The Temporal Logic of Branching Time

Mordechai Ben-Ari
Tel Aviv University

Zohar Manna
Stanford University
Weizmann Institute of Sciences

Amir Pnueli
Tel Aviv University
Weizmann Institute of Sciences

Abstract

A temporal language and system are presented which are based on branching time structure. By the introduction of symmetrically dual sets of temporal operators, it is possible to discuss properties which hold either along one path or along all paths. Consequently it is possible to express in this system all the properties that were previously expressible in linear time or branching time systems. We present an exponential decision procedure for satisfiability in the language based on tableaux methods, and a complete deduction system. As associated temporal semantics is illustrated for both structured and graph representation of programs.

Introduction

From the first introduction of the Temporal Logic formalism as a tool for reasoning about programs, there arose a basic question which later almost developed into a controversy. The question involves the nature of the underlying structure of time on which the formalism is based. The dichotomy is between the linear time approach which considers time to be a linear sequence, and the branching time approach, which adopts a tree structured time, allowing some instants to have more than a single successor.

The difference in approaches has very little to do with the philosophical question of the structure of physical time which leads to the metaphysical problems of determinancy versus free will. Instead, it is pragmatically based on the choice of the type of programs and properties one wishes to formalize and study.

Linear time is the correct model to use in order to characterize the set of all execution sequences which a program generates and to study properties which uniformly hold for all the execution sequences of a program. Even a nondeterministic program generates for each set of possible choices a linear execution sequence in which each execution state has a unique successor. The class of properties related to the set of execution sequences are the universal properties such as universal total correctness and universal responsiveness. Both these

This research was supported in part by: NSF under grant MCS-80-06930, Office of Naval Research under grant N000-14-76-C-0687 and the Israeli Academy of Sciences and Humanities, the Basic Research Foundation.

Permission to make digital or hard copies of part or all of this work or personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.
© 1981 ACM 0-89791-029-X...$5.00

properties require that every execution sequence of the program will eventually achieve some goal such as termination with a correct result or correct response to some request. The interpretation of temporal formulas over execution sequences of a given program was found to be very useful for reasoning about both sequential deterministic programs and concurrent programs. In the case of concurrent programs, where the nondeterminism is caused by different scheduling scripts, we generally wish to prove that the program terminates or responds correctly regardless of how the individual processes are scheduled. This approach is pursued in [PN], [MP], [L].

The branching time approach, on the other hand, considers for a given program the set of all execution trees generated by the program. With a nondeterministic program P and a given input x we can associate the tree of all possible computations of P on x. Since the program is nondeterministic, some of the execution states will have more than one successor corresponding to a nondeterministic choice. Over execution trees we can study existential properties such as correct termination for at least one possible computation (for every input). More generally, we may study the property that there is always one possible computation which realizes some goal. This certainly does not imply that all computations will realize the same goal. Consequently, this approach is useful for nondeterministic programs which are executed by systematically exploring all possible choices by methods such as breadth first search, etc. This interpretation of nondeterminism is recommended for example in [F] as a design tool and is the one classically used in automata and complexity theory. The branching time approach is implied in the underlying structure of Dynamic Logic ([H], [A]), but was not previously studied in a temporal framework.

In the end, the choice between linear and branching models cannot be made on philosophical grounds but instead should be dictated by the type of programs, execution policies and properties which one wishes to study. For a fuller discussion of this issue see [L].

A natural step at this point would be to formalize and investigate a branching temporal logic in order to compare the two approaches discussed above under the controlled environment of the same logic. It turns out that a unified system which combines both approaches is no more complex than the two separate systems.

We define $\mathcal{UB}$: the unified system of branching time. The underlying model will be the branching tree of all possible computations of a program. We define, however, additional temporal operators that allow reference either to all possible execution sequences or only to a single sequence. The meta-theoretical results in $\mathcal{UB}$ include:

1) An exponential decision procedure for satisfiability.
2) A finite model property.
3) A simple axiomatization which is shown to be complete.

The decision procedure uses semantic tableaux. Tableaux systems provide a rapid way of deciding "natural" formulas. The completeness theorem shows how to "read-off" a proof from a tableau.

The expressive power of the system is illustrated by formalizing both universal and existential properties of nondeterministic programs. Finally we give a temporal semantics for nondeterministic programs, complementing the semantics given in [PN] for deterministic but concurrent programs. The semantics will be given for programs which are presented both in graph form and in structured form.

The UB Language-Syntax and Semantics

The base of $\mathcal{UB}$ is the propositional calculus on $\sim$ and $\vee$ with the other connectives defined as usual. We use six modal operators symmetrically represented by the digraphs: $\forall G, \forall F, \forall X, \exists G, \exists F$ and $\exists X$. The first symbol denotes quantification over paths. The second symbol denotes the temporal quantification along the selected paths, with G,F and X having their meanings as in [PN].

Let T be a tree and s a node in T. Let a be a proposition which can hold at some nodes in the tree. Then the intuitive meaning of the modal operators when applied to a proposition is as follows:

$\forall Ga$	holds at s (in T) iff a is true at all nodes of the subtree rooted at s (including s).
$\forall Fa$	holds at s iff on every path departing from s there is some node at which a is true.
$\forall Xa$	holds at s iff a is true at every immediate successor (descendant) of s.
$\exists Ga$	holds at s iff there exists a path departing from s such that a is true at all nodes on this path.
$\exists Fa$	holds at s iff a is true at some node in the subtree rooted at s, i.e. there exists a path departing from s such that a is true at some node on this path.
$\exists Xa$	holds at s iff a is true at one of the immediate successors of s.

We give a now a more formal definition of the semantics of $\mathcal{UB}$.

