Propositional Temporal Logics and their Use in Model Checking

Jeffery Zucker[1]

Department of Computer Science and Systems, McMaster University, Hamilton, Ontario L8S 4K1, Canada

Abstract. For the sake of proving correctness of programs with respect to their specifications, a number of formalisms exist. A traditional one has been proof systems involving Floyd-Hoare correctness formulae. More recently, especially with regard to concurrent programs such as air traffic control systems or operating systems, which are nonterminating and concurrent, and in connection with the desire for automatic verification, other formalisms have been found to be more useful. This paper, and the following one, survey two such types of formalism which have proved to be particularly successful for efficient automatic verification, or "model checking". In this paper we consider *branching time propositional temporal logics*, which serves as a good introduction for the more general formalism of the *propositional μ-calculus*, which is considered in the next paper. The emphasis is on a broad understanding rather than on technical details.

0 Introduction

Traditionally, *Floyd-Hoare formulae*, and associated proof systems, have been used for program specification and proofs of correctness. However for concurrent programs such as operating systems and air traffic control systems, which are *nondeterministic* and *nonterminating*, Floyd-Hoare systems have limited applicability.

The behaviour of such systems can be described by a family of *infinite execution traces, i.e., infinite branches of a computation tree.* A good formalism for investigating such structures is *propositional temporal logic.* This approach was already proposed in Pnueli (1977).

Temporal logic is a type of *modal logic*. It has *temporal modalities*, which permit the expression of properties such as *invariances*, *eventualities* and *precedences* (as we will see). This makes it suitable for parallel, nonterminating programs.

There are many versions of propositional temporal logic. We will consider, in particular, *branching time temporal logics.* Such logics describe partial ordered sets of *discrete instants.* At each instant, several *alternative futures* are possible.

So the models of this logic are *labelled directed graphs*, resembling Kripke structures for modal logics. The nodes represent *states* or *time instants* ("possible worlds" in Kripke semantics!), *labelled* with the set of *atomic propositions* true at that state.

---

[1]Research supported by a grant from the Natural Sciences and Engineering Research Council of Canada.

A *directed edge* from node s to node t means that it is possible to pass from s to t, or that t is a possible future from s.

Note that the truth of a formula at a state will depend, in general, on the truth of subformulas at *other states*. This contrasts with classical propositional logic, where the truth of a formula in a particular state depends only on the truth of subformulas at that state.

Details of the syntax and semantics of this logic will be given in the following sections. We give a brief sketch here.

Suppose that we have a concurrent system consisting of an n-tuple of *processes* $P_1, \ldots \ldots, P_n$, with some synchronization mechanism between them.

First, each process P_i is represented by a graph as above (a "flow graph", usually finite) as follows. The process is divided into a number of *blocks* or *regions*, so as to suppress details irrelevant to its synchronization behaviour with the other processes. Each such region is then represented by a node, and a *transition* between two such regions by an edge.

In this way, we construct a *synchronization skeleton* of the system.

Next, the concurrency behaviour of the system as a whole is represented by all *nondeterministic interleavings* of these transitions. In this way, we get an infinite *computation tree*, with infinite *computation paths.*

This formalism is useful for a large class of common problems, and, in addition, serves as a good introduction for the more general formalism of the *propositional μ-calculus*, which is considered in the following paper (Zucker 1992).

This and the following paper do not contain new results, nor are they intended as complete surveys of the field. Rather they are intended as introductions to the field for non-specialists. The references should be consulted for further reading.

This paper has benefitted enormously from the papers Clarke and Emerson (1981), Clarke, Emerson and Sistla (1983, 1986) and Emerson and Srinivasan (1988).

Thanks also to Allen Emerson, Ryszard Janicki and Peter Lauer for comments on earlier versions of this and the following paper.

1 The system $\boldsymbol{CTL}^*$

The strongest system of temporal logic we will consider is $\boldsymbol{CTL}^*$, which we investigate in this section. Historically, $\boldsymbol{CTL}^*$ was a development of a weaker system $\boldsymbol{CTL}$ (for "Computation Tree Logic"), to which we will turn in §5.

We now give the syntax of $\boldsymbol{CTL}^*$.

There are two classes of formulae: *state formulae* (true or false of states), and *path formulae* (true or false of paths).

Formulae are built up using the primitive *modalities* $\mathsf{E}p$ ("p holds for *some* path from the present state)", $\mathsf{X}p$ ("p holds from the *next* state on the current path)", and $p\mathsf{U}q$ ("at some state on the path q will hold, and *until* then, p holds").

