definition of \( \| \| \).

— Suppose that \( \|t\| = \{t'| t' \equiv t\} \). Then for \( a \in A \),

\[
\|at\| = \{t'| t' \equiv \Sigma_{i \in I} at_i, \text{where } t_i \in \{t\}\}\]

\[= \{t'| t' \equiv \Sigma_{i \in I} at_i, \text{where } t_i \equiv t\} \text{ (by induction hypothesis)}\]

\[= \{t'| t' \equiv at\} \]

— Suppose that \( \|t_i\| = \{t_i'| t_i' \equiv t_i\} \) for \( i = 1, 2 \). Then

\[
\|t_1 + t_2\| = \{t'| t' \equiv t_1' + t_2', \text{where } t_i' \in \{t_i\} \text{ for } i = 1, 2\}\]

\[= \{t'| t' \equiv t_1' + t_2', \text{where } t_i' \equiv t_i\} \text{ (by induction hypothesis)}\]

\[= \{t'| t' \equiv t_1 + t_2\}. \]

**Corollary.** \( \forall t_1, t_2 \in \mathcal{I}[\Sigma] t_1 \equiv t_2 \iff \models t_1 \equiv t_2 \).

This proposition means that for \( t \in \mathcal{I}[\Sigma] \), \( \|t\| \) is the congruence class of \( t \). It will be shown in Proposition 12 that any formula of \( F(A) \) can be transformed into a (possibly infinite) disjunction of terms of \( \mathcal{I}[\Sigma \cup \{T\}] \). The reader is invited to notice that for \( t \in \mathcal{I}[\Sigma \cup \{T\}] \), \( \|t\| \) contains all the terms obtained by substituting arbitrary terms of \( \mathcal{I}[\Sigma, X] \) for occurrences of \( T \). The following proposition, proved in Graf and and Sifakis (1985) gives an exact syntactical characterization of \( \|t\| \) in terms of the relations \( \equiv_{Ax} \) and \( \prec \) defined by,

— \( \equiv_{Ax} \) is the least congruence on \( \mathcal{I}[\Sigma \cup \{T\}] \) induced by the axioms T1, T2, T3, and T4.

— \( \prec \) is the least order relation on \( \mathcal{I}[\Sigma \cup \{T\}] \cup \mathcal{I}[\Sigma, X] \) such that,
\( - t < T \) for any \( t \in \mathcal{I}[\Sigma \cup \{ T \}] \cup \mathcal{I}[\Sigma, X] \),

\( - t < t' \) implies \( at < at' \) for \( a \in A \), \( t, t' \in \mathcal{I}[\Sigma \cup \{ T \}] \cup \mathcal{I}[\Sigma, X] \),

\( - t_1 < t'_1 \) and \( t_2 < t'_2 \) implies \( t_1 + t_2 < t'_1 + t'_2 \) for \( t_i, t'_i \in \mathcal{I}[\Sigma \cup \{ T \}] \cup \mathcal{I}[\Sigma, X] \) for \( i = 1, 2 \).

**Proposition 2.** (Graf and Sifakis, 1985)

\[ |t| = \{ t' \in \mathcal{I}[\Sigma, X] | t' \approx < T \} \text{ for } t \in \mathcal{I}[\Sigma \cup \{ T \}] \]

### 3.1.2. A Deductive System for STL(A)

The table below gives an equational deductive system for STL(A). The proof of its soundness is given by the Propositions 3 to 8.

**Table 1**

A Deductive System for STL(A)

<table>
<thead>
<tr>
<th>Axioms</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>B</strong></td>
</tr>
<tr>
<td><strong>T1</strong></td>
</tr>
<tr>
<td><strong>T2</strong></td>
</tr>
<tr>
<td><strong>T3</strong></td>
</tr>
<tr>
<td><strong>T4</strong></td>
</tr>
<tr>
<td><strong>D1</strong></td>
</tr>
<tr>
<td><strong>D2</strong></td>
</tr>
<tr>
<td><strong>D3</strong></td>
</tr>
<tr>
<td><strong>D4</strong></td>
</tr>
<tr>
<td><strong>D5</strong></td>
</tr>
<tr>
<td><strong>N1</strong></td>
</tr>
<tr>
<td><strong>N2</strong></td>
</tr>
<tr>
<td><strong>ST1</strong></td>
</tr>
<tr>
<td><strong>ST2</strong></td>
</tr>
<tr>
<td><strong>ST3</strong></td>
</tr>
<tr>
<td><strong>DE</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>R1</strong></td>
</tr>
<tr>
<td><strong>R2</strong></td>
</tr>
<tr>
<td><strong>R3</strong></td>
</tr>
</tbody>
</table>

\( \equiv f' \lor g' \lor g \)
PROPOSITION 3 (Tree axioms). $T_1$, $T_2$, $T_3$ and $T_4$ are valid.

Proof. $T_1$, $T_2$ and $T_4$ are obvious from the definition of $| |$. $T_3$ follows from Lemma 1. 

