CTL and Equivalent Sublanguages of CTL*

Klaus Schneider

*University of Karlsruhe, Department of Computer Science,
Institute for Computer Design and Fault Tolerance*

(Prof.Dr. D. Schmid)

P.O. Box 6980, 76128 Karlsruhe, Germany,

Phone: +49 721 608 3641,

e-mail: Klaus.Schneider@informatik.uni-karlsruhe.de,

<http://goethe.ira.uka.de/people/schneider>

Abstract

Specifications of finite state algorithms with a complex temporal behavior such as protocols or hardware systems are often given in temporal logics as e.g. CTL or CTL. On the one hand, CTL offers in comparison to CTL* efficient model checking algorithms, but on the other hand, CTL seems to suffer from a limited expressiveness. In this paper, it is shown that the limitations one feels when using CTL are often due to syntactical restrictions and not due to its limited expressiveness. Hence, a new sublanguage of CTL* is presented that has the same expressiveness as CTL, but less syntactical restrictions. Moreover, it is shown, how specifications written in this language can be translated into CTL to be verified by a standard model checker for CTL.*

Keywords

Formal Methods, Temporal Logic, Model Checking

1 INTRODUCTION

Temporal logics are convenient formalisms for specifying systems with a complex temporal behavior. There are a lot of different temporal logics which differ in their semantics as well as in their expressiveness (for an excellent overview see [Emer90]). Some temporal logics consider the future as well as the past, while others only have future time temporal operators. Another difference in the semantics is the modeling of the considered points of time: usually discrete points of time are considered, but for special purposes, there are also temporal logics with a continuous model of time.

The most frequently used temporal logics are CTL [CIEm81], LTL [Pnue77], and the superset CTL* of the latter ones [EmHa86]. These logics have a discrete model of time and usually only have future time operators. Models of these logics are often given as (finite) state transition diagrams called Kripke structures [Emer90]. As these

Kripke structures are used to model the systems to be verified, each state of the Kripke structure is also a possible system state. Each path through a Kripke structure is called a computation path and directly corresponds to a possible computation of the modeled system. Usually, it is assumed that each transition in the Kripke structure requires one unit of time and corresponds to a basic computation of the modeled finite state system.

CTL and CTL* are branching time temporal logics, i.e. in each program state, there can be more than one successor state and hence, several computation paths are starting in this state. This feature can be used to model inputs that can not be predicted in general. In CTL*, we can (and in CTL we *must*) specify in each state, whether a temporal property has to hold for all or for at least one computation path starting in this state. We do not have this choice in LTL: if a LTL formula is interpreted over a Kripke structure, it is required that the formula has to hold for all paths of the Kripke structure. Hence, in LTL, the Kripke structure is viewed as a set of paths, rather than a state transition system.

The past decade has seen an extensive debate [Lamp80, EmLe85, LiPn85] and [EmHa86] whether branching time or linear time temporal logics are more suited for the specification and verification of finite state systems. In general, LTL specifications tend to be more readable than CTL specifications, since LTL directly allows to formalize properties with more than one event, as temporal operators may be nested arbitrarily. For example, the property ‘*b* has to hold, when *a* is true for the second time’ can be expressed* in LTL as $[(X[b \ W \ a]) \ W \ a]$. The same is allowed also in CTL*, but not in CTL.

Nevertheless, CTL plays the mayor role in the verification of temporal properties of finite state programs due to the efficiency of its model checking algorithm: checking that a CTL formula φ holds in a Kripke structure $\mathcal{M}$ with $\|\mathcal{S}\|$ states and $\|\mathcal{R}\|$ transitions can be done in time $O((\|\mathcal{S}\| + \|\mathcal{R}\|) \cdot \|\varphi\|)$, while the model checking problems for LTL and CTL* are known to be PSPACE-complete [SiCl85, EmLe85]. Moreover, model checkers for CTL can use a symbolic state traversal [BCMD92] which avoids to enumerate the set of states as well as the transition relation (symbolic traversals represent sets of states by their characteristic function in form of binary decision diagrams [Bry86]). This has lead in the past to a breakthrough in the verification of finite state systems and allowed to fix several bugs [BCDM86, DiCl86, CGHJ93a]. Symbolic model checking can not directly be applied to LTL or CTL*. However, there are some approaches to remedy this fact [CIGH94a, Jong91a, Schn96c] by translating the formulae to equivalent ω -automata.

In some papers, it has been argued that CTL is not expressive enough for the specification of complex temporal behavior. In particular, a major drawback of CTL is its inability to express fairness, i.e. that a property holds infinitely often on a path. It has been proved that the CTL* formula $EGFx$ is not equivalent to any CTL formula, hence CTL cannot express the fact that there is a path where a property x holds infinitely often. However, the model checking algorithm for CTL has been extended in [EmLe86] to handle fairness constraints, i.e. to restrict the path quantifiers only

*The semantics of the temporal operators is given in the following.

to quantify over fair paths with respect to a set of given fairness constraints without loss of the linear model checking algorithm. Nevertheless, even if a property is expressible in CTL, it is sometimes not obvious how. For example, in [JoMC94], the verification of a ‘single pulser circuit’ has been studied in various verification systems. CTL is expressive enough to formalize the behavior of the single pulser circuit, but in [JoMC94] some difficulties for its specification in CTL have been reported, which are mainly due to the hard syntactical restrictions of CTL.

Hence, in this paper it is stated that the expressiveness of CTL (with the extension of fairness constraints) is sufficient for the formalization of most specifications. Nevertheless, it must be admitted that it is sometimes hard to write down specifications in CTL directly, especially if one is not an expert in the research in this area. Hence, in this paper a subset of CTL* called LeftCTL* is defined that has less restrictions on the syntax, but the same expressiveness as CTL. This allows to write down specifications more directly. A translation procedure from LeftCTL* to CTL is also presented such that standard CTL model checkers as SMV [McMi93a] can be used for model checking of LeftCTL* specifications. In particular, this allows the translation of the LTL specification of the single pulser into an equivalent CTL specification. However, the linear runtime of the model checking procedure can not be stated for LeftCTL* since LeftCTL* is in general more succinct than CTL. This means that there are LeftCTL* formulae where each equivalent CTL formula is more than exponentially longer.

It is remarkable that LeftCTL* is also related in some way to LTL. The better readability of LTL in comparison to CTL is a consequence of the arbitrary deep nesting of temporal operators. While CTL requires that path quantifiers and temporal operators are coupled to each other, this rule is weakened to a large extent in the definition of LeftCTL* similar to LTL. In fact, a lot of LTL formulae are also LeftCTL* formulae.

While LeftCTL* is as expressive as CTL, it is also possible to enhance the expressiveness of CTL similar to [BeGr94a]. In [BeGr94a], a temporal logic called CTL² has been presented that is on the one hand a proper subset of CTL*, but on the other hand a proper superset of CTL (both in terms of syntax and expressiveness). In contrast to CTL, CTL² allows at most one nesting of temporal operators. Although CTL² is more expressive than CTL, it has still a polynomial model checking algorithm. One key of the model checking algorithm of CTL² is the elimination of these operator nestings. The underlying theorems for eliminating these nestings can also be applied to LeftCTL*. This allows to extend LeftCTL* analogous to the extension of CTL to CTL², such that a logic is obtained that is a superset of both LeftCTL* and CTL². The translation algorithm presented in this paper is then able to translate this logic back to CTL². These extensions are considered in section 5 of the paper.