The inductive definition of the class of formulae follows.

First, there is a class $\boldsymbol{PropAt}$ of *propositional atoms* $P, \ldots$.

The classes $\boldsymbol{StateForm}$ of state formulae and $\boldsymbol{PathForm}$ of path formulae are then defined by mutual induction:

S1. $\boldsymbol{PropAt} \subset \boldsymbol{StateForm}$.

S2. $p, q \in \boldsymbol{StateForm} \implies p \wedge q, \neg p \in \boldsymbol{StateForm}$.

S3. $p \in \boldsymbol{PathForm} \implies \mathsf{E}p \in \boldsymbol{StateForm}$.

P1. $\boldsymbol{StateForm} \subset \boldsymbol{PathForm}$.

P2. $p, q \in \boldsymbol{PathForm} \implies p \wedge q, \neg p \in \boldsymbol{PathForm}$.

P3. $p, q \in \boldsymbol{PathForm} \implies \mathsf{X}p, p\mathsf{U}q \in \boldsymbol{PathForm}$.

We can now define other useful modalities (with their informal meanings).

$$\begin{array}{rcll}
\mathsf{A}p & \equiv & \neg\mathsf{E}\neg p & \text{“}p\text{ holds for \textit{all} paths from present state”} \\
\mathsf{F}p & \equiv & \mathsf{true}\,\mathsf{U}p & \text{“}p\text{ holds at \textit{some} future state on current path”} \\
\mathsf{G}p & \equiv & \neg\mathsf{F}\neg p & \text{“}p\text{ holds at \textit{all} future states on current path”} \\
\mathsf{F}^\infty p & \equiv & \mathsf{GF}p & \text{“}p\text{ holds at \textit{infinitely many} future states on current path”} \\
\mathsf{G}^\infty p & \equiv & \mathsf{FG}p & \text{“}p\text{ holds at \textit{almost all} future states on current path”}
\end{array}$$

We can also define a “weak until” operator:

$$p\mathsf{W}q \ \equiv \ p\mathsf{U}q \vee \mathsf{G}p \qquad \text{“}p\text{ holds (at least) until }q\text{ holds”}$$

Note. For expository purposes, it will sometimes be convenient to (temporarily) add certain predicate constants, with specified meanings, to the language. (See, *e.g.*, §4.)

2 Formal semantics of $\boldsymbol{CTL}^*$

The semantics is defined relative to a *model* $\mathcal{M} = (S, R, L)$ where S is a set of *states* $s, s', \ldots,$ R is a binary relation on S such that

$$\forall s \exists s' (sRs'), \tag{1}$$

(s' is a “successor” state to s), and L is a *labelling* which assigns to each state a set of propositional atoms, *i.e.*, $L : S \to \mathcal{P}(\boldsymbol{PropAt})$ (where $\mathcal{P}$ is the power set , so that $L(s)$ is the set of propositional atoms taken to be *true* at state s. A *full path* is an infinite sequence

$$x = s_0, s_1, s_2, \ldots$$

such that for all i, $s_i R s_{i+1}$. Note that R represents the “successor” relation in the computation tree of some program.

Remark 1. The elements of R are called *state transitions*. The condition (1) on R is not required by all authors. In its absence we would define a *full path* as a maximal (not necessarily infinite) R-related path. The resulting theory would not be significantly different.

Now we will define the *satisfaction predicates* for state and path formulae, *i.e.*, the two notions

$$\mathcal{M}, s \models p, \qquad p \text{ a state formula}$$
$$\mathcal{M}, x \models p, \qquad p \text{ a path formula}$$

by mutual recursion on the complexity of p. We let x be the full path $s_0, s_1, \ldots,$ and for any i, x^i denotes the suffix full path $s_i, s_{i+1}, \ldots$. (And we drop the '$\mathcal{M}$' where it is understood.)

S1. $s \models P \iff P \in L(s)$ for P atomic.
S2. $s \models p \wedge q \iff s \models p$ and $s \models q$.
$s \models \neg p \iff$ not $s \models p$.
S3. $s \models \mathsf{E}p \iff$ for some full path y starting from s, $y \models p$.

P1. $x \models p \iff s_0 \models p$ for p a state formula.
P2. $x \models p \wedge q \iff x \models p$ and $x \models q$.
$x \models \neg p \iff$ not $x \models p$.
P3. $x \models \mathsf{X}p \iff x^i \models p$.
$x \models p\mathsf{U}q \iff \exists i \left[x^i \models q \text{ and } \forall j < i \, (x^j \models p) \right]$.