LEMMA 1. If $f \in F(A)$ is a formula of the form $f = \sum_{i \in T} f_i$ where each $f_i$ is of the form $b_i f_i'$, $T$ or Nil then,

$$t, t' \in | f | \text{ implies } t + t' \in | f |.$$  \hspace{1cm} (1)

Proof. Obviously, (1) holds for any formula of the form $b f'$, $T$, or Nil. Now, we prove the lemma for $f = f_1 + f_2$. The general case follows as an obvious generalization. If $f_1, f_2$ are formulas for which (1) holds and $t, t' \in | f_1 + f_2 |$ then there exist $t_i, t'_i \in | f_i |$ for $i = 1, 2$ such that $t \approx t_1 + t_2$ and $t' \approx t'_1 + t'_2$. From the fact that (1) holds for $f_1$ and $f_2$ we get $t_i + t'_i \in | f_i |$ for $i = 1, 2$. Thus, $t + t' \approx t_1 + t_2 + t'_1 + t'_2 \approx (t_1 + t'_1) + (t_2 + t'_2) \in | f_1 + f_2 |$. 

Remarks. Notice that (1) does not hold for any formula $f \in F(A)$. For example, for $f = a \text{Nil} \lor b \text{Nil}$, we have $a \text{Nil} \in | f |$ and $b \text{Nil} \in | f |$ but $a \text{Nil} + b \text{Nil} \notin | f |$.

---

PROPOSITION 4 (Distributivity axioms). $D_1$, $D_2$, $D_3$, $D_4$, $D_5$ are valid formulas.

Proof. $D_1$. $t \in | f_1 + (f_2 \lor f_3) |$ iff $\exists t_1, t_i$ such that $t \approx t_1 + t_i$, $t_i \in | f_i |$ and $t_i \in | f_1 \lor f_2 \lor f_3 |$ iff $\exists t_1, t_i$ such that $t \approx t_1 + t_i$, $t_i \in | f_i |$ and $t_i \in | f_1 \lor f_2 \lor f_3 |$.

$D_2$. $t \in (b_1 \lor b_2) f$ iff $t \approx \sum_{i \in I} a_i t_i$, where $a_i \in b_1 \lor b_2$, $t_i \in | f_i | \forall i \in I$ iff either $\forall i \in I a_i \in b_1$ in which case $t \in | b_1 f |$ or $\forall i \in I a_i \in b_2$ in which case $t \in | b_2 f |$ or $\exists I_1, I_2, I = I_1 \cup I_2, \forall i \in I$, $a_i \in b_1$, $\forall i \in I_1 a_i \in b_2$ and $t \approx \sum_{i \in I_1} a_i t_i + \sum_{i \in I_2} a_i t_i$. We have $\sum_{i \in I_1} a_i t_i \in | b_1 f |$ and $\sum_{i \in I_2} a_i t_i \in | b_2 f |$. Thus, $t \in | b f |$. 

$D_3$. A similar proof can be done.

$D_4$. From the definition of $| |$ it is obvious that, $|(b_1 \lor b_2)(f_1 \land f_2)| \subseteq |b_1 f_1 \land b_2 f_2 |$. Suppose that $t \in |b_1 f_1 \land b_2 f_2 |$. This implies $t \in |b_1 f_1 |$ and $t \in |b_2 f_2 |$. Thus, $t \approx \sum a_i t_i$, where $a_i \in b_1$, $t_i \in | f_i |$ and $a_i \in b_2$, $t_i \in | f_2 |$. From this deduce $a_i \in b_1 \lor b_2$ and $t_i \in | f_i \lor f_2 |$. Thus, $t \in |(b_1 \lor b_2)(f_1 \lor f_2) |$.

$D_5$. Suppose that $t \in | \sum f_i \land \sum g_k |$ where each $f_i$ and $g_k$ is of the form $b f$ or $T$. This implies by definition of $| |$, $t \approx \sum_{i \in I_1} a_i t_i$ and
$t \simeq \sum_{k \in K} \sum_{m \in \mathcal{K}_k} p_{km}$, where $\forall i \forall n \ t_{in} \in |f_i|$, $\forall k \forall m \ p_{km} \in |g_k|$, and each $t_{in}$ and the $p_{km}$ is of the form at $a \in A$, $t \in \Sigma[\Sigma, X]$. Due to the fact that $t \simeq \sum_{i \in I} \sum_{n \in I} t_{in} \simeq \sum_{k \in K} \sum_{m \in \mathcal{K}_k} p_{km}$, we have, $\forall i, n \exists k, m t_{in} \simeq p_{km}$ and $\forall k, m \exists i, n t_{in} \simeq p_{km}$. Thus, $\forall i, n t_{in} \in |f_i \vee g_k|$ and $\forall k, m p_{km} \in |g_k \vee f_i|$. 

Conversely, suppose that $t \in \Sigma(f_i \vee g_k) + \Sigma(g_k \vee f_i)$. This implies, $\exists t' \in \Sigma(f_i \vee g_k), \exists p' \in \Sigma(g_k \vee f_i)$ such that $t \simeq t' + p'$. This implies, $\exists \{t_i\}_{i \in I}, \exists \{p_k\}_{k \in K}$ such that $t' \simeq \sum t_i$, $t_i \in |(f_i \vee g_k)| \forall i \in I$ and $p' \simeq \sum p_k$. $p_k \in |(g_k \vee f_i)|$. Thus, $\forall i \in I \exists k \ t_i \in |g_k|$. From $p' \in |\Sigma g_k|$ and Lemma 1, one can deduce that $\forall i \in I \ p' + t_i \in |\Sigma g_k|$. Thus, $p' + \sum t_i \simeq p' + t' \in |\Sigma g_k|$. Symmetrically, we have $p' + t' \in |\Sigma f_i|$. Hence, we get the result, $t \simeq t' + p' \in |\Sigma f_i \vee g_k|$. 