The outline of the paper is as follows: in the next section, the sublanguages of CTL* that will be considered throughout the paper are defined. Section 3 and section 4 present the reduction of LeftCTL* to CTL, and in section 5 some further enhancements of LeftCTL* analogous to [BeGr94a] are considered. Section 6 illustrates the presented methods by translating the single pulser [JoMC94] specifications from LeftCTL* to CTL.

2 TEMPORAL LOGICS

In general, the formulae of a logic depend on the set of available variables and on the set of available operators. In temporal logics, there are three kinds of operators: boolean operators, temporal operators and path quantifiers.

Boolean operators such as $\neg, \vee, \wedge, \rightarrow, \leftrightarrow, \overline{\wedge}, \overline{\vee}$ or $\oplus$ map given boolean values to boolean values. The mapping, i.e. the semantics of these boolean operators can be given by well-known truth tables. Clearly, there are $2^4 = 16$ different boolean binary operators, but is well-known that all boolean operators can be expressed by one of the bases $\{\overline{\wedge}\}$, $\{\overline{\vee}\}$, $\{\neg, \wedge\}$, $\{\neg, \vee\}$, $\{\neg, \wedge, \vee\}$ or for Reed-Muller normal forms $\{\wedge, \oplus\}$. In the following the operators $\neg, \wedge$ and $\vee$ are used with their intuitive semantics.

Temporal operators map signals on signals, where a signal is defined to be a function from natural numbers to boolean values. Temporal relations between signals can be expressed by temporal operators. For example, $[x \text{ W } b]$ holds at a certain point of time t_0 iff x holds when b holds for the first time after t_0 . If b never holds after t_0 , then $[x \text{ W } b]$ holds trivially. A lot of other temporal operators can be defined in terms of W , e.g.

$$\begin{aligned} [x \text{ U } b] &= [b \text{ W } (x \rightarrow b)] & [x \underline{\text{U}} b] &= [x \text{ U } b] \wedge Fb \\ [x \text{ B } b] &= [(\neg b) \text{ W } (x \vee b)] & [x \underline{\text{B}} b] &= [x \text{ B } b] \wedge Fb \\ [x \text{ AT } b] &= \text{X}[x \text{ W } b] & [x \underline{\text{W}} b] &= [x \text{ W } b] \wedge Fb \\ Gx &= [0 \text{ W } (\neg x)] & Fx &= \neg[0 \text{ W } x] \end{aligned}$$

U is the ‘until’-operator, i.e. $[x \text{ U } b]$ holds at t_0 iff x holds until b becomes true for the first time after t_0 . B is the ‘before’-operator, i.e. $[x \text{ B } b]$ holds whenever x holds before b holds. Gx holds at a certain point of time, iff x holds from this point on; and Fx holds at a certain point of time, iff x holds at least once after this point. For each of the ‘event-oriented’ temporal operators W , U , B and AT , there is a strong variant which is equivalent to the weak version, except that it is required that the event has to occur. Strong variants are underlined in this paper to distinguish them from their weak variants. Usual operator bases for temporal operators are e.g. $\{\text{AT}\}$, $\{\underline{\text{AT}}\}$, $\{\text{W}, \text{X}\}$, $\{\underline{\text{W}}, \text{X}\}$, $\{\text{U}, \text{X}\}$ or $\{\underline{\text{U}}, \text{X}\}$.

Finally, *path operators A and E are used to express that a temporal property has to hold for all computation paths or for at least one*. Actually, only one of the path quantifiers E and A is necessary, as the equations $\text{E}\varphi = \neg\text{A}\neg\varphi$ and $\text{A}\varphi = \neg\text{E}\neg\varphi$ hold. Nevertheless, we use both quantifiers in order to have a comfortable language. The translation from one of the mentioned temporal operator basis into another one holds also for CTL, e.g. consider the following equations:

$$\begin{aligned} \text{E}[x \text{ W } b] &= \text{E}[(\neg b) \underline{\text{U}} (x \wedge b)] \vee \text{EG}\neg b & \text{A}[x \text{ W } b] &= \neg\text{E}[(\neg b) \underline{\text{U}} (\neg x \wedge b)] \\ \text{E}[x \text{ U } b] &= \text{E}[x \underline{\text{U}} b] \vee \text{EG}x & \text{A}[x \text{ U } b] &= \neg\text{E}[(x \wedge \neg b) \underline{\text{U}} (\neg x \wedge \neg b)] \\ \text{E}[x \text{ B } b] &= \neg\text{A}[(\neg x) \underline{\text{U}} b] & \text{A}[x \text{ B } b] &= \neg\text{E}[(\neg x) \underline{\text{U}} b] \end{aligned}$$

As already outlined, the particular choice of the operator bases is almost irrelevant. However, it has to be noted here, that the W -operator has some properties that are useful for our translation procedure.

Definition 1 (Syntax of CTL*) *The following mutually recursive definitions introduce the set of path formulae $\mathcal{P}_\Sigma^*$ and the set of state formulae $\mathcal{S}_\Sigma^*$ over a given finite set of variables V_Σ :*

- *The set of path formulae $\mathcal{P}_\Sigma^*$ over the variables V_Σ is the smallest set which satisfies the following properties:*
 - *each state formula is a path formula, i.e. $\mathcal{S}_\Sigma^* \subseteq \mathcal{P}_\Sigma^*$*
 - *path formulae are closed with respect to boolean operations, i.e. $\neg\varphi, \varphi \wedge \psi, \varphi \vee \psi \in \mathcal{P}_\Sigma^*$ if $\varphi, \psi \in \mathcal{P}_\Sigma^*$*
 - *path formulae are closed with respect to temporal operators, i.e. $X\varphi, G\varphi, F\varphi, [\varphi W \psi] \in \mathcal{P}_\Sigma^*, [\varphi \underline{W} \psi] \in \mathcal{P}_\Sigma^*, [\varphi U \psi] \in \mathcal{P}_\Sigma^*$ and $[\varphi \underline{U} \psi] \in \mathcal{P}_\Sigma^*$ if $\varphi, \psi \in \mathcal{P}_\Sigma^*$*
- *The set of state formulae $\mathcal{S}_\Sigma^*$ over the variables V_Σ is the smallest set which satisfies the following properties:*
 - *each variable in V_Σ is a state formula, i.e. $V_\Sigma \subseteq \mathcal{S}_\Sigma^*$*
 - *state formulae are closed with respect to boolean operations, i.e. $\neg\varphi, \varphi \wedge \psi, \varphi \vee \psi \in \mathcal{S}_\Sigma^*$ if $\varphi, \psi \in \mathcal{S}_\Sigma^*$*
 - *$E\varphi, A\varphi \in \mathcal{S}_\Sigma^*$ for all $\varphi \in \mathcal{P}_\Sigma^*$*

The set of CTL formulae over the variables V_Σ is the set of state formulae $\mathcal{S}_\Sigma^*$ over V_Σ .*

Models of CTL* are given as so-called Kripke structures. However, there are also other variants, where computation trees or even arbitrary sets of paths are used for the definition of the semantics [Emer90]. While the computation tree semantics and the semantics based on Kripke structures are the same* for CTL*, some additional restrictions are necessary for the ‘set-of-path semantics’ [Emer90]. In the following, the Kripke structure semantics is used for the paper.