Definitions. Let p be a state formula.

(1) p *is satisfiable in* $\mathcal{M}$ iff $\mathcal{M}, s \models p$ for some state s in $\mathcal{M}$.

(2) p is *satisfiable* iff it is satisfiable in some model.

(3) $\models p$ (p is *valid*) iff $\mathcal{M}, s \models p$ for every model $\mathcal{M}$ and state s in $\mathcal{M}$.

Remark 2. To correlate this semantic definition with the one in the following paper, we can use the notation

$$[\![p]\!] \;=_{df}\; \{s \mid s \models p\}.$$

(See Remark 1 in §3, *loc. cit.*)

Remark 3. The temporal operators of $\boldsymbol{CTL}^*$ (E, X and U) are all *future temporal operators*, which means that the truth at some state s of a formula p defined with these operators depends only on the truth of subformulae of p *at s or later states.*

3 Fixed point characterizations

We first define some notation. For two formulas p and q, '$p = q$' means that p and q are *semantically equivalent*, *i.e.*, for all models M and states s: $\mathcal{M}, s \models p \iff \mathcal{M}, s \models q$.

We will now display a series of semantic equivalences, some of which will be used later in the model checking section, and all of which are instances of the general theory of fixed points to be presented in the following paper.

These equivalences should be intuitively clear form the meanings of the modal operators given in §1.2. (Note that below by "future", we mean "present or future".)

$$\underline{\mathsf{EG}p} \;=\; p \wedge \mathsf{EX}\underline{\mathsf{EG}p}. \tag{1}$$

The l.h.s. here says that for *some* path from the present state, p holds *everywhere* on that path. The r.h.s. says that p holds at the present state, *and also*, at *some* successor state ("EX"), for *some* path from that state, p holds *everywhere* on that path. So $\mathsf{EG}p$ is a solution of the equation (in the propositional variable Z) $Z = p \wedge \mathsf{EX}Z$, or a *fixed point* of the functional $\lambda Z[\,p \wedge \mathsf{EX}Z]$. (We have used underlining to clarify this.)

$$\underline{\mathsf{AG}p} \;=\; p \wedge \mathsf{AX}\underline{\mathsf{AG}p} \tag{2}$$

The l.h.s. says that for *every* path from the present state, p holds *everywhere* on that path. (In other words, p holds at *all* future states.) The r.h.s. says that p holds at the present state, *and also*, at *every* successor state ("AX"), p holds at *all* states future to it. So $\mathsf{AG}p$ is a fixed point of $\lambda Z[\,p \wedge \mathsf{AX}Z]$.

In the following 4 cases, the reader should similarly be able to provide the functional for which the expression on the l.h.s. is a fixed point.

$$\underline{\mathsf{EF}p} \;=\; p \vee \mathsf{EX}\underline{\mathsf{EF}p} \tag{3}$$

The l.h.s. says that for *some* path from the present state, p holds *somewhere* on that path. (In other words, p holds at *some* future state.) The r.h.s. says that *either* p holds at the present state, *or* at *some* successor state ("EX"), p holds at *some* state future to it.

$$\underline{\mathsf{AF}p} \;=\; p \vee \mathsf{AX}\underline{\mathsf{AF}p} \tag{4}$$

The l.h.s. says that for *every* path from the present state, p holds *somewhere* on that path. (In other words, p *must hold* at some future point.) The r.h.s. says that *either* p holds at the present state, *or* at *every* successor state ("AX"), p *must hold* at some point future to it.

The reader should be able to give informal meanings for the last two examples:

$$\underline{\mathsf{E}(p\mathsf{U}q)} \;=\; q \vee (p \wedge \mathsf{EX}\underline{\mathsf{E}(p\mathsf{U}q)}) \tag{5}$$

$$\underline{\mathsf{A}(p\mathsf{U}q)} \;=\; q \vee (p \wedge \mathsf{AX}\underline{\mathsf{A}(p\mathsf{U}q)}) \tag{6}$$

In the following paper, we will return to the topic of fixed points, and also distinguish between least and greatest fixed points.