**Proposition 5 (Negation axioms).** N1, N2 are valid formulas.

**Proof.** N1. $t \not\in |bf + T|$ iff $\forall t', \forall a_i, t_i$ such that $t \simeq a_i + t' \ (a_i \not\in b$ or $t_i \not\in |f|)$ iff $t \not\in \text{Nil}$ or $t$ can be put into the form $\sum a_i t_i$ and $\forall i \in I \not\in \emptyset$ $(a_i \not\in b$ or $t_i \not\in |f|)$ iff $t \not\in \text{Nil}$ or $t$ can be put into the form $\sum a_i t_i$ and either $\forall i \in I \ a_i \not\in b$ or $\forall i \in I \ a_i \not\in b$ and $t_i \not\in |f|$ or $\exists t', \exists t''$ $I'' \not\in \emptyset \ I = I' \cup I''$ and $\forall i \in I' \ a_i \not\in b$ and $\forall i \in I'' \ (a_i \in b$ and $t_i \not\in |f|)$. This is equivalent to $t \in \text{Nil} \lor \neg bT \lor b \rightarrow f \lor \neg f + bT + bf]$. 

N2. $t \not\in |\sum b_i f_i|$ iff $t \not\in \text{Nil}$ or $t$ can be put into the form $\sum k a_k t_k$ and $(\exists i \in I \ \forall k \in K \ (a_k \not\in b_i$ or $t_k \not\in |f_i|))$ or $\exists k \in K \forall i \in I \ (a_k \not\in b_i$ or $t_k \not\in |f_i|))$. Notice that $t \in |\sum b_i f_i + T|$ iff $t$ can be put into the form $\sum k a_k t_k$, where $\forall k \exists a_k (a_k \in b_i$ and $t_k \in |f_i|)$. From this remark we obtain, $t \not\in |\sum b_i f_i|$ iff $t \not\in \text{Nil}$ or $t$ can be put into the form $\sum k a_k t_k$ and $t \not\in |\sum b_i f_i + T|$ or $\exists k \ a_k t_k \in \bigwedge_i (\neg b_i T \lor b_i \neg f_i) \iff t \in \text{Nil} \lor \neg (\sum b_i f_i + T) \lor \bigwedge_i (\neg b_i T \lor b_i \neg f_i) + T]$. 

**Proposition 6 (Strictness axioms).** ST1, ST2, ST3 are valid formulas.

**Proof.** Obvious from the definition of $|\cdot|$. 

**Proposition 7 (Decomposition axiom).** DE is a valid formula.

**Proof.** For any $t \in \Sigma[\Sigma, X]$, either $t \simeq \text{Nil}$ or $\exists a_i, t_i$ such that $t \simeq \sum a_i t_i \in |\text{AT}|$. Thus, either $t \in |\text{Nil}|$ or $t \in |\text{AT}|$. 

**Proposition 8.** R1, R2, R3 are valid rules.

**Proof.** R1. Evident.

R2. We have to prove that for $f_1, f_2 \in F(A)$ and $\forall b \subseteq A$, $|f_1| \subseteq |f_2|$ implies $|bf_1| \subseteq |bf_2|$. This follows easily from, $|bf_1| = \{t t \simeq \sum a_i t_i, a_i \in b, t_i \in |f_i|\} \subseteq \{t t \simeq \sum a_i t_i, a_i \in b, t_i \in |f_2|\} = |bf_2|$. 

R3. We have to prove that for $f_i, f'_i \in F(A)$ for $i = 1, 2$ $|f_i| \subseteq |f_2|$ and $|f'_i| \subseteq |f'_2|$ implies $|f_i + f'_i| \subseteq |f_2 + f'_2|$. This is obtained by, $|f_i + f'_i| =
The following propositions give interesting theorems of STL(A).

**Proposition 9.**

**Th1.** \( AT + T \equiv AT \)

**Th2.** \( \text{Nil} \land bf \equiv \bot \) for \( bf \in F(A) \)

**Th3.** \( \neg \text{Nil} \equiv AT. \)

**Proof.**

**Th1.** \( AT + T \equiv AT + (\text{Nil} \lor AT) \) by DE

\[ \equiv AT + \text{Nil} \lor AT + AT \] by D1

\[ \equiv AT \] by T3, T4, and idempotence of \( \lor \).

**Th2.** \( \text{Nil} \land bf \equiv \bot \) is equivalent to \( \neg \text{Nil} \lor \neg (bf) \equiv T. \)

\[ \neg \text{Nil} \lor \neg (bf) \equiv \neg \text{Nil} \lor \text{Nil} \lor \neg (bf + T) \lor \]

\[ (\neg T \lor b(\neg f)) + T \] (by N2)

\[ \equiv T. \]

**Th3.** is a consequence of Th1 and Th2.

**Proposition 10.**

**Th4.** \( f_1 + f_2 + T \equiv (f_1 + T) \land (f_2 + T), \) where each \( f_i \) is

of the form \( b_if_i' \), \( T \) or \( \text{Nil} \).

