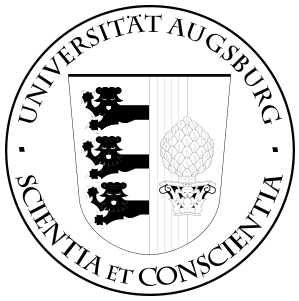
Universität Augsburg



Embedding CTL* in an Extension to Interval Temporal Logic (ITL)

F. Ortmeier, M. Balser, A. Dunets, S. Bäumler

Report 2008-16 2008

Institut für Informatik
D-86135 Augsburg

Embedding CTL* in an Extension to Interval Temporal Logic (ITL)

Frank Ortmeier, Michael Balser, Andriy Dunets, and Simon Bäumler

Lehrstuhl für Softwaretechnik und Programmiersprachen,
Universität Augsburg, D-86135 Augsburg
{ortmeier, balser, dunets}@informatik.uni-augsburg.de

Abstract. In this paper we present an embedding of the most common branching time logics (CTL/CTL*) in an extension of interval temporal logic (ITL$^+$). The significance of this result is threefold: first the theoretical aspect is, that branching time and linear time are not so much different. A more practical aspect is that the intuitive interactive proof method of symbolic execution of ITL$^+$ can be used for branching time logics as well. The opposite direction is interesting as well, for a subset of finite state systems, interactive verification of ITL$^+$ formulas can be translated into a model checking problem.
The proof presented in this paper has been done with the interactive theorem prover KIV. So this contribution can also be seen as a case study on reasoning about temporal logics in an interactive verification environment.

Key words: temporal logics, CTL, ITL$^+$, interactive verification, model checking

1 Introduction

One important domain of application of formal verification techniques is the analysis of safety critical systems. During the last years several well-known techniques of safety analysis have been formalized and have been applied to numerous domains like railways, avionics, space systems or automation control. A lot of these techniques base their formal semantics on Computational Tree Logic (CTL) and use state charts for system description. In most cases the system models are finite state abstractions of the real world. This allows to use model checking techniques and get automatic verification results. This is a great help for many problems.

On the other hand finite state abstraction (as any other kind of abstraction does) often raises the question of soundness of an abstraction. This becomes most clearly visible, if continuous variables of the physical world are abstracted

to discrete values. Consider for example an autonomous railroad crossing system. The idea of this system is that the train and the crossing autonomously control the lowering and raising of the bars (of the crossing) without the supervision of a central control in the station. An obvious hazard is, that the train passes the crossing, while the bars are open. A trivial formalization would be:

$$Train.Pos = Crossing.Pos \wedge \neg Crossing.Bars = Open$$

Closer examination shows, that the formalization of this property is adequate only, if the position (and speed) of the train has not been discretized. If a finite state abstraction (i.e. the position of the train is represented by integer values) is used as system model, the hazard must be formalized as follows:

$$Train.Pos \leq Crossing.Pos \wedge (Train.Pos + Train.Speed) > Crossing.Pos \wedge$$

$$\wedge \neg Crossing.Bars = Open$$

An elaborate description and analysis of this example with formal methods may be found in [10]. But even this short glance shows the problems that finite state abstractions can bring. At the same time model checking is a great help in industrial praxis and should be possible as well. Because of this it will be very useful if the proof obligations, which have been proven correct for finite state systems can be proven correct for a refined infinite state system. So in short words: We would like to apply model checking if possible and interactive verification if necessary.

For an efficient combination switching between formalism must be easy with respect to transforming a) the system model and b) the proof obligation. If this can be achieved, then this combined strategy gives the best tradeoff between effort, time and significance of the analysis.

For interactive verification, our method of choice is symbolic execution. Symbolic execution is a well known approach for the verification of sequential programs [5, 6] and gives intuitive proofs which can be automated to a large extent. In [1], we have proposed a calculus to also apply symbolic execution to verify temporal properties of concurrent systems. The properties are expressed in interval temporal logic (ITL), systems can be described using parallel programs or state charts [11]. A implementation of the calculus for ITL$^+$ is part of the KIV system [6]. For model checking, we use SMV [8] as we have obtained the best results in terms of system size with this model checker. For this model checker temporal properties are expressed in concurrent tree logic (CTL) and system specifications are written as finite automata. State charts with Statemate semantics [2] can be directly used as ITL$^+$ specifications for the interactive theorem prover KIV [12]. This – and the in this paper presented theorem to translate CTL into ITL$^+$ and vice versa – allows to efficiently combine model checking and interactive verification.

Assume you want to prove an ITL$^+$ property φ for a state chart system SC. In the practice of interactive verification many attempts fail because φ does not describe the intended property or because the system SC has not

been modeled correctly. The central theorem of this paper allows to evaluate a translation $\tilde{\varphi}$ (in CTL) of an ITL$^+$-formula φ for a finite abstraction $\tilde{SC}$ of the system SC by model checking. The abstraction may in many case be found very easily, because many state chart models can be translated canonically into finite automata models (like those known from SMV [8] for example). It is then only a "push-the-button" effort to evaluate different $\tilde{\varphi}$. This helps a lot for finding the intended property φ, which can then be verified for SC.