4 Safety, Liveness and Fairness Properties

With the operators of $\boldsymbol{CTL}^*$, we can express some important properties of programs.
1. Safety: This means that the system will never reach an undesirable state. This is typically an *invariance* assertion $\mathsf{AG}p$ for some property p. An example is *mutual exclusion*, which can be expressed by $\mathsf{AG}(\neg\mathsf{cs}_1 \vee \neg\mathsf{cs}_2)$, where cs_i expresses that process number i is in its critical state ($i = 1, 2$).
2. Liveness: This means that the system will always reach a desirable state. This is typically an *inevitability* assertion $\mathsf{AF}p$ for some property p. An example is *absence of starvation* of process 1: $\mathsf{try}_1 \Rightarrow \mathsf{cs}_1$, where try_1 expresses that process 1 is in its "trying region".
3. Fairness properties: Consider the two path assertions: '$\mathsf{enabled}_i$' and '$\mathsf{executed}_i$', which say, respectively, that process i is enabled, and process i is executed (for $i = 1, \ldots, m$). We can then express the following properties of a computation path:

(a) The path is *impartial* or *unconditionally fair*: $\bigwedge_{i=1}^{m} \mathsf{F}^{\infty}\mathsf{executed}_i$.
(b) The path is *just* or *weakly fair*: $\bigwedge_{i=1}^{m}(\mathsf{G}^{\infty}\mathsf{enabled}_i \implies \mathsf{F}^{\infty}\mathsf{executed}_i)$.
(c) The path is *strongly fair*: $\bigwedge_{i=1}^{m}(\mathsf{F}^{\infty}\mathsf{enabled}_i \implies \mathsf{F}^{\infty}\mathsf{executed}_i)$.

5 The subsystem $\boldsymbol{CTL}$ of $\boldsymbol{CTL^*}$

For many purposes, it is found convenient to work with a subsystem of the language $\boldsymbol{CTL^*}$. This subsystem, presented, *e.g.*, in Clarke and Emerson (1981), has only *state formulae*, which use the temporal modalities only in certain combinations:

$$p \ ::= \ P \mid \mathsf{true} \mid p \wedge q \mid \neg p \mid \mathsf{EX}p \mid \mathsf{AX}p \mid \mathsf{E}(p\mathsf{U}q) \mid \mathsf{A}(p\mathsf{U}q). \tag{1}$$

From these we can define:

$$\begin{aligned}
\mathsf{AF}p &\equiv \mathsf{A}(\mathsf{true}\,\mathsf{U}\,p)\\
\mathsf{EF}p &\equiv \mathsf{E}(\mathsf{true}\,\mathsf{U}\,p)\\
\mathsf{EG}p &\equiv \neg\mathsf{AF}\neg p\\
\mathsf{AG}p &\equiv \neg\mathsf{EF}\neg p
\end{aligned}$$

– and hence *safety* and *liveness* properties (see §1.2). We can also define, *e.g.*,

$$\begin{aligned}
\mathsf{E}(p\mathsf{W}q) &\equiv \mathsf{E}(p\mathsf{U}q \vee \mathsf{G}p)\\
&\equiv \mathsf{E}(p\mathsf{U}q) \vee \mathsf{EG}p
\end{aligned}$$

However, we cannot, in general, express *fairness* properties (*loc. cit.*) in $\boldsymbol{CTL}$.

In the rest of the paper, we will concentrate on $\boldsymbol{CTL}$. It is simpler than $\boldsymbol{CTL^*}$, but sufficiently expressive so that the model checking property for it is interesting, as we will see.

Remark. A mild modification of $\boldsymbol{CTL}$ is presented in Clarke and Emerson (1981) There the concurrent processes $P_1, \ldots, P_n$ are represented explicitly in the syntax (compare with (1)):

$$p \ ::= \ \ldots \mid \mathsf{EX}_i p \mid \mathsf{AX}_i p$$

($i = 1, \ldots, n$), where X_i refers to a state following a step by process P_i, and the edges of the flow graph are labelled with the process numbers. The semantics are modified in an obvious way.

6 The model checking algorithm for $\boldsymbol{CTL}$

The model checking problem for a temporal logic may be stated as follows. *Given a formula p_0 and a model $\mathcal{M}$, is p satisfiable in $\mathcal{M}$?* Or, more specifically: *Given also a state s in $\mathcal{M}$, does $\mathcal{M}, s \models p_0$?*

We will show how to solve this problem for $\boldsymbol{CTL}$, using essentially the algorithm in Clarke, Emerson and Sistla (1986).

Finiteness Assumption. *We assume that $\mathcal{M}$ is finite.* (This just means that S is finite.)