**Th5.** \( f_1 + f_2 + T \equiv f_1 + T, \) where each \( f_i \) is

of the form \( b_if_i' \), \( T \) or \( \text{Nil} \).

**Th6.** \( (b_1 \lor b_2)f + T \equiv (b_1f + T) \lor (b_2f + T). \)

**Th7.** \( b(f_1 \lor f_2) + T \equiv (bf_1 + T) \lor (bf_2 + T). \)

**Proof.**

**Th4.** \( (f_1 + T) \land (f_2 + T) \equiv f_1 + (f_2 \lor T) + T \land (f_2 \lor T) + f_2 +

\[ \land (f_1 \lor T) + T \land (f_1 \lor T) \] by D5

\[ \equiv f_1 + T + f_2 + T \equiv f_1 + f_2 + T. \]

**Th5.** A consequence of Th4.

**Th6.** \( (b_1 \lor b_2)f + T \equiv b_1f + T \lor b_2f + T \lor b_1f + b_2f + T \) by D1, D2

\[ \equiv b_1f + T \lor b_2f + T \] by Th5.

**Th7.** A similar proof can be done.

**Remarks.**

1. Notice that the elements of \( 2^A \) represent some kind of

"weakest precondition" operators as \( b \bot \equiv \bot \) and \( b(f_1 \land f_2) \equiv bf_1 \land bf_2 \) for \( b \in A. \) These operators do not distribute over \( \lor. \) However, due to D3

\( b(f_1 \lor f_2) \) has an interesting decomposition in terms of \( bf_1 \) and \( bf_2 \) where

the non-deterministic construct + plays an important role.

2. Notice that due to DE, the equivalence of formulas depends on

the set \( A. \) The following are theorems only if \( A = \{a\}. \)

\( \text{Nil} \lor aT \equiv T, \) which implies by R2

\( a(\text{Nil} \lor aT) \equiv aT, \) which by D3 is equivalent to

\( a\text{Nil} \lor aaT \lor a\text{Nil} + aaT \equiv aT. \) This implies by R3
\[(a\text{Nil} \lor aT \lor a\text{Nil} + aT) + (a\text{Nil} + a\text{Nil}) \equiv aT + (a\text{Nil} + a\text{Nil}),\]
equivalent to \[aT + a\text{Nil} + a\text{Nil} \equiv aT + a\text{Nil} + a\text{Nil}.

3.1.3. A Relative Completeness Result for \(\text{STL}(A)\)

**Proposition 11.** For any formula \(f \in F(A)\) there exists \(f' \in F(A)\) with possible occurrences of \(\bot\), but without occurrences of \(\land\) and \(\neg\) such that \(\vdash f \equiv f'\).

**Proof.** Negation can be eliminated by applying the following rules:

\[\neg(f_1 \lor f_2) \rightarrow \neg f_1 \land \neg f_2\]
\[\neg(f_1 \land f_2) \rightarrow \neg f_1 \lor \neg f_2\]
\[\neg(\sum b_i f_i) \rightarrow \neg(\sum b_i f_i + T) \lor [\land (-b_i T \lor b_i \neg f_i)] + T\]
\[\neg(\sum b_i f_i + T) \rightarrow \lor -b_i T \lor \lor b_i \neg f_i \lor \lor (-b_i T + b_i \neg f_i) \lor \text{Nil}\]
\[\lor \rightarrow T\]
\[\neg T \rightarrow \bot\]
\[\neg \text{Nil} \rightarrow AT.\]

Thus, we obtain from \(f\) a formula \(f'\) without occurrences of negations. Using \(D1\) and the Boolean axioms, \(f'\) can be transformed into an equivalent formula which is a disjunction of conjunctions of terms of the form \(\Sigma f_i\), where each \(f_i\) is of the form \(bf\), \(T\) or \(\text{Nil}\). By application of Th4, D5 and then D4, Th2 and properties of Boolean algebras, conjunctions can be pushed into a lower level (as shown by the following example). By repeating this transformation, conjunctions can be eliminated.

**Example.** For \(a, b, c, d \in A\) we have,
\[
\langle\{a, b\} T + c \text{Nil} \rangle \land \langle\{a, c\} \text{Nil} + \{a, d\} T \rangle \equiv
\langle\langle\{a, b\} T \land \{\{a, c\} \text{Nil} \lor \{a, d\} T\}\rangle \lor [\neg \text{Nil} \land \langle\{a, c\} \text{Nil} \lor \{a, d\} T\rangle] \lor \langle\{a, c\} \text{Nil} \lor \{a, d\} T \lor \text{Nil}\rangle \equiv
\langle\{a, c\} \text{Nil} \lor \{a, d\} T \lor \text{Nil}\rangle \equiv
\]
\[aT + c \text{Nil} + (a \text{Nil} \lor c \text{Nil}) \equiv aT + c \text{Nil}.
\]

Let \(A\) be a recursive set. Denote by \(F_{\text{rec}}(A)\) the sublanguage of \(F(A)\) obtained by restricting the action operators \(b\) to recursive subsets of \(A\) (notice that the set of recursive subsets of \(A\) from a Boolean lattice).