A model T for $\mathcal{UB}$ is a triple $T = (S,P,R)$ where S is a set of states and P is an assignment of proposition letters to states. For a proposition a and a state $s \in S$, $a \in P(s)$ iff a is true at the state s. R is a binary relation on states which defines the structure of T. When sRt holds, we say that t is an immediate successor (descendant) of s. To capture the concept of non-ending time, we require that R be total, i.e. $\forall s \exists t \cdot (sRt)$ - every state has a successor. R^* is the reflexive transitive closure of R. Thus sR^*t iff there is an R-path leading from s to t. An s-branch b is an infinite path $b = (s = s_0, s_1, \ldots)$ such that $s_i \in S$ and $s_i R s_{i+1}$. We define the notion of a general formula p being satisfied at a node s in T - written as $T,s \models p$ or $s \models p$ when T is implicitly understood.

1. For a proposition a, $s \models a$ iff $a \in P(s)$.
2. $s \models \sim p$ iff $s \not\models p$
3. $s \models p \vee q$ iff $s \models p$ or $s \models q$
4. $s \models \forall Gp$ iff $\forall b \forall t\ (t \in b \supset t \models p)$
5. $s \models \forall Fp$ iff $\forall b \exists t\ (t \in b \wedge t \models p)$
6. $s \models \forall Xp$ iff $\forall t (sRt \supset t \models p)$
7. $s \models \exists Gp$ iff $\exists b \forall t\ (t \in b \supset t \models p)$
8. $s \models \exists Fp$ iff $\exists b \exists t\ (t \in b \wedge t \models p)$
9. $s \models \exists Xp$ iff $\exists t\ (sRt \wedge t \models p)$

In the above formulas the quantification of b is over all the s-branches in T.

A formula p is <u>satisfiable</u> if for some model T and some state $s \in S$ - $T,s \models p$.

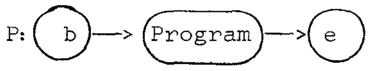
A formula p is true in T if for every state $s \in S$ - $T,s \models p$. We write $T \models p$.

A formula p is <u>valid</u> if it is true for every model T. We write $\models p$.

In subsequent sections we will present a procedure for deciding satisfiability and an axiomatic system for proving all valid formulas in $\mathcal{UB}$.

<u>Expressing Program Properties in UB</u>

As an example of the power of the language to express both universal and existential properties, consider a nondeterministic program:



We distinguish two special locations in this program - the beginning node b and the exit node e. We define special location propositions of the form $at\ell$ for each location ℓ in the program. The proposition $at\ell$ is true at execution state s if the execution currently resides at location ℓ in the program. Consequently the proposition atb holds at all initial execution states and the proposition ate is true at all terminated execution states. Let φ and ψ be respectively input and output predicates forming a correctness specification for the program P. Following [M1], [M2] we can distinguish four types of correctness of P relative to (φ,ψ):

a) P is partially $\exists$-correct with respect to (φ,ψ).

Either there is an infinite computation or there is a finite correct computation. This is expressible by the UB formula:

$$(\mathit{atb} \wedge \varphi) \supset \exists G(\mathit{ate} \supset \psi)$$

If we evaluate this formula over execution trees of the program P it forces every tree whose initial state satisfies φ to contain a path all of whose states satisfy $\mathit{ate} \supset \psi$. This implies that the computation corresponding to the path either diverges and never reaches e or terminates at e with ψ correct. Once we reach e the rest of the branch must infinitely repeat the same state since the program dictates no farther change. This explains the representation of finite computations in our infinite model.

b) P is totally $\exists$-correct with respect to (φ,ψ). There is a finite correct computation, but other computations may be incorrectly terminating or divergent. This is expressible by:

$$(\mathit{atb} \wedge \varphi) \supset \exists F(\mathit{ate} \wedge \psi)$$

c) P is partially $\forall$-correct with respect to (φ,ψ). Every terminating computation must satisfy ψ, but there is no guarantee of the existence of any terminating computation. This is expressible by:

$$(\mathit{atb} \wedge \varphi) \supset \forall G\ (\mathit{ate} \supset \psi)$$

d) P is totally $\forall$-correct with respect to (φ,ψ). Every computation terminates and satisfies ψ on termination. This is expressible by:

$$(\mathit{atb} \wedge \varphi) \supset \forall F(\mathit{ate} \wedge \psi)$$

A similar four way classification is given in [H] for the different notions of total correctness. The model of program to be studied here admits also a *failure* node which implies incorrect termination or abortion:

$$\begin{tikzcd}
 & & e \\
P'\!: b \arrow[r] & \boxed{\text{Program}} \arrow[ur] \arrow[dr] & \\
 & & f
\end{tikzcd}$$

The following four notions of total correctness of P' with respect to (φ,ψ) are possible:

A) For all inputs satisfying φ, all computations of P terminate and yield correct results, i.e. results satisfying ψ. This is expressible as:

$$(\mathit{atb} \wedge \varphi) \supset \forall F(\mathit{ate} \wedge \psi)$$

B) For all φ-inputs, every computation either terminates successfully or loops. No computation may fail.

$$(\mathit{atb} \wedge \varphi) \supset \forall G(\sim\mathit{atf} \wedge (\mathit{ate} \supset \psi))$$

C) For all φ-inputs each computation will either fail or reach e successfully. No infinite computations are allowed.

$$(\mathit{atb} \wedge \varphi) \supset \forall F(\mathit{atf} \vee (\mathit{ate} \wedge \psi))$$

D) For all φ-inputs there is at least one successful computation. Other computations may fail or diverge.

$$(\mathit{atb} \wedge \varphi) \supset \exists F(\mathit{ate} \wedge \psi)$$

Note that B) and C) did not guarantee the existence of a successful computation. To enforce this it is sufficient to add the clause $\exists F at b$ to the right hand side of the implications.