Definition 2 (Kripke Structures) *A Kripke structure $\mathcal{M} = (S, \mathcal{R}, \mathcal{L})$ for a set of variables V_Σ is given by a finite set of states S , a transition relation $\mathcal{R} \subseteq S \times S$ that must be total (i.e. $\forall s_0. \exists s_1. (s_0, s_1) \in \mathcal{R}$), and a labeling function $\mathcal{L} : S \rightarrow 2^{V_\Sigma}$ that maps each state to a set of variables. For functions $\pi : \mathbb{N} \rightarrow S$, the predicate $\mathcal{P}_\mathcal{M}(\pi, s) := (s = \pi^{(0)}) \wedge \forall i \in \mathbb{N}. (\pi^{(i)}, \pi^{(i+1)}) \in \mathcal{R}$ defines the set of paths starting in the state s through $\mathcal{M}$.*

*It is remarkable that these two semantics are no longer equivalent if we add quantification over variables.

$(s_0, s_1) \in \mathcal{R}$ means that there is a transition from state $s_0 \in \mathcal{S}$ to state $s_1 \in \mathcal{S}$. As $\mathcal{R}$ is total, each state has at least one successor state. In order to define the semantics of CTL* formulae, some manipulations on paths are necessary which are formulated here in a usual λ -calculus notation. A path π is defined as function that maps natural numbers to states of the Kripke structure. The k -th state of a path π is denoted by $\pi^{(k-1)}$ which is a function application of the function π to the natural number k . The suffix where the first δ states are cut off is denoted with $\lambda x. \pi^{(x+\delta)}$, i.e. a new path whose k -th state is $(\lambda x. \pi^{(x+\delta)})^{(k)} = \pi^{(k+\delta)}$.

Definition 3 (Semantics of CTL*) *Given a path π in a Kripke structure $\mathcal{M}$, the following rules define the semantics of CTL* path formulae:*

- $(\mathcal{M}, \pi) \models \varphi$ iff $(\mathcal{M}, \pi^{(0)}) \models \varphi$ for each state formula φ
- $(\mathcal{M}, \pi) \models \neg\varphi$ iff not $(\mathcal{M}, \pi) \models \varphi$
- $(\mathcal{M}, \pi) \models \varphi \wedge \psi$ iff $(\mathcal{M}, \pi) \models \varphi$ and $(\mathcal{M}, \pi) \models \psi$
- $(\mathcal{M}, \pi) \models \varphi \vee \psi$ iff $(\mathcal{M}, \pi) \models \varphi$ or $(\mathcal{M}, \pi) \models \psi$
- $(\mathcal{M}, \pi) \models X\varphi$ iff $(\mathcal{M}, \lambda x. \pi^{(x+1)}) \models \varphi$
- $(\mathcal{M}, \pi) \models G\varphi$ iff $(\mathcal{M}, \lambda x. \pi^{(x+i)}) \models \varphi$ holds for all $i \in \mathbb{N}$
- $(\mathcal{M}, \pi) \models F\varphi$ iff $(\mathcal{M}, \lambda x. \pi^{(x+i)}) \models \varphi$ holds for at least one $i \in \mathbb{N}$
- $(\mathcal{M}, \pi) \models [\varphi W \psi]$ iff either $(\mathcal{M}, \lambda x. \pi^{(x+\delta)}) \models \neg\psi$ for all $\delta \in \mathbb{N}$ or there is a $\delta \in \mathbb{N}$ such that $(\mathcal{M}, \lambda x. \pi^{(x+\delta)}) \models \psi \wedge \varphi$ holds and $(\mathcal{M}, \lambda x. \pi^{(x+t)}) \models \neg\psi$ holds for all $t < \delta$.
- $(\mathcal{M}, \pi) \models [\varphi \underline{W} \psi]$ iff there is a $\delta \in \mathbb{N}$ such that $(\mathcal{M}, \lambda x. \pi^{(x+\delta)}) \models \psi \wedge \varphi$ holds and $(\mathcal{M}, \lambda x. \pi^{(x+t)}) \models \neg\psi$ holds for all $t < \delta$.
- $(\mathcal{M}, \pi) \models [\varphi U \psi]$ iff either $(\mathcal{M}, \lambda x. \pi^{(x+\delta)}) \models \varphi \wedge \neg\psi$ for all $\delta \in \mathbb{N}$ or there is a $\delta \in \mathbb{N}$ such that $(\mathcal{M}, \lambda x. \pi^{(x+\delta)}) \models \psi$ holds and $(\mathcal{M}, \lambda x. \pi^{(x+t)}) \models \varphi \wedge \neg\psi$ holds for all $t < \delta$.
- $(\mathcal{M}, \pi) \models [\varphi \underline{U} \psi]$ iff there is a $\delta \in \mathbb{N}$ such that $(\mathcal{M}, \lambda x. \pi^{(x+\delta)}) \models \psi$ holds and $(\mathcal{M}, \lambda x. \pi^{(x+t)}) \models \varphi \wedge \neg\psi$ holds for all $t < \delta$.

For a given state s of a Kripke structure $\mathcal{M} = (\mathcal{S}, \mathcal{R}, \mathcal{L})$, the semantics of a state formula is given by the following definitions:

- $(\mathcal{M}, s) \models x$ iff $x \in \mathcal{L}(s)$
- $(\mathcal{M}, s) \models \neg\varphi$ iff not $(\mathcal{M}, s) \models \varphi$
- $(\mathcal{M}, s) \models \varphi \wedge \psi$ iff $(\mathcal{M}, s) \models \varphi$ and $(\mathcal{M}, s) \models \psi$
- $(\mathcal{M}, s) \models \varphi \vee \psi$ iff $(\mathcal{M}, s) \models \varphi$ or $(\mathcal{M}, s) \models \psi$
- $(\mathcal{M}, s) \models E\varphi$ iff there is a path π through $\mathcal{M}$ beginning in s such that $(\mathcal{M}, \pi) \models \varphi$ holds
- $(\mathcal{M}, s) \models A\varphi$ iff for all paths π through $\mathcal{M}$ beginning in s $(\mathcal{M}, \pi) \models \varphi$ holds

CTL* is a very powerful language, but has the disadvantage that the model checking problem, i.e. the problem of finding the set of states of a Kripke structure where a given CTL* formula holds, is PSPACE complete [Emer90]. Hence, a lot of sublan-

languages of CTL* have been defined [CIEm81, EmHa86, EmLe86, Emer90, BeGr94a] in order to find an appropriate language for specification and verification of finite state problems. ‘Appropriate’ means in this context to find a compromise between expressiveness and efficiency. In this paper, some more sublanguages are defined which will turn out to share the same expressiveness with CTL but are more succinct and more readable than CTL. A formal definition of these languages is given by the following definition in form of BNF grammars:

Definition 4 (Sublanguages of CTL*) *Given a finite set of variables V_Σ , the following grammars define the sublanguages LeftCTL*, LeftCTL⁺⁺ and CTL of CTL*:*

$$\begin{aligned}
 \text{LeftCTL}^* : \quad & S ::= V_\Sigma \mid \neg S \mid S \wedge S \mid S \vee S \mid EP_E \mid AP_A \\
 & P_E ::= S \mid \neg P_A \mid P_E \wedge P_E \mid P_E \vee P_E \mid XP_E \mid FP_E \\
 & \quad \mid [P_E \ W \ S] \mid [S \ U \ P_E] \mid [P_E \ B \ S] \\
 & \quad \mid [P_E \ \underline{W} \ S] \mid [S \ \underline{U} \ P_E] \\
 & P_A ::= S \mid \neg P_E \mid P_A \wedge P_A \mid P_A \vee P_A \mid XP_A \mid GP_A \\
 & \quad \mid [P_A \ W \ S] \mid [P_A \ U \ S] \mid [S \ B \ P_E] \\
 & \quad \mid [P_A \ \underline{W} \ S] \mid [P_A \ \underline{U} \ S] \\
 \text{LeftCTL}^{++} : \quad & S ::= V_\Sigma \mid \neg S \mid S \wedge S \mid S \vee S \mid EP_E \mid AP_A \\
 & P_E ::= S \mid XP_E \mid FP_E \\
 & \quad \mid [P_E \ W \ S] \mid [S \ U \ P_E] \mid [P_E \ B \ S] \\
 & \quad \mid [P_E \ \underline{W} \ S] \mid [S \ \underline{U} \ P_E] \\
 & P_A ::= S \mid XP_A \mid GP_A \\
 & \quad \mid [P_A \ W \ S] \mid [P_A \ U \ S] \mid [S \ B \ P_E] \\
 & \quad \mid [P_A \ \underline{W} \ S] \mid [P_A \ \underline{U} \ S] \\
 \text{CTL}^+ : \quad & S ::= V_\Sigma \mid \neg S \mid S \wedge S \mid S \vee S \mid EP \mid AP \\
 & P ::= \neg P \mid P \wedge P \mid P \vee P \mid XS \mid GS \mid FS \\
 & \quad \mid [S \ W \ S] \mid [S \ U \ S] \mid [S \ B \ S] \\
 & \quad \mid [S \ \underline{W} \ S] \mid [S \ \underline{U} \ S] \mid [S \ \underline{B} \ S] \\
 \text{CTL} : \quad & S ::= V_\Sigma \mid \neg S \mid S \wedge S \mid S \vee S \mid EP \mid AP \\
 & P ::= XS \mid GS \mid FS \\
 & \quad \mid [S \ W \ S] \mid [S \ U \ S] \mid [S \ B \ S] \\
 & \quad \mid [S \ \underline{W} \ S] \mid [S \ \underline{U} \ S] \mid [S \ \underline{B} \ S]
 \end{aligned}$$

The nonterminals S and P with or without indices describe sets of state or path formulae, respectively. LeftCTL* is the subset of CTL* where all events, i.e. all right hand arguments ψ in subformulae $[\varphi \ W \ \psi]$ are state formulae. LeftCTL⁺⁺ adds the restriction to LeftCTL* that on each path quantifier a temporal operator has to follow. In LeftCTL* and LeftCTL⁺⁺, we must distinguish between path formulae P_A and P_E , i.e. path formulae that can occur after the path quantifiers A and E, respectively. Finally, CTL is the subset of CTL*, where path quantifiers and temporal operators occur in pairs as defined in [CIEm81]. In [EmHa85], the language CTL⁺ has been defined as given above. CTL⁺ allows that path quantifiers can be applied on temporal operators as well as on boolean operators. Syntactically, we

Figure 1 Translation procedures between the sublanguages

have the following set inclusions: $CTL \subset LeftCTL^{++} \subset LeftCTL^* \subset CTL^*$ and $CTL \subset CTL^+ \subset LeftCTL^* \subset CTL^*$.

In [Schn96c] a subset of LTL has been investigated that can be translated in a syntax directed manner into deterministic ω -automata. The subset of LTL that can be translated by the method given in [Schn96c] without generating fairness constraints is also a subset of $LeftCTL^*$ and can hence also be translated directly to CTL. A lot of other sublanguages of CTL^* have been defined [EmHa86, Emer90] that form a hierarchy between CTL and CTL^* . These languages are however not considered in this paper as they are not related to the language $LeftCTL^*$. In order to translate a $LeftCTL^*$ formula into a CTL formula, two main problems have to be considered: there are two reasons why a $LeftCTL^*$ formula is not a CTL formula:

1. the formula contains a subformula $A\varphi$ or $E\varphi$ where φ does not begin with a temporal operator
2. the formula contains a subformula φ that begins with a temporal operator, but φ is not preceded by a path quantifier A or E

In the following, we will prove that the languages $LeftCTL^*$, $LeftCTL^{++}$, CTL^+ and CTL share the same expressiveness but differ in their succinctness. We will also give translation procedures from $LeftCTL^*$ to $LeftCTL^{++}$ and from $LeftCTL^{++}$ finally to CTL such that any CTL model checker can be used to check also $LeftCTL^{++}$ and $LeftCTL^*$ formulae. The translation procedure from $LeftCTL^{++}$ to CTL produces a CTL formula that has a size linear to the given input, and hence we prove that $LeftCTL^{++}$ has also a linear model checking procedure. The same does however not hold for the logics $LeftCTL^*$ and CTL^+ . These logics are more succinct than $LeftCTL^{++}$ and CTL, i.e. there are $LeftCTL^*$ and CTL^+ formulae that can only be expressed by more than exponentially longer CTL formulae. Hence, we loose the

linear runtime of the model checking procedures for the languages LeftCTL^* and CTL^+ .

3 REDUCTION FROM LeftCTL^* TO LeftCTL^{++}

In this section, we will prove that for each formula of LeftCTL^* there is an equivalent formula of LeftCTL^{++} . The principle we need for this proof is already well-known and has been used in [EmHa86] to prove that it is not a restriction on expressiveness when path quantifiers are restricted to be applied to temporal operators in CTL^* . Similar ideas have also been used to prove the expressive equivalence of CTL^+ and CTL in [EmHa85]. The key to the transformation of LeftCTL^* to LeftCTL^{++} is the following theorem:

Theorem 1 (Elimination of Conjunctions of Temporal Operators)

For all formulae $x_1, x_2, b_1, b_2 \in \mathcal{P}_\Sigma^$, the following equations are valid:*

● *Conjunctions of X:*

- $[Xx_1] \wedge [Xx_2] = X(x_1 \wedge x_2)$
- $[Xx_1] \wedge [Gx_2] = x_2 \wedge X(x_1 \wedge Gx_2)$
- $[Xx_1] \wedge [Fx_2] = (x_2 \wedge Xx_1) \vee X(x_1 \wedge Fx_2)$
- $[Xx_1] \wedge [x_2 \text{ W } b_2] = (b_2 \wedge x_2 \wedge Xx_1) \vee (\neg b_2 \wedge X(x_1 \wedge [x_2 \text{ W } b_2]))$
- $[Xx_1] \wedge [x_2 \text{ W } b_2] = (b_2 \wedge x_2 \wedge Xx_1) \vee (\neg b_2 \wedge X(x_1 \wedge [x_2 \text{ W } b_2]))$

● *Conjunctions of G:*

- $[Gx_1] \wedge [Xx_2] = x_1 \wedge X(x_2 \wedge Gx_1)$
- $[Gx_1] \wedge [Gx_2] = G(x_1 \wedge x_2)$
- $[Gx_1] \wedge [Fx_2] = [(Gx_1) \text{ W } (x_1 \rightarrow x_2)]$
- $[Gx_1] \wedge [x_2 \text{ W } b_2] = [(x_2 \wedge Gx_1) \text{ W } (x_1 \rightarrow b_2)]$
- $[Gx_1] \wedge [x_2 \text{ W } b_2] = [(x_2 \wedge Gx_1) \text{ W } (x_1 \rightarrow b_2)]$