**Proposition 12.** Let \(f\) be a formula of \(F_{\text{rec}}(A)\). If \(\models f \equiv \bot\) then \(\vdash f \equiv \bot\); otherwise, there exists \(f' \in F_{\text{rec}}(A)\), such that \(\vdash f \equiv f'\) and \(f'\) contains \(\lor\) as the only logical connective and has no occurrences of \(\bot\).
Proof. By applying the result of Proposition 11, one can obtain a formula \( f_1 \), \( \neg f \equiv f_1 \) such that \( f_1 \) contains \( \lor \) as the only logical connective but \( f_1 \) has possible occurrences of \( \bot \). In \( f_1 \) substitute \( \emptyset \) for any operator \( b \) representing the empty set (this is possible as we restrict unary operators to recursive subsets of \( A \)). By applying the rules \( b \bot \rightarrow \bot, \bot + f \rightarrow \bot, \emptyset f \rightarrow \bot, \) \( \bot \lor f \rightarrow f \) one can obtain from \( f_1 \) an equivalent formula \( f_2 \) such that either \( f_2 = \bot \) or \( f_2 \) has no occurrences of \( \bot \). In the latter case, \( f_2 \) is a formula constructed from \( \emptyset \) and \( T \) by using operators \( b \subseteq A \) such that \( b \not\subseteq \emptyset, \lor \) and \( + \). Obviously, for such a formula we have \( \not\not f_2 \equiv \bot \).

We prove hereafter the completeness of the deductive system given for \( F_{\text{rec}}(A) \).

Theorem 1. The deductive system of Table I is sound and complete for \( F_{\text{rec}}(A) \).

Proof. Soundness has been proved by the Propositions 3 to 8. Completeness is a consequence of Proposition 12, where it has been shown that \( \forall f \in F_{\text{rec}}(A) \models f \equiv \bot \) implies \( \not\not f \equiv \bot \). (*) Completeness, i.e., \( [\forall g \in F_{\text{rec}}(A) \models g \) implies \( \not\not g ] \) is equivalent to \( [\forall g \in F_{\text{rec}}(A) \models \not g \equiv \bot \) implies \( \not\not \not g \equiv \bot ] \). By taking \( f = \not g \) in (*) we get the proof.

3.2. The Extension STL(A, X) of STL(A)

In this section we propose a synchronization tree logic for the specification and proof of programs of \( \mathcal{L}[Z, X] \). Let \( X \) be a set of variables. Consider the language of formulas, defined by,

- \( T, \emptyset \) and \( x \in X \) are formulas,
- \( b f \) is a formula if \( b \subseteq A \) and \( f \) is a formula,
- \( f + f', f \lor f' \) and \( \neg f \) are formulas if \( f \) and \( f' \) are formulas,
- \( \mu x \cdot f \) is a formula if \( x \in X \) and \( f \) is a formula positive in the variable \( x \), i.e., each free occurrence of \( x \) is under the scope of an even number of occurrences of \( \neg \).

Denote by \( F(A, X) \) the set of the closed formulas of this language (the notions of free occurrence of a variable and closed formula are the same as in predicate logic, i.e., \( \mu \) is treated as a quantifier).

To define the semantics of STL(A, X), we associate with any functional \( f \) with \( n \) free variables of \( F(A, X) \) a function \( [f] : C_1(A)^n \rightarrow C_1(A) \), where \( C_1(A) \) is defined by

\[ C_1(A) = \{ s \subseteq \mathcal{L}[Z, X] \mid t \in s \text{ and } t' \approx t \text{ implies } t' \in s \} \], i.e., elements of \( C_1(A) \) are unions of congruence classes.
For closed formulas \( f, |f| \) is a constant which defines the union of classes associated with \( f \).

**Definition of | |.** For \( b \subseteq A, f, f_1, f_2 \in F(A, X) \), \( y \in X, x = (x_1, ..., x_n) \in X^n \) and \( s = (s_1, ..., s_n) \in C1(A)^n \),

- \(| T|(s) = \Sigma [\Sigma, X]
- |Nil|(s) = \{ t \in [\Sigma, X] | t \approx Nil \}
- |x_i|(s) = s_i
- |bf|(s) = \{ t \in [\Sigma, X] | \exists a_i \in b \; \exists t_i \in |f|(s) \text{ for } i \in I, t \approx \sum_{i \in I} a_it_i \}
- |f_1 + f_2|(s) = \{ t \in [\Sigma, X] | \exists t_1 \in |f_1|(s) \; \exists t_2 \in |f_2|(s) \; t \approx t_1 + t_2 \}
- |f_1 \lor f_2|(s) = |f_1|(s) \lor |f_2|(s)
- |\neg f|(s) = [\Sigma, X] - |f|(s)
- |\mu f(y,x)|(s) = \bigcap \{ r \in C1(A) | |f|(r, s) \subseteq r \}.

**Abbreviations.** We use the same abbreviations as in 3.1.1. Furthermore, we put \( \forall x \cdot f(x) = \neg \mu x \cdot \neg f(\neg x) \).