A Deductive System for $\mathcal{UB}$

Presented below is a deductive system of axioms and inference rules for proving validity of $\mathcal{UB}$ formulas. We take $\forall G$, G and $\forall X$ as primitive operators and define the other modalities by:

D1. $\forall Fp \equiv \sim\exists G\sim p$

D2. $\exists Fp \equiv \sim\forall G\sim p$

D3. $\exists Xp \equiv \sim\forall X\sim p$

Alternately we could have taken all six operators to be primitive but then we should add D1-D3 as axioms.

The axiom schemata are:

A1. $\vdash \forall G(p\supset q) \supset (\forall Gp\supset\forall Gq)$

A2. $\vdash \forall X(p\supset q) \supset (\forall Xp\supset\forall Xq)$

A3. $\vdash \forall Gp \supset \forall Xp \wedge \forall X\forall Gp$

A4. $\vdash \forall G(p\supset\forall Xp) \supset (p\supset\forall Gp)$

These four axioms are of universal character. The other four are existential.

E1. $\vdash \forall G(p\supset q) \supset (\exists Gp\supset\exists Gq)$

E2. $\vdash \exists Gp \supset p \wedge \exists X\exists Gp$

E3. $\vdash \forall Gp \supset \exists Gp$

E4. $\vdash \forall G(p\supset\exists Xp) \supset (p\supset\exists Gp)$

The rules of inference are:

R1. If p is an instance of a propositional tautology then $\vdash p$.

R2. If $\vdash p$ and $\vdash p\supset q$ then $\vdash q$ (Modus Ponens).

R3. If $\vdash p$ then $\vdash \forall Gp$ (Generalization).

Note in particular the two induction axioms A4 and E4. A4 states that if p is true at the root of a tree, and for every state s in the tree p is inherited by all immediate descendants of s, then p is true for all nodes in the tree.

E4 on the other hand states that if p is true at the root s_0, and everywhere in the tree p is inherited by at least one immediate descendant, then there exists a rooted path (i.e. an s_0-branch) all of whose nodes satisfy p.

In appendix A we present some useful theorems of this deductive system. In particular there is an additional induction theorem:

$$\vdash \exists G(p\supset\forall Xp) \supset (p\supset\exists Gp).$$

Semantic Tableaux for $\mathcal{UB}$

In this section we describe the construction of semantic tableaux formulas in $\mathcal{UB}$, obtain a decision procedure for satisfiability and prove the finite model property. In the next Section we use the tableaux construction to derive the completeness of the deductive system.

A structure is a triple (S,P,R) where S is a set of states, P is an assignment of formulas to states and R is a binary relation on states. It is convenient to use the same name S for a structure and its set of states. A structure differs from a model in that the assignment P is not restricted to propositions, nor is it required to always contain p or $\sim p$ for every p.

A formula is a nexttime formula if its principal connective is a modality $\forall X$ or $\exists X$. A formula is elementary if it is a proposition, the negation of a proposition or a nexttime formula. A formula which is not elementary is classified as an α-formula or a β-formula according to the table below. This table also defines certain subordinate formulas α_i and β_i.

A structure T is called a Hintikka set iff:

H1. $\sim p\in s \Rightarrow p\notin s$ (abbreviating $p\in s$ for $p\in P(s)$).

H2. $\alpha\in s \Rightarrow \alpha_1\in s$ and $\alpha_2\in s$, where α, α_1 and α_2 are instances of the entries in the table.

H3. $\beta\in s \Rightarrow \beta_1\in s$ or $\beta_2\in s$, where β,β_1,β_2 are instances of the entries in the table.

H4.a. If $\forall Xp\in s$ then s has at least one successor and for each t, a successor of s, $p\in t$.

b. If $\exists Xp \in s$ then there is t, a successor of s, such that sRt and $p \in t$.

c. If $\exists Fp \in s$ then there exists an s-branch b and a $t \in b$ such that $p \in t$.

d. If $\forall Fp \in s$ then for every s-branch b there is a $t \in b$ such that $p \in t$.

T is a Hintikka set <u>for</u> p if $p \in s$ for some $s \in T$.

α	α_1	α_2
$p \wedge q$	p	q
$\sim(p \vee q)$	$\sim p$	$\sim q$
$\sim\sim p$	p	p
$\forall Gp$	p	$\forall T \forall Gp$
$\exists Gp$	p	$\exists T \exists Gp$
$\sim\forall\left\{\begin{matrix}G\\F\\X\end{matrix}\right\}p$	$\exists\left\{\begin{matrix}F\\G\\X\end{matrix}\right\}\sim p$	$\exists\left\{\begin{matrix}F\\G\\X\end{matrix}\right\}\sim p$
$\sim\exists\left\{\begin{matrix}G\\F\\X\end{matrix}\right\}p$	$\forall\left\{\begin{matrix}F\\G\\X\end{matrix}\right\}\sim p$	$\forall\left\{\begin{matrix}F\\G\\X\end{matrix}\right\}\sim p$

β	β_1	β_2
$p \vee q$	p	q
$\sim(p \wedge q)$	$\sim p$	$\sim q$
$\exists Fp$	p	$\exists X \exists Fp$
$\forall Fp$	p	$\forall X \forall Fp$

A consequence of the definitions is:

<u>Proposition 1</u> (Hintikka's Lemma for UB): A UB formula p is satisfiable (has a model) iff it has a Hintikka set.