● *Conjunctions of F:*

- $[Fx_1] \wedge [Xx_2] = (x_1 \wedge Xx_2) \vee X(x_2 \wedge Fx_1)$
- $[Fx_1] \wedge [Gx_2] = [(Gx_2) \text{ W } (x_2 \rightarrow x_1)]$
- $[Fx_1] \wedge [Fx_2] = F(x_1 \wedge Fx_2) \vee F(x_2 \wedge Fx_1)$
- $[Fb_1] \wedge [x_2 \text{ W } b_2] = [(b_1 \wedge [x_2 \text{ W } b_2]) \vee [x_2 \wedge b_2 \wedge Fx_1]] \text{ W } (b_1 \vee b_2)$
- $[Fb_1] \wedge [x_2 \text{ W } b_2] = [(b_1 \wedge [x_2 \text{ W } b_2]) \vee [x_2 \wedge b_2 \wedge Fx_1]] \text{ W } (b_1 \vee b_2)$

● *Conjunctions of W:*

- $[x_1 \text{ W } b_1] \wedge Xx_2 = (b_1 \Rightarrow (x_1 \wedge Xx_2)) \vee X(x_2 \wedge [x_1 \text{ W } b_1])$

$$\begin{aligned}
- [x_1 \text{ W } b_1] \wedge Gx_2 &= [(x_1 \wedge Gx_2) \text{ W } (x_2 \rightarrow b_1)] \\
- [x_1 \text{ W } b_1] \wedge Fb_2 &= [([b_2 \wedge [x_1 \text{ W } b_1]] \vee [x_1 \wedge b_1 \wedge Fx_2]) \text{ W } (b_1 \vee b_2)] \\
- [x_1 \text{ W } b_1] \wedge [x_2 \text{ W } b_2] &= \left(\begin{array}{l} [(x_1 \wedge b_1 \wedge [x_2 \text{ W } b_2]) \text{ W } (b_1 \vee b_2)] \vee \\ [(x_2 \wedge b_2 \wedge [x_1 \text{ W } b_1]) \text{ W } (b_1 \vee b_2)] \end{array} \right) \\
- [x_1 \text{ W } b_1] \wedge [x_2 \text{ W } b_2] &= \left(\begin{array}{l} [(x_1 \wedge b_1 \wedge [x_2 \text{ W } b_2]) \text{ W } (b_1 \vee b_2)] \vee \\ [(x_2 \wedge b_2 \wedge [x_1 \text{ W } b_1]) \text{ W } (b_1 \vee b_2)] \end{array} \right)
\end{aligned}$$

● *Conjunctions of $\underline{\text{W}}$:*

$$\begin{aligned}
- [x_1 \underline{\text{W}} b_1] \wedge Xx_2 &= (b_1 \Rightarrow (x_1 \wedge Xx_2) | X(x_2 \wedge [x_1 \underline{\text{W}} b_1])) \\
- [x_1 \underline{\text{W}} b_1] \wedge Gx_2 &= [(x_1 \wedge Gx_2) \underline{\text{W}} (x_2 \rightarrow b_1)] \\
- [x_1 \underline{\text{W}} b_1] \wedge Fb_2 &= [([b_2 \wedge [x_1 \underline{\text{W}} b_1]] \vee [x_1 \wedge b_1 \wedge Fx_2]) \underline{\text{W}} (b_1 \vee b_2)] \\
- [x_1 \underline{\text{W}} b_1] \wedge [x_2 \text{ W } b_2] &= \left(\begin{array}{l} [(x_1 \wedge b_1 \wedge [x_2 \text{ W } b_2]) \underline{\text{W}} (b_1 \vee b_2)] \vee \\ [(x_2 \wedge b_2 \wedge [x_1 \underline{\text{W}} b_1]) \underline{\text{W}} (b_1 \vee b_2)] \end{array} \right) \\
- [x_1 \underline{\text{W}} b_1] \wedge [x_2 \underline{\text{W}} b_2] &= \left(\begin{array}{l} [(x_1 \wedge b_1 \wedge [x_2 \underline{\text{W}} b_2]) \underline{\text{W}} (b_1 \vee b_2)] \vee \\ [(x_2 \wedge b_2 \wedge [x_1 \underline{\text{W}} b_1]) \underline{\text{W}} (b_1 \vee b_2)] \end{array} \right)
\end{aligned}$$

Note also that the following special cases hold (these hold also for $\underline{\text{W}}$):

- $[x_1 \text{ W } b] \wedge [x_2 \text{ W } b] = [(x_1 \wedge x_2) \text{ W } b]$
- $[x_1 \text{ W } b] \vee [x_2 \text{ W } b] = [(x_1 \vee x_2) \text{ W } b]$
- $[x_1 \text{ W } b] \rightarrow [x_2 \text{ W } b] = [(x_1 \rightarrow x_2) \text{ W } b]$
- $([x_1 \text{ W } b] = [x_2 \text{ W } b]) = ((x_1 = x_2) \text{ W } b)$

Note that by negating both sides of the equations and using deMorgan's theorem and the equations $\neg Gx = F(\neg x)$, $\neg Fx = G(\neg x)$, $\neg[x \text{ W } b] = [(\neg x) \underline{\text{W}} b]$, and $\neg[x \underline{\text{W}} b] = [(\neg x) \text{ W } b]$, we obtain analog equations for transforming disjunctions of temporal operators into conjunctions of temporal operators. Applying these equations repeatedly allows to transform a conjunction of formulae of the forms Xx , Gx , Fx , $[x \text{ W } b]$, $[x \underline{\text{W}} b]$ into either a disjunction or a simple formula of one of these forms.

Theorem 2 (Temporal Disjunctive and Conjunctive Normal Form)

Any arbitrary CTL* formula can be transformed in both of the following normal forms, where α_i is a CTL* state formula:

- $\bigvee_{i=1}^n \alpha_i \wedge [X\beta_i] \wedge [G\gamma_i] \wedge [F\delta_i] \wedge [\rho_i \text{ W } \pi_i] \wedge [\sigma_i \underline{\text{W}} \xi_i]$
- $\bigwedge_{i=1}^n \alpha_i \vee [X\beta_i] \vee [G\gamma_i] \vee [F\delta_i] \vee [\rho_i \text{ W } \pi_i] \vee [\sigma_i \underline{\text{W}} \xi_i]$

Proof. Each CTL* formula can be transformed to an equivalent CTL* formula in disjunctive normal form $\bigvee_{i=1}^n \bigwedge_{j=1}^{m_i} \Phi_{i,j}$ where each formula $\Phi_{i,j}$ is either a state

formula, or is of one of the following forms: Xx , Gx , Fx , $[x \text{ W } b]$, $[x \text{ W } b]$. The equivalences in theorem 1 immediately reduce this form to the first one specified in the theorem by collecting all X -, G -, F -, W and W formulae. The second form is just the dual of the first one and is obtained for a given formula φ by first computing the first normal form for $\neg\varphi$ and negating this afterwards. ■

Now, assume a formula $E\varphi \in \text{LeftCTL}^*$ is to be translated to LeftCTL^+ . We first compute the temporal disjunctive normal form and then shift the path quantifier E over the top-level disjunctions and over the state formula α_i according to the following lemma:

Lemma 1 *Given an arbitrary state formula $\psi \in \Sigma^+$ and an arbitrary path formula $\varphi \in \Sigma^+$, the following equation hold:*

$$\begin{array}{ll} \models E\psi = \psi & \models A\psi = \psi \\ \models E(\psi \wedge \varphi) = \psi \wedge E\varphi & \models A(\psi \wedge \varphi) = \psi \wedge A\varphi \\ \models E(\psi \vee \varphi) = \psi \vee E\varphi & \models A(\psi \vee \varphi) = \psi \vee A\varphi \end{array}$$

Hence, it remains now to consider $E([X\beta_i] \wedge [G\gamma_i] \wedge [F\delta_i] \wedge [\rho_i \text{ W } \pi_i] \wedge [\sigma_i \text{ W } \xi_i])$. Theorem 1 is now used to replace the top-level conjunction that follows the path quantifier E either by a formula starting with a disjunction of temporal operators or a single temporal operator. The first case is reduced to the latter one by shifting E once more over the top-level disjunction. If the same procedure is now applied to all subformulae starting with a path quantifier, finally all path quantifiers are shifted towards temporal operators. Hence, we have proved the following result:

Theorem 3 (Expressiveness and Succinctness of LeftCTL^*)

For any arbitrary LeftCTL^ formula φ there is a LeftCTL^{++} formula ψ such that $\models \varphi = \psi$.*

Considering succinctness of the logics, we add the result for CTL^+ of [EmHa85]. For the logic LeftCTL^* , things are even worse because the effect that blows up a CTL^+ formula during the translation can be repeated in LeftCTL^* .