**Lemma 4.** Let \( f(x_1, ..., x_n) \) be a functional of \( F(A, X) \) in which no negation occurs, with free variables \( x_1, ..., x_n \). For all sequences \( \{ f_{ik} \}_{k \in \mathbb{N}} \) of \( F(A, X) \) such that \( \forall k f_{ik} \Rightarrow f_{ik+1} \),

\[ f(\bigvee f_{ik}, ..., \bigvee f_{nk}) \equiv \bigvee f(f_{ik}, ..., f_{nk}), \text{ i.e., } f \text{ is } \lor \text{-continuous.} \]

**Proof.** It is sufficient to prove that the operators \( \lor, \land, +, b \subseteq A \) are \( \lor \)-continuous. It is well known that \( \land, \lor \) are \( \lor \)-continuous. Let \( \{ f_i \}_{i \in \mathbb{N}}, \{ g_i \}_{i \in \mathbb{N}} \) be two “increasing” sequences on \( F(A, X) \).

(a) \( b(\lor f_i) \equiv \lor bf_i \) by monotonicity of \( b \).

- \( \lor bf_i \Rightarrow b(\lor f_i) \) by monotonicity of \( b \).

- Let \( t \in \{ b(\lor f_i) \} \). This implies that \( t \) is of the form \( t \approx \sum_k a_k t_k \) such that \( \forall k \in K a_k \in b \) and \( t_k \in f_i \) (where \( K \) is a finite set of indices). Thus, \( \forall k \in K \exists j(k) \in \mathbb{N} a_k \in b \) and \( t_k \in |f_{j(k)}| \), which implies \( a_k t_k \in |bf_{j(k)}| \) .

(b) \( \lor (f_i + g_i) \equiv \lor (f_i + g_i) \) is a consequence of the monotonicity of \( + \).

**Lemma 5.** Let \( f(x_1, ..., x_n) \) be a functional of \( F(A, X) \) in which no
negation occurs, with free variables $x_1, \ldots, x_n$. For all sequences $\{f_{ik}\}_{k \in \mathbb{N}}$ of $F(A, X)$ such that $\forall k \ f_{ik+1} \Rightarrow f_{ik}$, $f(\bigwedge f_{1k}, \ldots, f_{nk}) \equiv \bigwedge f_{1k}, \ldots, f_{nk}$, i.e., $f$ is $\land$-continuous.

Proof. As in the preceding proof, it is sufficient to show that $b \ni A$ and $+$ are $\lor$-continuous. Let $\{f_i\}, \{g_i\}$ be “decreasing” sequences of $F(A, X)$.

(a) $b(\bigwedge f_i) \equiv \bigwedge b f_i$.

(b) $b(\bigwedge f_i) \Rightarrow \bigwedge b f_i$ by the monotonicity of $b$.

Let $t \in [\bigwedge b f_i]$. This implies $t \approx \sum_k a_k t_k$ and $t \in [b f_i]$ $\forall i \in \mathbb{N}$. Thus, $\forall k \in K \; \forall i \in \mathbb{N} \; a_k \in b$, and $t_k \in [f_i]$, which implies $\forall k \in K \; a_k t_k \in [b(\bigwedge f_i)]$. From the definition of $|\cdot|$, we deduce that $t \approx \sum_k a_k t_k \in [b(\bigwedge f_i)]$.

Proposition 13. Any functional of $F(A, X)$ which is positive in any variable $x$, is both $\lor$-continuous and $\land$-continuous.

Proof. A direct consequence of Lemmas 4 and 5.

From this proposition follows,

Proposition 14. For any well-guarded functional $f(x)$ of $F(A, X)$ $\mu x \cdot f(x) \equiv \bigvee_{k \in \mathbb{N}} f^k(\bot)$ and dually $\nu x \cdot f(x) \equiv \bigwedge_{k \in \mathbb{N}} f^k(T)$.

A consequence of this proposition is that $\mu x \cdot f(x)$ and $\nu x \cdot f(x)$ represent unions of congruence classes, as the finite approximants of fixpoints represent unions of classes, too.

The following proposition shows that the congruence class of recursive terms of $\mathbb{F}[\Sigma, X]$ can be represented by formulas of $F(A, X)$.

Proposition 15. For $t'$, $\text{rec } x \cdot t \in \mathbb{F}[\Sigma, X]$, $t' \approx \text{rec } x \cdot t$ iff $t' \in [\nu x \cdot t]$.

Proof. To prove the proposition, we prove for any $\text{rec } x \cdot t \in \mathbb{F}[\Sigma, X]$

(a) $\text{rec } x \cdot t \in [\nu x \cdot t]$, i.e., $\forall k \in \mathbb{N} \; \text{rec } x \cdot t \in [t_k]$, where $t_0 = t[T/x]$, and $t_{k+1} = t_k[t_k/x]$

(b) $\forall t' \in \mathbb{F}[\Sigma, X] \; (t' \in [\nu x \cdot t] \text{ implies } t' \approx \text{rec } x \cdot t)$, i.e., $\forall k \in \mathbb{N} \; t' \in [t_k] \text{ implies } t' \approx \text{rec } x \cdot t$. 