It is easy to check that any model is a Hintikka set when we extend P to cover all formulas which are true in a state. Similarly, any Hintikka set can be extended to form a model.

A semantic tableau is a systematic search for a Hintikka set. The tableau is constructed as a tree of nodes. Each node contains a set of formulas derived from the original formula p whose satisfiability we wish to check. Later we identify some of the nodes as states of a structure which will be shown to be a Hintikka set.

Notation: If n is a node of T then U_n is the set of formulas labelling n. A formula p may be marked as "checked" p'.

Let p be a UB formula and label the root of the tree T by $\{p\}$. The tableau is constructed inductively from the root by applying the following rules to nodes n which are leaves of T.

R_α: If $\alpha \in U_n$ then create m a son

$$U_m = (U_n - \{\alpha\}) \cup \{\alpha', \alpha_1, \alpha_2\}.$$

By this we mean-mark α as checked and add α_1 and α_2.

R_β: If $\beta \in U_n$ then create two sons, m_1 and m_2 of n and let:

$$U_{m_i} = (U_n - \{\beta\}) \cup \{\beta', \beta_i\},\ i = 1,2.$$

R_X: If all non-elementary formulas in n are checked, let

$$V_n = \{\exists Xp_1, \ldots \exists Xp_k, \forall Xq_1, \ldots \forall Xq\}$$

be the set of nexttime formulas in U_n.

Then, for each $i = 1, \ldots, k$ create a son m_i with

$$U_{m_i} = \{p_i, q_1, \ldots, q_\ell\}.$$

If $k = 0$, $\ell > 0$, create an only son m with

$$U_m = \{q_1, \ldots, q_\ell\}.$$

If $k = \ell = 0$ then node n is a terminal leaf; no further rules are applied to n.

A node n which is not a leaf is called an α-node, β-node or an X-node if the R_α-rule, R_β-rule or R_X-rule, respectively was applied at n.

Every X-node is defined to be a state. Two states s_1 and s_2 are related by R if s_2 is the first state encountered as a descendant of s_1 in T. The construction of T is kept finite by observing the two following termination rules:

T1: If a created node n contains both p and $\sim p$ then mark this node as

<u>closed</u> and do **not** expand it any further.

T2: If a state m is to be created as as a son of n, and there is a state t (which has already been created) elsewhere in the tableau such that $U_m = U_t$, then do not create m but connect n to t instead.

Since the number of distinct formulas that can appear in a U_n is finite, these two rules ensure termination. The resulting structure is no longer a tree because of T2 but is finite.

Consider the resulting structure T as a candidate for being a Hintikka set. It satisfies H1, H2, H3 and H4a,b but not necessarily H4c,d. Consequently we proceed to eliminate from T violations of H4-c,d. The elimination is performed by successive deletion of nodes. A similar procedure is used in [PR] for constructing a model for PDL. The deletion of nodes follows the rules below:

M1. Every node which contains both p and $\sim p$ for some p is deleted.

M2. If n is an α-node and m, its son, has been deleted, then delete n.

M3. If n is a β-node and both its sons, m_1 and m_2, have been deleted, then delete n.

M4. If n is an X-node and any of its descendants has been deleted, then delete n.

M5. Let $\exists Fp \in n$ which is also a state. If there is no path from n leading to a node t containing p, then delete n.

M6. Let $\forall Fp \in n$ which is also a state. If there is a maximal n-path b (a path which cannot be extended), such that

1) For all $t \in b$, $p \notin t$;

2) Every $\forall Fp$ node $t \in b$ (i.e. a node to which the β rule for $\forall Fp$ has been applied) has only one descendant;

then delete n.

<u>Lemma 1</u> The root n_0 of T is deleted by the elimination process iff $p \in U_{n_0}$ is is unsatisfiable.

$\Rightarrow$ The proof of the completeness theorem in the next section shows that if the root of T is deleted then the negation of the formula p in the root is provable. Since the proof system is sound, p is unsatisfiable.

$\Leftarrow$ We need to show that if the root is not deleted then p is satisfiable. By Proposition 1 it is sufficient to show that there is a Hintikka set for p. The only step that is not straightforward from the construction is to show that H4 holds for $\forall f$-formulas. In fact, the surviving part of T may not form a Hintikka set but we show that we can always construct a Hintikka set out of the surviving part of T. An <u>alternative node</u> for $\forall Fp$ is a β-node for $\forall Fp$ both of whose sons have survived.

Informally we unwind T so that every non-fulfilling branch is forced to eventually include the fulfilling son of some alternative node. We construct a structure T' whose nodes are instances of surviving nodes of T. Denote instances of $n \in T$ by $n', n'' \in T'$ etc. In the remainder of the proof, "nodes" refers only to surviving nodes.

n_0' is the root of T'. If n' is a leaf of T' then extend T' as follows, where $\pi(n_i', n_j')$ denotes the path from n_i' to n_j' in T'.

W1. If n is not an alternative node in T for any $\forall Fp$, then for every son m_i of n let m_i' be a son of n' in T'.

W2. If n is an alternative node in T for some $\forall Fp \in n$, let k be the number of instances of n in $\pi(n_0', n')$.

(i) If $k = 1$ then let m_1' (arbitrarily) be the son of n'

in T.

(ii) If $k > 1$ then if m_1'' (m_2'') was the son of n'' taken at the k-1'st instance n'' of n then let m_2' (m_1') be the son of n' in T'.

Thus we alternate our choices.

W3. If n has a previous instance n'' in $\pi(n'',n')$ and every alternative node t which has an instance in $\pi(n'',n')$ has at least two instances in $\pi(n'',n')$ then identify n' with n''.