Theorem 4 (Expressiveness and Succinctness of CTL^+)

For each CTL^+ formula φ there is a CTL formula ψ such that $\models \varphi = \psi$ and $\|\psi\| \in O(\|\varphi\|!) = O(2^{\|\varphi\| \log(\|\varphi\|)})$.

In order to see, where the blow-up occurs, consider the following CTL^+ formula which is also a LeftCTL^* formula: $E \bigwedge_{j=1}^n Fx_j$. The result of the above translation procedure is in this case the following: define for an arbitrary permutation π of the numbers $1, \dots, n$ the formulae $\xi_{\pi,1} := EFx_{\pi(1)}$ and $\xi_{\pi,j+1} := EF(x_{\pi(j+1)} \wedge \xi_{\pi,j})$. The result of the procedure is then $\bigvee_{\pi \in P_n} \xi_{\pi,n}$, where P_n is the set of all permutations of the numbers $1, \dots, n$.

4 REDUCTION FROM LeftCTL⁺⁺ TO CTL

In LeftCTL⁺⁺ each path quantifier precedes a temporal operator, but not vice versa. Hence, in order to translate a LeftCTL⁺⁺ formula into an equivalent CTL formula, path quantifiers have to be added to those temporal operators which are not already preceded by a path quantifier. The following theorem allows this and is the background for the definition of the languages LeftCTL⁺⁺ and LeftCTL*:

Theorem 5 (Adding Path Quantifiers)

For all path formulae $\varphi \in \mathcal{P}_\Sigma^$ and all state formulae $\psi \in \mathcal{S}_\Sigma^*$, the following equations hold:*

(1) $\models \text{EX}\varphi = \text{EXE}\varphi$	(2) $\models \text{AX}\varphi = \text{AXA}\varphi$
(3) $\models \text{EF}\varphi = \text{EFE}\varphi$	(4) $\models \text{AG}\varphi = \text{AGA}\varphi$
(5) $\models \text{E}[\varphi \text{ W } \psi] = \text{E}[(\text{E}\varphi) \text{ W } \psi]$	(6) $\models \text{A}[\varphi \text{ W } \psi] = \text{A}[(\text{A}\varphi) \text{ W } \psi]$
(7) $\models \text{E}[\varphi \text{ W } \psi] = \text{E}[(\text{E}\varphi) \text{ W } \psi]$	(8) $\models \text{A}[\varphi \text{ W } \psi] = \text{A}[(\text{A}\varphi) \text{ W } \psi]$
(9) $\models \text{E}[\psi \text{ U } \varphi] = \text{E}[\psi \text{ U } (\text{E}\varphi)]$	(10) $\models \text{A}[\varphi \text{ U } \psi] = \text{A}[(\text{A}\varphi) \text{ U } \psi]$
(11) $\models \text{E}[\psi \text{ U } \varphi] = \text{E}[\psi \text{ U } (\text{E}\varphi)]$	(12) $\models \text{A}[\varphi \text{ U } \psi] = \text{A}[(\text{A}\varphi) \text{ U } \psi]$
(13) $\models \text{E}[\varphi \text{ B } \psi] = \text{E}[(\text{E}\varphi) \text{ B } \psi]$	(14) $\models \text{A}[\psi \text{ B } \varphi] = \text{A}[\psi \text{ B } (\text{E}\varphi)]$

The equations (10) and (12) do also hold if ψ is a path formula.

In case of binary temporal operators, the preceding path quantifier can only be shifted to one of the arguments. This must generally hold, since otherwise we could transform every CTL* formula into an equivalent CTL formula, which is however not possible since CTL* is more expressive than CTL.

The language LeftCTL⁺⁺ is defined in such a way that the arguments of the binary operators where no path quantifier can be shifted to are already state formulae. Hence, rewriting formulae of LeftCTL⁺⁺ with the equations of theorem 5 yields in pure CTL formulae of roughly the same length.

Theorem 6 (Reduction of LeftCTL⁺⁺ to CTL)

For any arbitrary LeftCTL⁺⁺ state formula φ there is a CTL formula ψ such that $\models \varphi = \psi$ and $\|\psi\| \in O(\|\varphi\|)$.

Proof. The proof is done by a structural induction along the algorithm in figure 3. As the given formula is in the language LeftCTL⁺⁺, the cases $\Phi \equiv \neg\varphi$, $\Phi \equiv \varphi \wedge \psi$ and $\Phi \equiv \varphi \vee \psi$ cannot occur in the functions. The induction steps can then be simply reduced to the equations of theorem 5. ■

Theorem 7 (Reduction of LeftCTL* to CTL)

For any arbitrary LeftCTL state formula φ there is a CTL formula ψ such that $\models \varphi = \psi$.*

Proof. Again the proof is done by structural induction along the structure of the formula by regarding the algorithm in figure 3. The function $E_conj(\Phi)$ computes the conjunctive temporal normal form of Φ and then applies the equations of theorem 1 on each clause and shifts the path quantifier inwards to the temporal operators. A_disj is the dual function that computes the disjunctive temporal normal form of Φ and shifts the path quantifier A inwards. In all other cases, only path quantifiers are added according to theorem 5. Obviously, the procedure terminates after a finite number of steps with a CTL formula. ■

It is important to see that the condition that the formula ψ has to be a state formula in theorem 5 is necessary. In particular, the following lemma holds:

Figure 2 A Kripke structure for disproving some equations.