From this proposition follows,
(a) We prove \( \forall k \in \mathbb{N} \) \( \text{rec} \cdot x \cdot t \in |t_k| \) by induction on \( k \in \mathbb{N} \).

--- for \( k = 0 \), this is obvious.

--- Suppose that for some \( k \in \mathbb{N} \) \( \text{rec} \cdot x \cdot t \in |t_k| \).

--- Let us prove that \( \text{rec} \cdot x \cdot t \in |t_{k+1}| \). From \( t_{k+1} = t[t_k/x] \), the definition of \( \| \| \) and induction hypothesis we obtain \( t[\text{rec} \cdot x \cdot t/x] \in |t[t_k/x]| \). From \( \text{rec} \cdot x \cdot t \approx t[\text{rec} \cdot x \cdot t/x] \) and the fact that for any \( f \in F(A, X) |f| \) is a union of classes of \( \approx \), we obtain the result.

(b) To prove (b) it is sufficient to prove that \( \forall k \in \mathbb{N} \) \( (t' \in |t_k| \) implies \( t' \approx_k \text{rec} \cdot x \cdot t \). For any \( f \in \mathfrak{T}[\Sigma \cup \{T\}] \) we define \( \text{DT}(f) \) by, \( \text{DT}(f) = \min\{k | \exists s = s_1, ..., s_nf \rightarrow T\} \) as in (Graf and Sifakis, 1985). There has been proved that, \( \forall f \in \mathfrak{T}[\Sigma \cup \{T\}] \) \( \text{DT}(f) > k \) and \( t', t'' \in |f| \) implies \( t' \approx_k t'' \).

As \( \text{rec} \cdot x \cdot t \in \mathfrak{T}[\Sigma, X] \) the functional \( t \) is guarded and for any \( k \) we have \( \text{DT}(t_k) > k \). Thus, we obtain the result by the fact that \( \text{rec} \cdot x \cdot t \in |t_k| \) for any \( k \).

From this proposition and obvious properties of \( \| \| \), one deduces the adequacy of \( \mathfrak{T}[\Sigma, X] \) as a tool for the specification and proof of programs in \( \mathfrak{T}[\Sigma, X] \).

**Proposition 16** (adequacy). \( \forall t, t' \in \mathfrak{T}[\Sigma, X] \) \( (t \approx t' \iff \forall f \in F(A, X)(t \models f \iff t' \models f)) \).

Finally, it is not difficult to check that all the axioms and rules of \( \mathcal{STL}(A) \) are valid for \( \mathcal{STL}(A, X) \) and we get the following proposition.

**Proposition 17.** The deductive system obtained by adding \( \text{AFP} \vdash f(\mu x \cdot f) \supseteq \mu x \cdot f \), \( \text{RFP}(\neg f(g) \supseteq g) \vdash (\neg (\mu x \cdot f) \supseteq g) \) to the axioms and rules of \( \mathcal{STL}(A) \) is sound for \( \mathcal{STL}(A, X) \).

3.3. Definition of Temporal Modalities in \( \mathcal{STL}(A, X) \)

In this section we show how the modalities of standard branching time logics (Ben-Ari, Manna and Pnueli, 1983 and Queille and Sifakis, 1983) can be expressed in \( \mathcal{STL}(A, X) \). The operators \( b 
subseteq A \) of synchronization tree logic correspond to modalities expressing inevitable reachability of their argument by executing one action belonging to \( b \). In \( \mathcal{STL}(A, X) \) (and also in \( \mathcal{STL}(A) \)) operators \( \langle b \rangle \) for \( b \subseteq A \), expressing possible reachability by one action in \( b \), can be defined by taking, \( \langle b \rangle f = bf + T \), where \( b \subseteq A, f \in \mathcal{STL}(A, X) \).

**Properties.**

1. \( t \in |\langle b \rangle f| \iff \exists t' \exists a \in b \ (t \rightarrow "t'" \text{ and } t' \in |f|) \),
2. \( \langle b \cup c \rangle f \equiv \langle b \rangle f \lor \langle c \rangle f \),
(3) \( \langle b \rangle (f \lor f') \equiv \langle b \rangle f \lor \langle b \rangle f' \),
(4) \( \langle b \rangle f \land \langle c \rangle f' \equiv \langle b \rangle f + \langle c \rangle f' \).

Proof. 1. By the definition of \(| . |\).
2. \( \langle b \lor c \rangle f = (b \lor c) f + T \equiv (bf + T) \lor (cf + T) = \langle b \rangle f \lor \langle c \rangle f \) (by Th6).
3. A similar proof can be done by using Th7.
4. \( \langle b \rangle f \land \langle c \rangle f' = (bf + T) \land (cf + T) \equiv bf + cf' + T = \langle b \rangle f + \langle c \rangle f' \) (by Th4).