The construction must ultimately terminate since there is only a finite number of alternative nodes. For suppose that $\pi = m', m_2', \ldots$ is an infinite path generated by the construction of T'. Then since T is finite there is a k such that for all $i \geqslant k$, m_i has infinitely many instances in $\pi_k = m_k', m_{k+1}', \ldots$ In particular, every alternative node in π_k must appear infinitely often and thus for some ℓ, m_k' and m_ℓ' are instances of the same node and every alternative node which appears in π_k appears at least twice in π_k. Thus W3 should have been applied to identify m_k' and m_ℓ'.

Let $\forall Fp \in n' \in T'$ and suppose that there is an n'-branch b in T' which does not fulfill $\forall Fp$. By construction (W2), no alternative node for $\forall Fp$ can appear more than once in b. Since T' is finite, eventually there must be nodes m', m'' such that no alternative node for $\forall Fp$ appears in $\pi(m',m'')$. By the tableau construction, if $\forall Fp \in n' \in b$ and b is not fulfilling then $\forall X \forall Fp$ propagates $\forall Fp$ to all nodes of b including m'. But then from the inverse image of $\pi(m',m'')$ we would be able to construct in T an infinite non-fulfilling path for $\forall Fp \in m$ containing no alternative nodes for $\forall Fp$, contradicting the assumption that m was not deleted by (M6).

Let $\exists Fp \in n' \in T'$. We show how to trace a fulfilling path in T'. By assumption there is a fulfilling path π_1 in T. As long as we don't reach an alternative node in π, then we can trace the path π_1' of instances in T'. Similarly, if $m_i \in \pi_1$ is an alternative node in T, $m_{i+1} \in \pi_1$ and $m_{i+1}' \in T'$. If T' contains the wrong son m_j' then if m_i has a next instance m_i'' in T' which is accessible from m_i' then add $m_i', \ldots, m_i''$ to π_1' and continue with the correct son by W2. If not, then note that if $\exists Fp \in m_i$, then $\exists Fp$ is in both sons of m_i. Find a fulfilling path π_2 in T from m_j and continue to trace a fulfilling path in T'. By assumption m_i has no instance in T' accessible from m_j' so eventually $\exists Fp$ is fulfilled or we start tracing π_3 for a <u>different</u> alternative node m_k. Since the number of (alternative) nodes is finite, eventually $\exists Fp$ is fulfilled. $\square$

<u>Proposition 2</u>. There is a decision procedure for satisfiability in $\mathcal{UB}$. $\mathcal{UB}$ has the finite model property.

Proposition 2 follows easily from the previous lemma and constructions. Note that if we are only interested in satisfiability then we do not need to construct the "unwound" tableau. It is sufficient to check whether the root node was deleted by the elimination process. The decision procedure including both the tableau construction and the deletion procedure can be shown to be exponential in the size of the formula.

<u>Completeness</u>

Let p be a valid $\mathcal{UB}$ formula. Create a tableau for $\{\sim p\}$. If the elimination procedure is applied then n_0 is eliminated, otherwise $\sim p$ would have been satisfiable. If $U_n = \{p_i\}$ is a set of formulas in a node n then $\underline{af}_n$, the associated formula of n, is $\bigvee_i(\sim p_i)$. In a typical proof of completeness by the tableau method (for example [RU]), one shows that $\vdash \underline{af}_n$ for every leaf and that provability is preserved as one ascends the tree to the root. In $\mathcal{UB}$ these meta-theorems hold but are not sufficient because of the non-fulfilling branches. For these we need induction axioms A4 and E4. We show in a series of lemmas that we can prove the af of every eliminated node. Since n_0 is eliminated, its af which is p is provable.

Lemma 2. If n is a closed leaf then $\vdash af_n$.

Proof. $\vdash \sim\sim p \vee \sim p$ by R1, hence $\vdash af_n$ by dilution. $\square$

Lemma 3. If $n \in T$ is an (i) α-node, (ii) β-node, (iii) X-node and (i) $\vdash af_m$, (ii) $\vdash af_{m_1}$ and $\vdash af_{m_2}$, (iii) $\vdash af_{m_i}$ for some i, then $\vdash af_n$.

Proof: For R_α and R_β the lemma follows by simple propositional reasoning and T13-T14 (Appendix A). For the R_x-rule, we need to use T9 to deduce

$$\vdash \forall Xq_1 \wedge \ldots \wedge \forall Xq_\ell \supset \forall X{\sim}p_i \quad \text{from}$$
$$\vdash \forall X\,(q_1 \wedge \ldots \wedge q_\ell \supset {\sim}p_i) \qquad \square$$

This part of the completeness proof, the definition of af and the proof that $\vdash af$ is preserved, is greatly simplieifed by the use of the nexttime operator whe compared with a classical completeness proof [RU]. Now we pay for this simplicity by the need to give the meta-theory for the induction axioms. In practice, this meta-theory is easy to apply and proofs can be constructed by the tableau method. Some of the techniques used below were first used in [PR].

Let t be a node (state) in T which was deleted by M5 because $\exists F{\sim}p$ was not fulfilled. Let $[t]^*$ be the set of states accessible from t by taking the $\exists X\exists F{\sim}p$ defined sons. For $u \in [t]^*$ let $[u]$ be the immediate successors of u in $[t]^*$ and let V_u be the set of all formulas q such that $\forall Xq$ is a universal nexttime formula in u. Let $W_u = \bigwedge_{q \in V_u} q$ and $W^t = \bigvee_{u \in [t]^*} W_u$. W^t is called the invariant of t.

Lemma 4: $\vdash W^t \supset \forall XW^t$

Lemma 5. For t as in lemma 4 and for all $u \in [t]^*$, $\vdash W_u \supset p$, hence $\vdash W^t \supset p$.