Lemma 2 *In general, the following equations are not valid:*

(1) $\not\models E[\varphi \text{ W } \psi] = E[(E\varphi) \text{ W } \psi]$	(2) $\not\models A[\varphi \text{ W } \psi] = A[(A\varphi) \text{ W } \psi]$
(3) $\not\models E[\varphi \text{ W } \underline{\text{W}} \psi] = E[(E\varphi) \text{ W } \underline{\text{W}} \psi]$	(4) $\not\models A[\varphi \text{ W } \underline{\text{W}} \psi] = A[(A\varphi) \text{ W } \underline{\text{W}} \psi]$
(5) $\not\models E[\varphi \text{ U } \psi] = E[\varphi \text{ U } (E\psi)]$	(6) $\not\models E[\varphi \text{ U } \underline{\text{U}} \psi] = E[\varphi \text{ U } \underline{\text{U}} (E\psi)]$
(7) $\not\models EG\varphi = EGE\varphi$	(8) $\not\models AF\varphi = AFA\varphi$

Proof. Consider state s_0 in the Kripke structure $\mathcal{M}$ in figure 2. There are exactly two paths starting in state s_0 . It is easy to see that on the one hand $(\mathcal{M}, s_0) \not\models E[(F\bar{x}) \text{ W } (Gb)]$ holds, but on the other hand we have $(\mathcal{M}, s_0) \models E[(EF\bar{x}) \text{ W } (Gb)]$ since $(\mathcal{M}, s_0) \models EF\bar{x}$ holds. This is due to the fact that the E path quantifier inside the W -expression allows to choose the right path in state s_0 while the entire formula holds on the left hand path of s_0 . Hence, equation (1) is disproved. We can disprove (3) by exchanging W by $\underline{W}$ in the previous argumentation. (2) and (4) can be reduced to (3) and (1), respectively by the identity $\neg[\varphi \text{ W } \psi] = [(\neg\varphi) \text{ W } \underline{\text{W}} \psi]$. Now, consider state s_4 . Again there are two paths starting in state s_4 . It can be proved that $(\mathcal{M}, s_4) \not\models E[(F\bar{x}) \text{ U } (Gb)]$, but on the other hand $(\mathcal{M}, s_4) \models E[(F\bar{x}) \text{ U } (EGb)]$. The latter holds since we can choose in state s_5 the left alternative to validate $(\mathcal{M}, s_5) \models$

Figure 3 Algorithm for translating LeftCTL* to CTL

EGb, while the right alternative is chosen to satisfy the entire formula. Hence, equation (6) is disproved and the same argumentation disproves also (5). Finally, if (7) would hold, then we could reduce $EGFx$ to $EGEFx$. However, this is a CTL formula, which contradicts the fact that no CTL formula is equivalent to $EGFx$ [EmHa85]. (8) can be reduced to (7) by pushing negation symbols inwards and outwards. ■

5 FURTHER EXTENSIONS OF LeftCTL*

In each case, where the definition of LeftCTL* requires a state formula instead of a (more general) path formula, it can be shown that there is an example, which could no longer be expressed in CTL. Nevertheless, in some special cases more nestings of temporal operators can be allowed without extending the expressiveness of the logic. *This means that there are still CTL* formulae which are equivalent to some LeftCTL* formula, but which do not follow the syntax of LeftCTL**. In this section, we consider some further extensions of LeftCTL*.

For example, consider the simple formula $AFXx$. It is certainly not a LeftCTL* formula, but it can be reduced by the equality $FXx = XFx$ to $AXFx$. The latter is a LeftCTL* formula and is translated by our algorithm to the CTL formula $AXAFx$. As the path quantifiers can be always added to formulae beginning with a X-operator, it is reasonable to shift X-operators outside as shown in the following theorem:

Theorem 8 (Homomorphism of X)

For all CTL path formulae, the following equations hold:*

$$\begin{array}{ll}
 \neg Xx = X\neg x & (Xy) \vee (Xx) = X(x \vee y) \\
 (Xy) \wedge (Xx) = X(x \wedge y) & FXx = XFx \\
 GXx = XGx & [(Xx) \text{ W } (Xy)] = X[x \text{ W } y] \\
 [(Xx) \text{ U } (Xy)] = X[x \text{ U } y] & [(Xx) \text{ U } (Xy)] = X[x \text{ U } y] \\
 [(Xx) \text{ B } (Xy)] = X[x \text{ B } y] & [(Xx) \text{ B } (Xy)] = X[x \text{ B } y]
 \end{array}$$

However, if only one argument of a binary operator has a leading X-operator, as e.g. in $[(Xy) \text{ W } x]$, these laws cannot be used directly. In this case, a new variable q is introduced and x is substituted by Xq in the given formula such that $[(Xy) \text{ W } (Xq)]$ is obtained. Of course, the transition equation $G(Xq = x)$ has to be added to the Kripke structure of the given model checking problem. After that, the X-operators can be shifted outwards. For example, the model checking problem $\mathcal{M} \models X[b \text{ W } Xa] \wedge c$ is transformed into the following:

$$\left[\begin{array}{l} \mathcal{M} \\ G(Xq_1 = b) \\ G(Xq_2 = c) \\ G(Xq_3 = q_2) \end{array} \right] \models XX([q_1 \text{ W } a] \wedge q_3)$$

For the translation of $AFXx$, it is sufficient to shift the X operator outwards, i.e. to change the ordering of the nesting of the temporal operators. In other cases, it is even possible (and for our translation necessary), to eliminate the nesting of temporal operators. These elimination laws can be used to extend the definition of the language LeftCTL^* in such a manner that in each case, where the grammar rules for LeftCTL^* in definition 4 expected a state formula, we can now allow some special forms of path formulae. For example, the elimination theorem $G[x \text{ U } b] = G(x \vee b)$ leads to the new grammar rule $S ::= AG[P_A \text{ U } P_A]$. A direct application of the elimination theorem reduces this rule to $S ::= AGP_A \vee P_A$, and this can be derived from the already existing rules* in contrast to $AG[P_A \text{ U } P_A]$.

Similar theorems have been used in [BeGr94a] to allow at most one nesting of temporal operators to obtain the language CTL^2 . These theorems can also be used to extend the definition of LeftCTL^* . Some of them are listed below:

Theorem 9 (Elimination of Temporal Operator Nestings) *For all CTL^* path formulae, the following equations hold:*

● *Nestings of G and F:*

$$\begin{array}{ll} GXx = XGx & FXx = XFx \\ GGx = Gx & FFx = Fx \\ G[x \text{ U } b] = G(x \vee b) & \\ G[x \text{ W } b] = G(b \rightarrow x) & F[x \text{ W } b] = F(x \wedge b) \vee FG\neg b \end{array}$$

● *Nestings of U:*

$$\begin{array}{l} - [x \text{ U } (b_1 \vee b_2)] = [x \text{ U } b_1] \vee [x \text{ U } b_2] \\ - [(x_1 \wedge x_2) \text{ U } b] = [x_1 \text{ U } b] \wedge [x_2 \text{ U } b] \\ - [x \text{ U } (Xb)] = (Xb) \vee [x \text{ U } (x \wedge b)] \\ - [(Xx) \text{ U } b] = b \vee X[x \text{ U } (x \wedge b)] \\ - [(Gx) \text{ U } b] = b \vee Gx \\ - [x \text{ U } (Fb)] = [Fb] \vee [Gx] \\ - [([x \text{ U } a]) \text{ U } b] = b \vee [(x \vee a) \text{ U } b] \wedge ([a \text{ W } (Xb)] \vee [[x \text{ U } a] \text{ W } b]) \\ - [([x \text{ W } a]) \text{ U } b] = b \vee [(x \vee \neg a) \text{ U } b] \wedge ([a \text{ W } (Xb)] \vee [[x \text{ W } a] \text{ W } b]) \end{array}$$

Some of the above equations have been used in [BeGr94a]. While in the above theorem only one nesting of temporal operators is considered, one could also consider arbitrary deep nestings to extend the language. For example, at the next level of nestings the following equations can be added: $G(x \vee [y \text{ U } b]) = G(x \vee y \vee b)$, $G(x \vee [y \text{ W } b]) = G(x \vee y \vee \neg b)$, $G(x \vee Gy) = [(Gy) \text{ W } (\neg x)], \dots$

* $S ::= AP_A ::= AGP_A$

6 EXAMPLE: THE SINGLE PULSER

In [JoMC94] the single pulser has been investigated as an example circuit that is hard to specify in some logic languages such as CTL. The circuit has a boolean valued input in and a boolean valued output o . It is required that between two rising edges of in there is exactly one point of time, where o is high. Using linear temporal logics, this can be expressed by the following specifications:

1. $AG[\neg in \wedge X in \rightarrow X([o B (\neg in \wedge X in)] \vee [o W (\neg in \wedge X in)])]$
2. $AG \left[o \wedge \neg(\neg in \wedge X in) \rightarrow \left(\begin{array}{l} X[(\neg o) \cup (\neg in \wedge X in)] \wedge \\ X[(\neg o) W (\neg in \wedge X in)] \end{array} \right) \right]$
3. $AG[o \rightarrow X \neg o]$

Specification 1 states that when a rising edge is detected at the input in , then the output o must be high before another rising edge at in is detected. It is also allowed that o can be high when the next rising edge at in is detected. If a second rising edge does not occur, then this first specification does trivially hold. The second specification states, that when o is high and no rising edge follows at in , then from the next point of time on, o must be low at least until the next rising edge occurs on in and also at the point of time when the next rising edge occurs. Finally, the third specification states that after o is high it is definitely low at the next point of time. While 1. states the existence of an output, 2. and 3. assure that the output is in each case a single pulse. 3. is not subsumed by 2., as the first two specs allow the behavior given in figure 4, where o is not a single pulse.

Figure 4 Not a Single Pulsers behavior.

For the translation of the specifications, it is convenient to introduce new variables ℓ_{in} , ℓ_{in}^2 , ℓ_o and ℓ_o^2 which follow the transition equations $G[X\ell_{in} = in]$, $G[X\ell_{in}^2 = \ell_{in}]$, $G[X\ell_o = o]$ and $G[X\ell_o^2 = \ell_o]$. As explained in the last section, these equations allow the substitution occurrences of in and o , such that all X operators can be shifted outwards. This yields in the following modified specifications:

1. $AXXG[\neg \ell_{in}^2 \wedge \ell_{in} \rightarrow ([\ell_o B (\neg \ell_{in} \wedge in)] \vee [\ell_o W (\neg \ell_{in} \wedge in)])]$
2. $AXXG \left[\ell_o^2 \wedge \neg(\neg \ell_{in}^2 \wedge \ell_{in}) \rightarrow \left(\begin{array}{l} [(\neg \ell_o) \cup (\neg \ell_{in} \wedge in)] \wedge \\ [(\neg \ell_o) W (\neg \ell_{in} \wedge in)] \end{array} \right) \right]$
3. $AXG[\ell_o \rightarrow \neg o]$

The translation of these formulae does only require to shift the path quantifier inwards.

REFERENCES

- [BCDM86] M.C. Browne, E.M. Clarke, D.L. Dill, and B. Mishra. Automatic Verification of Sequential Circuits Using Temporal Logic. *IEEE Transactions on Computers*, C-35(12):1034–1044, December 1986.
- [BCMD92] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. *Information and Computing*, 98(2):142–170, June 1992.
- [BeGr94a] O. Bernholtz and O. Grumberg. Buy one, get one free!!! In *First International Conference on Temporal Logic*, 1994.
- [Brya86] R.E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.
- [CGHJ93a] E.M. Clarke, O. Grumberg, H. Hiraishi, S. Jha, D.E. Long, K.L. McMillan, and L.A. Ness. Verification of the Futurebus+ Cache Coherence Protocol. In D. Agnew, L. Claesen, and R. Camposano, editors, *The Eleventh International Symposium on Computer Hardware Description Languages and their Applications*, pages 5–20, Ottawa, Canada, April 1993. IFIP WG10.2, CHDL'93, IEEE COMPSOC, Elsevier Science Publishers B.V., Amsterdam, Netherland.
- [CIEm81] E.M. Clarke and E.A. Emerson. Design and Synthesis of Synchronization Skeletons using Branching Time Temporal Logic. In D. Kozen, editor, *Proceedings of the Workshop on Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71, Yorktown Heights, New York, May 1981. Springer-Verlag.
- [CIGH94a] E.M. Clarke, O. Grumberg, and K. Hamaguchi. Another look at LTL model checking. In David L. Dill, editor, *Proceedings of the sixth International Conference on Computer-Aided Verification CAV*, volume 818 of *Lecture Notes in Computer Science*, pages 415–427, Stanford, California, USA, June 1994. Springer-Verlag.
- [DiCl86] D.L. Dill and E.M. Clarke. Automatic verification of asynchronous circuits using temporal logic. *IEE Proceedings*, 133 Part E(5):276–282, September 1986.
- [Emer90] E.A. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 996–1072, Amsterdam, 1990. Elsevier Science Publishers.
- [EmHa85] E.A. Emerson and J.Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *Journal of Computer and System Science*, 30:1–24, 1985.
- [EmHa86] E.A. Emerson and J.Y. Halpern. "sometimes" and "not never" revisited: On branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, January 1986.
- [EmLe85] E.A. Emerson and C.-L. Lei. Modalities for model checking: Branching time strikes back. In *Proceedings of the Twelfth Annual ACM*

- Symposium on Principles of Programming Languages*, pages 84–96, New York, January 1985. ACM.
- [EmLe86] E.A. Emerson and C.-L. Lei. Temporal reasoning under generalized fairness constraints. In B.Monien and G.Vidal-Naquet, editors, *3rd Annual Symposium on Theoretical Aspects of Computer Science*, pages 21–36, Orsay, France, January 1986. Springer-Verlag.
- [JoMC94] S.D. Johnson, P.S. Miner, and A. Camilleri. Studies of the single pulser in various reasoning systems. In T. Kropf and R. Kumar, editors, *Proc. 2nd International Conference on Theorem Provers in Circuit Design (TPCD94)*, volume 901 of *Lecture Notes in Computer Science*, pages 126–145, Bad Herrenalb, Germany, September 1994. Springer-Verlag. published 1995.
- [Jong91a] G.G de Jong. An automata theoretic approach to temporal logic. In K.G. Larsen and A. Skou, editors, *Proceedings of 3rd Workshop on Computer Aided Verification (CAV91)*, volume 575 of *Lecture Notes in Computer Science*, pages 477–487, Aalborg, July 1991. Springer-Verlag.
- [Lamp80] L. Lamport. "sometime" is sometimes "not never"-on the temporal logic of programs. In *Proceedings of the Seventh ACM Symposium on Principles of Programming Languages*, pages 174–185, New York, 1980. ACM.
- [LiPn85] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proceedings of the Twelfth Annual ACM Symposium on Principles of Programming Languages*, pages 97–107, New York, January 1985. ACM.
- [McMi93a] K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Norwell Massachusetts, 1993.
- [Pnue77] A. Pnueli. The temporal logic of programs. In *Proceedings of the Eighth Annual Symposium on Foundations of Computer Science*, volume 18, pages 46–57, New York, 1977. IEEE.
- [Schn96c] K. Schneider. Translating LTL Model Checking to CTL Model Checking. Technical Report SFB358-C2-3/96, Universität Karlsruhe, Institut für Rechnerentwurf und Fehlertoleranz, January 1996. <http://goethe.ira.uka.de/hvg/techreports/SFB358-C2-3-96.ps.gz>.
- [SiCl85] A.P. Sistla and E.M. Clarke. The complexity of propositional linear temporal logics. *Journal of Assoc. Comput. Mach.*, 32(3):733–749, July 1985.
- [Wolp87] P. Wolper. On the relation of programs and computations to models of temporal logic. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Temporal Logic in Specification*, pages 75–123, Altrincham, UK, 1987. Springer-Verlag.