The opposite approach benefits as well. If a finite state system SYS has been model checked for a CTL property φ. Then this system can be directly described by a state chart model SC (if the system is given in finite automata language – like i.e. SMV input language – only syntactical conversion are necessary). The model SC can then be extended to more adequately describe reality. In particular SC may become infinite state. The property φ may then be verified interactively for the infinite state system by translating φ into ITL$^+$. This approach may be used to either verify correctness of an abstraction or to increase significance of the verification. It is our experience, that both directions are frequently used in the practice of verification.

Our goal is a combination of interactive verification with symbolic execution and model checking. Model checking is most efficient for CTL which is a branching time logic. On the other hand, symbolic execution is currently only applicable to a linear time logic (in our case ITL), and there seems to be a fundamental problem to apply symbolic execution for CTL in general. However, we have been able to enrich ITL with two operators

$$\langle\varphi\rangle\ \psi \quad \text{and} \quad [\varphi]\ \psi$$

in order to express branching time problems. The first operator states that there is a path satisfying φ which also satisfies ψ. The second operator states that all paths satisfying φ also satisfy ψ. These operators are inspired by dynamic logic (DL) and we have been able to define a calculus for symbolic execution of these operators. We here refer to the extended logic as ITL$^+$. An overview of the logic and the calculus may be found in [1].

There is, however, a subtle – but significant – difference between these operators and the branching time operators $\mathbf{E}\ \varphi$ and $\mathbf{A}\ \varphi$ of CTL. In this paper, we focus on this difference and show how to translate model based CTL properties to ITL$^+$. As a consequence, we will be able to apply symbolic execution to the interactive verification of CTL properties. This novel approach is – in our opinion – by far easier and more flexible than the use of CTL tableau calculi such as [7, 13]. Thus, the theoretical result of this paper is of practical significance.

All presented proofs have been done in the interactive verification environment KIV. for this we had to build an algebraic specification of i) the logic ITL$^+$, the logic CTL, Kripke-structures and linear-time structures. As a result we could prove, that our embedding is correct with respect to the semantics of CTL and ITL$^+$. So the here presented work can also be seen as an interesting case study on reasoning about temporal logics and equivalence between logics in general. An HTML-export of the proofs can be downloaded from the web-site of the authors.

In Sect. 2 and 3 we give brief introductions to the logics CTL and ITL$^+$. Section 4 describes how to translate CTL properties to ITL$^+$, while in Sect. 5 an outline of the proof is presented and our experiences with KIV are shown in Sect. 6. Sect. 7 comprises some future work and limitations. The results of this paper are summarized in Section 8.

2 CTL*

We will briefly define syntax and semantics of CTL*. These definitions are standard in computer science. In particular we use the notations of [3] and [4].

Kripke structure

Let AP be a finite set of atomic propositions. We define the semantics of CTL* on a Kripke structure $\mathcal{K}$. A Kripke structure $\mathcal{K} = (S, R, L)$ is a 3-tuple of a set of states S, a total transition relation $R \subseteq S \times S$ and a labeling function $L : S \to 2^{AP}$ which labels states with atomic propositions. We also define the infinite paths of a Kripke structure: $p = (s_0, s_1, ...) \in \mathcal{K}$, where $\forall i \geq 0 : (s_i, s_{i+1}) \in R$ and $|p| = \infty$. We write $p[i]$ to denote the ith state s_i of path p and p_i for the suffix-path starting with the ith state.

Syntax

To define the syntax of CTL* it is convenient to introduce the notion of *state formulas* Φ and *path formulas* θ. We define these two inductively with the following BNF grammar.

$$\Phi ::= \alpha \mid \Phi \wedge \Phi \mid \neg \Phi \mid \mathbf{A}\ \theta \quad \text{with } \alpha \in AP$$
$$\theta ::= \Phi \mid \theta \wedge \theta \mid \neg\ \theta \mid \theta\ \mathbf{U}\ \theta \mid \mathbf{X}\ \theta$$

The set of all state formulas forms the logic CTL*. Other operators are derived like **G** (globally), **F** (eventually) or **E** (there exists a path). The logic CTL is a true subset of CTL*. In CTL, every temporal operator (**U**, **X**) must be combined with a path operator (**A** or **E**).

Semantics

Definition 1. *(Semantics of CTL*) Given a Kripke structure $\mathcal{K}$, a state $s \in S$ and a path $p \in \mathcal{K}$. A state formula Φ evaluates to $\mathcal{K}, s \models_C \Phi$ and a path formulas θ evaluates to $\mathcal{K}, p \models_C \theta$ according to the definitions of Table 1.*

3 ITL$^+$

ITL$^+$ is an extension of interval temporal logic (ITL) [9]. This logic and a corresponding proof calculus is implemented in the interactive theorem prover KIV.