Proof:(4) For all $u \in [t]^*$ we can deduce $\vdash Wu \supset \bigvee_{v \in [u]} (\bigwedge_{q \in V_v} \forall Xq)$ and then by T9 $\vdash Wu \supset \bigvee_{v \in [u]} \forall X(\bigwedge_{q \in V_r} q)$. The lemma follows from the definitions using T10 to extract $\forall X$. $\square$

Proof:(5) Let n be the node which was obtained from u by applying the X rule to $\exists X\exists F{\sim}p$. $U_n = V_u \cup \{\exists F{\sim}p\}$. Without loss of generality we can assume that $\exists F{\sim}p \in n \Rightarrow {\sim}p \in n$ or $\exists X\exists F{\sim}p \in n$ is the first tableau rule applied at n. Then $U_{m_1} = V_u \cup \{{\sim}p\}$. Without loss of generality we can assume that $\exists F{\sim}p \in n \Rightarrow {\sim}p \in n$ or $\exists X\exists F{\sim}p \in n$ is the first tableau rule applied at n. Then $U_m = V_u \cup \{{\sim}p\}$. The node m_1 must be deleted, otherwise M5 would not have been applied. By the inductive hypothesis $\vdash af_{m_1}$ which is $\vdash W_u \supset p$. $\square$

From Lemma 4, generalization and A4, $\vdash W^t \supset \forall GW^t$. From Lemma 5, generalization and A1, $\vdash \forall GW^t \supset \forall Gp$. Trivially, $\vdash W_t \supset W^t$ since $t \in [t]^*$. Thus $\vdash W_t \supset \forall Gp$ which is af_n for some node n obtained from t by applying the R_x-rule. By Lemma 3 (iii), $\vdash af_t$.

If t was deleted by M5, let b be a branch as described there: $\forall F{\sim}p \in t$, ${\sim}p \notin n \in b$ and all fulfilling alternatives already deleted.

For each $u \in b$ let u' be the immediate successor of u in b. Denote by u^* the set of states in b accessible from u and by $V_{u'}$ the set of formulas in the node u'.

$V_{u'}$ will be $\{q_i \mid \forall Xq_i \in u\} \cup \{r\}$ for the r such that $\exists Xr \in u$ caused u' to be generated.

Let Z_u be the conjunction of all the formulas in $V_{u'}$ and $Z^t = \bigvee_{u \in t^*} Z_u$.

Lemma 6: $\vdash Z^t \supset \exists XZ^t$.

Proof: Like Lemma 4 except that T11 is used to deduce that $\vdash Z_u \supset \bigwedge_{q_i \in V_{u'}} \forall Xq_i \wedge \exists Xr$ implies $\vdash Z_u \supset \exists X(\wedge q_i \wedge r)$. $\square$

Lemma 7: $\vdash Z^t \supset p$.

Proof: By M6 nodes containing $\sim p$ were deleted; hence by induction their af's are provable. As in Lemma 5, $\vdash Z_\alpha \supset p$ hence $\vdash Z^t \supset p$. □

Using E4, E1 instead of A4, A1 we obtain $\vdash Z^t \supset \exists GZ^t$, $\vdash \exists GZ^t \supset \exists Gp$, $\vdash Z^t \supset \exists Gp$ and $\vdash af_t$. □

Proposition 3: A1-A4, E1-E4, R1-R3 form a complete deductive system for $\mathcal{UB}$.

The $\mathcal{UB}$ Semantics of Nondeterministic Programs

The utility of $\mathcal{UB}$ for proving the program properties so elegantly expressible in the language depends on the ability to restrict the class of possible models to the class of execution trees of a given program P. This is done by specifying a set of axioms which impose the structure of computation according to a given program on our general models. It may also be considered as specifying the temporal semantics of the programming language by connecting its syntactical constructs to transformations and developments in time.

In order to do this we extend our language by allowing predicates on variables.

We have three types of variables:

a) Computation variables, $y_1, y_2, \ldots$ which are modified by the execution and vary from state to state.

b) Free variables $x_1, x_2, \ldots$ which remain constant in time and are used to express relations between values of computation variables in different instances. Thus

$$(y=x) \supset \exists F(y=f(x))$$

is the expression of the statement that there exists some computation and some state in it such that the value of y in this state is equal to f of the initial y.

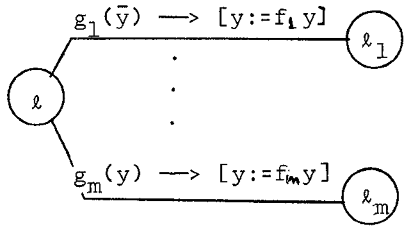
c) A program variable π whose value at any state points to the segment of program yet to be executed.

To emphasize the general principle underlying the semantics consider first an unstructured language in which programs are represented by transition graphs, $G = (N,E)$. The set N of nodes is called the set of locations. E is the set of edges each of which is labeled by a guarded instruction of the form

$$g(y) \to [y := f(y)]$$

with the meaning that this edge is enabled if $y(y)$ is true and passing through the edge involves the assignment of fy to y.

We form our temporal semantics of such programs by letting π range over N (the location set) and forming for each node a semantic formula. Let a node $\ell \in N$ admit the following transitions:


$g_1(\bar{y}) \longrightarrow [y:=f_1 y]$
ℓ_1
ℓ
$g_m(y) \longrightarrow [y:=f_m y]$
ℓ_m

Then we form the formula scheme

$$A_\ell: (\pi=\ell)\supset[\{\bigvee_{i=1}^{m}[g_i(y)\wedge Q(\ell_i,f_i y)]\} \equiv \exists XQ]$$

Here $Q = Q(\pi,y)$ is an arbitrary predicate depending in general on the program variable π and the computation variables y. It may also refer to free variables.