Notice that \( \langle b \rangle \) is similar to the “next time” operator of branching time logics or the “diamond” operator of dynamic logics or HML. In fact, any formula \( f \) of HML (Hennessy and Milner, 1985) can be translated into a formula \( H(f) \) of STL in the following manner.

\[
\begin{align*}
- H(T) &= T \\
- H(\langle a \rangle f) &= \langle a \rangle H(f) \\
- H(f \lor f') &= H(f) \lor H(f') \\
- H(\neg f) &= \neg H(f).
\end{align*}
\]

On the contrary, the translation from STL(\( A \)) into HML can only be done in the case where \( A \) is finite. Furthermore, in STL(\( A, X \)) the temporal modalities of the logics proposed in (Ben-Ari, Manna, and Pnueli, 1983; Queille and Sifakis, 1983) can be defined for \( f \in F(A, X) \) by,

\[
\begin{align*}
- \text{POT}(f) &= \mu x \cdot (f \lor \langle A \rangle x), \\
- \text{INEV}(f) &= \mu x \cdot (f \lor Ax),
\end{align*}
\]

where in Ben-Ari, Manna, and Pnueli (1983) \( \text{POT}(f) \) and \( \text{INEV}(f) \) are respectively denoted by \( \exists F(f) \) and \( \forall F(f) \). Obviously, \( \text{POT}(f) \) and \( \text{INEV}(f) \) express respectively possible and inevitable reachability of \( f \) by executing sequences of actions. To obtain conditional versions of the operators POT and INEV as in Queille and Sifakis (1983) take for \( b \in A, f \in F(A, X) \),

\[
\begin{align*}
- \text{POT}(b, f) &= \mu x \cdot (f \lor \langle b \rangle x), \\
- \text{INEV}(b, f) &= \mu x \cdot (f \lor bx).
\end{align*}
\]

These (binary) operators express possible and inevitable reachability of a term satisfying \( f \) by executing only actions belonging to \( b \).

An important feature of STL(\( A, X \)) is that it contains the language for the description or programs of \( \Sigma[X, X] \) and it is at least as expressive as logics used for the expression of their properties. So, the verification of an assertion \( t \models f \) is reduced to the proof of \( \models t \Rightarrow f \). The following example illustrates this idea.
EXAMPLE. Proving that $t \models f$ where $t = \text{rec } x \cdot a(ax + b \text{ Nil})$ and $f = \text{INEV}(\{a\}, \langle b \rangle \text{ Nil})$ is equivalent to proving the theorem, $\neg \forall x \cdot a(ax + b \text{ Nil}) \supset \mu x \cdot (\langle b \rangle \text{ Nil } \lor ax)$.

We have,

1. $\neg \forall x \cdot a(ax + b \text{ Nil}) \supset a(ax + b \text{ Nil}) + b \text{ Nil})$ by the dual of AFP.
2. $\neg a(ax + b \text{ Nil}) + b \text{ Nil}) \supset a(T + b \text{ Nil})$ by R2, R3,
3. $\neg a(T + b \text{ Nil}) \equiv a\langle b \rangle \text{ Nil}$ by definition,

Let $g = \langle b \rangle \text{ Nil } \lor a\langle b \rangle \text{ Nil } \lor a\mu x \cdot (ax \lor \langle b \rangle \text{ Nil})$.

4. $\neg a\langle b \rangle \text{ Nil } \supset g$ by D3 and Boolean axioms,
5. $\neg g \supset \mu x(ax \lor \langle b \rangle \text{ Nil})$ by AFP, R2, R3.

By 1, 2, 3, 4, and 5 we obtain, $\neg \forall x \cdot a(ax + b \text{ Nil}) \supset \mu x \cdot (ax \lor \langle b \rangle \text{ Nil})$.

3.4. Other Results Concerning STL(A, X)

A synchronization tree logic is presented in (Graf, 1983 and Graf and Sifakis, 1984), for the specification and the proof of controllable processes of CCS, i.e., processes $t$ such that there exists $t'$ without occurrences of $\tau$, observationally congruent to $t$. The logic presented there is compared with a logic with next time and least fixpoint operator ($\mu$-calculus) (Kozen, 1982). Both logics admit a common class of models: $A$-labelled trees representing elements of $\Sigma[\Sigma, X]$. A function $h$ is defined in a compositional manner, associating with a formula $f \in F(A, X)$ a formula $h(f)$ of the $\mu$-calculus such that, $\forall t \in \Sigma[\Sigma, X] t \models f$ iff $t \models h(f)$. As a result of this work, it follows that synchronization tree logics allow much more concise description of properties and are more adequate for the definition of syntax directed proof methods.

4. Conclusion

We have proposed a logic for the description and specification of simple non-deterministic programs. The language of formulas of this logic is an extension of the term language used to describe programs. This provides a uniform frame to deal with both programs and properties: programs are formulas of the logic, and proving the validity of an assertion $t \models f$ is reduced to the proof of the validity of the formula $\models t \models f$.

The language considered for the description of programs is certainly simple but we believe that our approach can be applied to obtain adequate
logics for term languages used in calculi for communicating systems as in (Graf, 1983; Graf and Sifakis, 1984; Graf and Sifakis, 1985). Synchronization tree logics seem to be an interesting specification tool as they provide operators for the direct expression of usual operations on non-deterministic programs. Compared to standard modal logics, they allow more concise descriptions and easier manipulation of formulas. Finally, their underlying structure seems to be quite original, and it has many interesting properties concerning relations between logical connectives, the non-deterministic construct $\oplus$, and modalities.

ACKNOWLEDGMENT

We wish to thank the referees for the helpful comments and suggestions, concerning in particular the proof of the completeness result.

REFERENCES