This assumption is reasonable, since: (i) it is often the case in practice; and (ii) many modal systems have the *Finite Model Property*: If p is satisfiable, then p is satisfiable by a finite model. (See Stirling 1992.)

Notation. For any formula p, $\boldsymbol{state}_{\mathcal{M}}(p)$ is the set of states in $\mathcal{M}$ which satisfy p. (We will generally drop the subscript '$\mathcal{M}$'.)

Note that $\boldsymbol{state}(p) = [\![p]\!]$, in the notation of §2, Remark 2.

Suppose, then, we are give a finite model $\mathcal{M} = (S, R, L)$, and a formula p_0 of $\boldsymbol{CTL}$. We want to test whether p_0 is satisfied in $\mathcal{M}$, or, more specifically, which states are in $\boldsymbol{state}(p_0)$. In fact our algorithm will label each state s of $\mathcal{M}$ with those subformulas of p_0 which hold at s.

We will show how to compute $\boldsymbol{state}(p)$ for all *subformulae* p of p_0 (including p_0 itself), by structural induction on p. (Compare the satisfaction definition in §2.)

Case 1. $p \in \boldsymbol{PropAt}$. Then $\boldsymbol{state}(p) = \{s \in S \mid p \in L(s)\}$.

Case 2. $p \equiv q_1 \wedge q_2$. Then $\boldsymbol{state}(p) = \boldsymbol{state}(q_1) \cap \boldsymbol{state}(q_2)$.

Case 3. $p \equiv \neg q$. Then $\boldsymbol{state}(p) = S \setminus \boldsymbol{state}(q)$.

Case 4. $p \equiv \mathsf{EX}q$. Then $\boldsymbol{state}(p) = \{s \in S \mid \exists t : sRt : t \in \boldsymbol{state}(q)\}$.

Case 5. $p \equiv \mathsf{AX}q$. Then $\boldsymbol{state}(p) = \{s \in S \mid \forall t : sRt : t \in \boldsymbol{state}(q)\}$.

Case 6. $p \equiv \mathsf{E}(q_1 \mathsf{U} q_2)$. This case and the next are the most interesting. Recall equation (5) in §1.3:

$$\underline{\mathsf{E}(q_1 \mathsf{U} q_2)} = q_2 \vee (q_1 \wedge \mathsf{EX}\underline{\mathsf{E}(q_1 \mathsf{U} q_2)}). \tag{1}$$

So p is a fixed point—in fact, the least fixed point—of $\lambda Z[q_2 \vee (q_1 \wedge \mathsf{EX}Z)]$. Hence we will construct $\boldsymbol{state}(p)$ by "approximating from below", *i.e.*, as the union of an increasing sequence of sets

$$\boldsymbol{state}_0(p) \subseteq \boldsymbol{state}_1(p) \subseteq \boldsymbol{state}_2(p) \subseteq \ldots$$

where the sets $\boldsymbol{state}_k(p)$ are constructed in stages. At Stage k, $\boldsymbol{state}_k(p)$ is constructed as follows.

Stage 0*:*

$$\boldsymbol{state}_0(p) =_{df} \boldsymbol{state}(q_2).$$

Stage $k+1$*:*

$$\boldsymbol{state}_{k+1}(p) =_{df} \boldsymbol{state}_k(p) \cup \big(\boldsymbol{state}(q_1) \cap \{s \mid \exists t : sRt : t \in \boldsymbol{state}_k(p)\}\big). \tag{2}$$

Since S is finite (by the Finiteness Assumption), for some $k \leq \|S\|$ (the cardinality of S),

$$\boldsymbol{state}_k(p) = \boldsymbol{state}_{k+1}(p) = \ldots = \boldsymbol{state}(p).$$

So we continue constructing these approximations $\boldsymbol{state}_k(p)$ to $\boldsymbol{state}(p)$ until $k = \|S\|$, or until we notice that the approximations are not growing any more.

Case 7. $p \equiv \mathsf{A}(q_1 \mathsf{U} q_2)$. This is like case 6, except that in equation (1) 'E' is replaced by 'A', and in equation (2) '$\exists$' is replaced by '$\forall$'.