Note the presence of the '$\equiv$' connective which implies that this formula contains two implications. The first is stating that for every $i = 1,\ldots,m$ such that g_i is true there is a successor state s in which $\pi_s = \ell_i$ and $y_s = f_i$ (current y). The other implication is a complementary statement saying that the only possible successor states are derived in this fashion.

Next we will show that the temporal formalism is not necessarily restricted to unstructured languages. Consider as an illustration Dijkstra's language of guarded commands [D]. Here we let π range over program segments. S will often stand for an arbitrary program segment.

The semantics of an assignment statement may be given by:

$$(\pi = \{y := e;\ S\}) \supset [Q(\{S\}, e) \equiv \exists XQ(\pi,y)]$$

which states that provided we are about to execute {y := e; S}, Q will be true in the next instant iff $Q(\pi,y)$ is true for $\pi = \{S\}$ and $y = e$.

In the following let

$C = [\overset{m}{\underset{i=1}{\square}} g_i(y) \rightarrow c_i]$ where c_i is any command and g_i's are guards. The semantics of conditional is given by the axiom:

$$(\pi = \{C;S\}) \supset [\{\bigvee_{i=1}^{m} [g_i(y) \wedge Q(\{c_i;S\},y)]\} \equiv \exists XQ(\pi,y)]$$

It states that $Q(\pi,y)$ will be true of the next instant iff for some $i, g_i(y)$ is true and so is Q with $\pi = \{c_i;S\}$ and y.

Similarily for the repetitive command *C we have

$$(\pi = \{{*}C;S\}) \supset [\{\mathit{if}\ \bigvee_{i=1}^{m} g_i\ \mathit{then}\ Q(\{C;{*}C;S\},y)\ \mathit{else}\ Q(\{S\},y)\} \equiv \exists XQ(\pi,y)]$$

Here the computation step chooses between the case that some guard is true and the conditional C has to be executed first, and the case that all guards are false and we proceed beyond the repetition *C.

Discussion and Conclusions

In this paper we presented a unified branching time system which enjoys the joint advantages of both linear time and branching time systems, in being able to express and reason about the two basic types of termination, universal and existential. We have established the logical properties of the $\mathcal{UB}$ properties by presenting a decision algorithm and a complete deductive axiomatic system for the propositional fragment of the language. The decision procedure presented is obviously exponential.

This language must of course be compared with process logic languages such as PL[HKP] and its predecessors. These languages certainly can express any of the properties expressible in $\mathcal{UB}$ and many more. However, there is a price to pay for this expressibility which is the complexity of the language. A sign of this is the fact that PL is nonelementary (has nonelementary decision procedure) while $\mathcal{UB}$ is exponential.

Admittedly we do have six modal operators which is a disadvantage compared to simpler systems such as DX for linear time [GPSS] or the corresponding branching time systems. On the other hand the formation rules of these operators are simple and uniform, and they do enable us to express most of the interesting program properties discussed in the literature.

Another advantage lost in the transition from linear to branching time is expressive completeness in the sense of [GPSS]. Here the problem is inherent and cannot be remedied by the addition of one or two extra operators. In the full version of the paper we will bring a proof of the following:

<u>Proposition 4:</u> No branching time temporal language with a *finite* number of modal operators can be expressively complete.

This theorem, due to Gabbay (Unpublished manuscript) is based on the following observations:

a) A temporal language with a finite number of operators can always be translated into a first order formula with a number of distinct variable names which is fixed for the language.

b) In first order language it is easy to come up with formulas which need an arbitrarily large number of distinct variable names. Consider for example the statement:

There exist k time instants $t_1,\ldots,t_k$ no two of which are related.

This statement needs k variables for its expression for an arbitrary k. These formulas for sufficiently large k cannot therefore be expressed in any temporal logic.

Some recent works do indeed present branching time systems which are richer than ours and include additional operators. Such are the systems discussed in [EC] and [A2].

References

[A1] K.R. Abramson, Modal logic of concurrent nondeterministic programs, Symposium on Semantics of Concurrent Computations, Lecture Notes in Computer Science 70, Springer Verlag, Berlin, 1979-, 21-33.

[A2] K.R. Abrahamson, Decidability and Expressiveness of Logic of Processes, Ph.D. Thesis, University of Washington, August 1980.

[D] E.W. Dijkstra, Guarded commands, nondeterminancy and formal derivation of programs, C. ACM 18(8), 1975, 453-457.

[EC] E.A. Emerson and E.M. Clarke, Characterizing correctness properties of parallel programs using fixpoints, TR-04-80, Aiken Computation Laboratory, Harvard.

[F] R.W. Floyd, Nondeterministic algorithms, J. ACM 14(4), 1967, 636-644.

[FL] M.J. Fischer and R.E. Ladner, Propositional dynamic logic of regular programs, Journal of Computer and System Sciences 18(2), 1979, 194-211.

[GPSS] D. Gabbay, A. Pnueli, S. Shelah and J. Stavi, The temporal analysis of fairness, Seventh ACM Symposium on Principles of Programming Languages, 1980, 163-173.

[H] Harel, First Order Dynamic Logic, Lecture Notes in Computer Science 68, Springer-Verlag, Berlin, 1979.

[HKP] D. Harel, D. Kozen and R. Parikh, Process logic: expressiveness, decidability, completeness, 21th Symposium on Foundations of Computer Science, 1980.

[HC] G.E. Hughes and M.J. Cresswell, An Introduction to Modal Logic, Methuen, London, 1968.