In order to have a better intuitive understanding of these last two cases, let us consider important special cases of them:

Case 6′. $p \equiv \mathsf{EF}q \equiv \mathsf{E}(\mathsf{true}\,\mathsf{U} q)$. This just says that p will hold at *some* future state on the current path. Now (1) becomes

$$\underline{\mathsf{EF}q} \;=\; q \vee \mathsf{EX}\underline{\mathsf{EF}q}. \tag{1′}$$

So p is now the least fixed point of $\lambda Z[q \vee \mathsf{EX}Z]$. So we construct $\boldsymbol{state}(p)$ as the union of approximations

$$\boldsymbol{state}_0(p) \;\subseteq\; \boldsymbol{state}_1(p) \;\subseteq\; \boldsymbol{state}_2(p) \;\subseteq \ldots$$

constructed in stages as follows.

Stage 0*:*

$$\boldsymbol{state}_0(p) \;=_{df}\; \boldsymbol{state}(q).$$

Stage $k+1$*:*

$$\boldsymbol{state}_{k+1}(p) \;=_{df}\; \boldsymbol{state}_k(p) \;\cup\; \{s \mid \exists t : sRt : t \in \boldsymbol{state}_k(p)\}. \tag{2′}$$

Again, $\boldsymbol{state}(p) = \boldsymbol{state}_k(p)$ for some $k \leq \|S\|$.

Note that in this case, $\boldsymbol{state}(p)$ consists of those states s such that *for some path* x from s, p holds at some state on x *at most* k *steps away* from s.

Case 7′. $p \equiv \mathsf{AF}q \equiv \mathsf{A}(\mathsf{true}\,\mathsf{U} q)$. This just says that p will hold at *all* future states on the current path. This is like Case 6′, with 'A' replacing 'E' in (1′) and '$\forall$' replacing '$\exists$' in (2′).

Note that in this case, $\boldsymbol{state}(p)$ consists of those states s such that *for all paths* x from s, p holds at some state on x *at most* k *steps away* from s.

7 Time complexity of the algorithm

The algorithm is remarkably efficient, with time complexity $O(|p_0| \times |\mathcal{M}|)$ (in the notation of the previous section), where $|p_0|$ is the length of the formula p_0 and $|\mathcal{M}|$ is the *size* of the model $\mathcal{M} = (S, R, L)$, defined as $|\mathcal{M}| = \|S\| + \|L\|$. The factor $|p_0|$ comes in because of the structural induction on (subformulas of) p_0, and the factor $|\mathcal{M}|$ comes in with cases 6 and 7 of the algorithm, which involves up to $\|S\|$ stages, and (over all stages) an examination of up to $\|R\|$ possible successor states t of s (see equation (2)).

8 Conclusion

We have observed *regularities* in the languages $\boldsymbol{CTL}^*$ and $\boldsymbol{CTL}$, leading to *fixed point characterizations* (see §1.3). In fact, the model checking algorithm (§3) was based on this.

In the following paper, the notion of fixed point will become central; in fact it will be basic to the syntax and semantics of the language!

References

Clarke, E.M., Emerson, E.A. (1981): Design and synthesis of synchronization skeletons using branching time temporal logic. Proc. Workshop on Logics of Programs, Yorktown Heights, NY. Lecture Notes in Computer Science, 131, Springer-Verlag

Clarke, E.M., Emerson, E.A., Sistla, A.P. (1983): Automatic verification of finite state concurrent systems using temporal specifications: A practical approach. Proc. 10th Annual ACM Symp. on Principles of Programming Languages, Austin, TX, 117–126.

Clarke, E.M., Emerson, E.A., Sistla, A.P. (1986): Automatic verification of finite state concurrent programs using temporal logic. ACM TOPLAS **8**, 244–263.

Emerson, E.A. (1990): Temporal and modal logic. In: Handbook of Theoretical Computer Science, Vol. B: Formal Models and Semantics (ed. J. van Leeuwen), Elsevier, 995–1072.

Emerson, E.A., Srinivasan, A.J. (1988): Branching time temporal logic. In: Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency: School/Workshop, Noorwijkerhout, The Netherlands, May/June 1988 (ed. J.W. de Bakker, W.P. de Roever, G. Rozenberg). Lecture Notes in Computer Science, 354, Springer-Verlag.

Pnueli, A. (1977): The temporal logic of programs. 18th Annual Symp. on the Foundations of Computer Science, Providence, RI, 46–57.

Stirling, C. (1992): modal and temporal logics. In: Handbook of Logic in Computer Science (ed. S. Abramsky, D. Gabbay, T. Maibaum), Oxford University Press, to appear.

Zucker, J. (1992): The propositional μ-calculus and its use in model checking. In this volume.

Added in proof. A good survey of this topic is given in Emerson (1990), which unfortunately we only became aware of after completing this paper.