[L] L. Lamport, "Sometime" is sometimes "not never", Seventh ACM Symposium on Principles of Programming Languages, 1980, 174-185.

[M1] Z. Manna, Mathematical theory of partial correctness, Symposium on Semantics of Algorithmic Languages, Lecture Notes in Mathematics 188, Springer Verlag, Berlin, 1971, 252-269.

[M2] Z. Manna, Second order mathematical theory of computation, Second ACM Symposium on Theory of Computing, 1970, 158-168.

[MP] Z. Manna and A. Pnueli, The modeal logic of programs, Automata, Languages and Programming, Lecture Notes in Computer Science 79, Springer-Verlag, Berlin, 1979, 385-409.

[PN] A. Pnueli, The temporal semantics of concurrent programs, Symposium on Semantics of Concurrent Computations, Lecture Notes in Computer Science 70, Springer Verlag, 1979, 1-20.

[PR] V.R. Pratt, A practical decision method for propositional dynamic logic, Tenth ACM Symposium on Theory of Computing, 1977, 326-337.

[RU] N. Rescher and A. Urquhart, Temporal Logic, Springer-Verlag, Vienna, 1971.

[S] R.M. Smullyan, First-Order Logic, Springer-Verlag, Berlin, 1968.

Appendix A: Discussion of $\mathcal{UB}$

If $\vdash \forall Gp \supset p$ (T1, below) is added to A1-A4 then we get a complete deductuve system for the universal fragment of branching time. If $\forall X$ and $\exists X$ are merged (along with $\forall G$ and $\exists G$) so that $\forall Xp \equiv {\sim}\forall X{\sim}p$ is an axiom then we get a complete deductive system for linear time. By T1 and T5, the axioms could be expressed more symmetrically as:

$$A3: \vdash \forall Gp \supset p \wedge \forall Xp \wedge \forall X \forall Gp$$

$$E2: \vdash \exists Gp \supset p \wedge \exists Xp \wedge \exists X \exists Gp.$$

Also, E3 can be derived by taking T1 and T6 as axioms. Some axiom of the form $\forall \supset \exists$ is needed to limit the models to non-ending time.

Theorems of $\mathcal{UB}$

T1. $\vdash \forall Gp \supset p$

T2. $\vdash \forall Gp \supset \forall Fp$

T3. $\vdash \forall X(p \supset q) \supset (\exists Xp \supset \exists Xq)$

T4. $\vdash \forall G(p \supset q) \supset (\forall Fp \supset \forall Fq)$

T5. $\vdash \exists Gp \supset \exists Xp$

T6. $\vdash \forall Xp \supset \exists Xp$

T7. $\vdash \forall G(p \wedge q) \equiv \forall Gp \wedge \forall Gq$

T8. $\vdash \exists G(p \wedge q) \supset \exists Gp \wedge \exists Gq$

T9. $\vdash \forall X(p \wedge q) \equiv \forall Xp \wedge \forall Xq$

T10. $\vdash \exists X(p \wedge q) \supset \exists Xp \wedge \exists Xq$

T11. $\vdash \forall Xp \wedge \exists Xq \supset \exists X(p \wedge q)$

T12. $\vdash \forall Gp \wedge \exists Gp \supset \exists G(p \wedge q)$

T13. $\vdash \forall Gp \equiv p \wedge \forall X \forall Gp$

T14. $\vdash \exists Gp \equiv p \wedge \exists X \exists Gp$

T15. $\vdash \forall Gp \equiv \forall G \forall Gp$

T16. $\vdash \exists Gp \equiv \exists G \exists Gp$

T17. $\vdash \exists G(p \supset \forall Xp) \supset (p \supset \exists Gp)$

T18. $\vdash \forall F \forall Gp \supset \forall G \forall Fp$

T19. $\vdash \exists G((p \vee \exists Gq) \wedge (\exists Gp \vee q)) \equiv (\exists Gp \vee \exists Gq)$

T20. $\vdash \forall X \forall Gp \equiv \forall G \forall Xp$

T21. $\vdash \exists X \exists Gp \supset \exists G \exists Xp$.

Comments: The proofs of T1-T12 are straightforward. T13-T21 are proved using induction axioms A4 and E4. It is also possible to prove derived rules: $\vdash p \rightarrow \vdash Mp$ and $\vdash p \supset q \rightarrow \vdash Mp \supset Mq$ for any modality M. We saw how T9 and T11 are used in the completeness proof to deduce the inductiveness of the invariants. T13 and T14 are the key to the tableau constructions: $\vdash \sim\exists G \supset \sim p \vee \sim\exists X \exists Gp$. To falsify $\exists Gp$, either p is false now or put off to tomoroow the task of falsifying $\exists Gp$.

T15-16 correspond to the transitivity **axioms** of the model system S4 [HC]. T17 is another induction axiom. We conjecture that replacing E4 by T17 results in a weaker system because the induction step needed $p \supset \forall Xp$ is too strong. The system is probably not different from linear time. T18 is our version of the S4.2 [HC] axiom $MLp \supset LMp$. Note that $\vdash \exists Gp \supset \forall Fp$ and $\vdash \forall F \exists Gp \supset \exists G \forall Fp$ can be proved but this is an artifact of the reflexiveness of $\mathcal{UB}$ and would not carry over if E2 were changed to $\vdash \exists Gp \supset \exists Xp \wedge \exists X \exists Gp$ as required classically in temporal logic [RU].

T19 is the S4.3 [HC] linearity axiom for $\exists G$. T20 show that $\forall X$ and $\forall G$ commute. For $\exists X$ and $\exists G$ only the direction shown in T21 holds.