atom	$\mathcal{K}, s \models_C \alpha$	iff	$\alpha \in L(s)$
neg-state	$\mathcal{K}, s \models_C \neg\, \Phi$	iff	$\mathcal{K}, s \not\models_C \Phi$
con-state	$\mathcal{K}, s \models_C \Phi_1 \wedge\, \Phi_2$	iff	$\mathcal{K}, s \models_C \Phi_1$ and $\mathcal{K}, s \models_C \Phi_2$
allpath	$\mathcal{K}, s \models_C \mathbf{A}\ \theta$	iff	forall $p = (s, s_1, ...) \in \mathcal{K}:\ \mathcal{K}, p \models_C \theta$
pathstate	$\mathcal{K}, p \models_C \Phi$	iff	$\mathcal{K}, p[0] \models_C \Phi$
neg-path	$\mathcal{K}, p \models_C \neg\, \theta$	iff	$\mathcal{K}, p \not\models_C \theta$
con-path	$\mathcal{K}, p \models_C \theta_1 \wedge\, \theta_2$	iff	$\mathcal{K}, p \models_C \theta_1$ and $\mathcal{K}, p \models_C \theta_2$
until	$\mathcal{K}, p \models_C \theta_1\ \mathbf{U}\ \theta_2$	iff	exists $i:\ \mathcal{K}, p_i \models_C \theta_2$ and forall $j < i:\ \mathcal{K}, p_j \models_C \theta_1$
next	$\mathcal{K}, p \models_C \mathbf{X}\ \theta$	iff	$\mathcal{K}, p_1 \models_C \theta$

Table 1. Semantics of CTL*

As an important addition to ITL, ITL$^+$ also contains modal operators $\langle\varphi\rangle\ \psi$ and $[\varphi]\ \psi$ which are similar to the quantifiers of CTL*. In this paper, we present only a subset of the logic. For the full definition of ITL$^+$ including quantification of dynamic variables, parallel programs with interleaving and nondeterministic choice, we refer to [1].

Intervals

In ITL$^+$, temporal formulas are evaluated on linear sequences of states – often called traces. States σ are defined as valuations of variables of an underlying predicate logic PL. $\sigma(var)$ selects the value of variable var in state σ. Similar to Interval Temporal Logic, we explicitly consider finite and infinite traces and therefore also refer to a sequence of states as an interval.

A selection of additional functions concerning intervals are helpful in the following. Consider a finite or infinite interval $I := (\sigma_0, \ldots)$. Then $I[i] := \sigma_i$ selects the ith state. $I|_i := (\sigma_i, \ldots)$ is a *postfix*, $I|^i := (\sigma_0, \ldots, \sigma_i)$ a *prefix*, and $I|_i^j := (\sigma_i, \ldots, \sigma_j)$ an *infix* of I.

Syntax

Let φ_{PL} be a first order formula. Formulas φ of ITL$^+$ are defined by the following BNF grammar:

$$\varphi ::= \varphi_{\mathrm{PL}} \mid \neg\, \varphi \mid \varphi \rightarrow \varphi \mid \varphi; \varphi \mid \textbf{step} \mid \varphi\ \textbf{until}\ \varphi \mid [\varphi]\ \varphi$$

Semantics

Definition 2. *(Semantics of ITL$^+$) Given an interval I, a formula φ evaluates to $I \models_I \varphi$. The semantics of the operators are defined as in Table 2.*

A formula φ is called valid *(abbreviated by $\models_I \varphi$), if $I \models_I \varphi$ for all I.*

The chop operator $\varphi; \psi$ is a special operator of Interval Temporal Logics. It corresponds to sequential composition of programs and is the key to semantically treat programs (and also state charts) as temporal formulas as proposed in [1, 12].

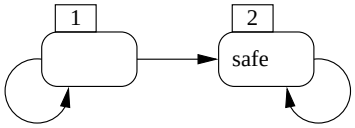
atom	$I \models_I \varphi_{\mathrm{PL}}$	iff	$I[0] \models_{PL} \varphi_{\mathrm{PL}}$
neg	$I \models_I \neg\, \varphi$	iff	$I \not\models_I \varphi$
con	$I \models_I \varphi \wedge \psi$	iff	$I \models_I \varphi$ and $I \models_I \psi$
chop	$I \models_I \varphi;\, \psi$	iff	there exists $n \leq \lvert I \rvert$ with $I\rvert^n \models_I \varphi$ and $I\rvert_n \models_I \psi$
			or $\lvert I \rvert = \infty$ and $I \models_I \varphi$
step	$I \models_I \mathbf{step}$	iff	$\lvert I \rvert = 1$
until	$I \models_I \varphi\ \mathbf{until}\ \psi$	iff	there exists $n \leq \lvert I \rvert$
			with $I\rvert_n \models_I \psi$
			and $I\rvert_m \models_I \varphi$ forall $0 \leq m < n$
box	$I \models_I [\varphi]\ \psi$	iff	forall I_0 with $I_0[0] = I[0]$
			holds $I_0 \models_I \varphi \Rightarrow I_0 \models_I \psi$

Table 2. Semantics of a selection of ITL$^+$ operators

Most of the LTL operators can be derived in ITL$^+$ with the exception of **until**, which is defined as a basic operator. Further LTL operators are defined as abbreviations. The eventually operator $\Diamond\ \varphi :\equiv \text{true}\ \mathbf{until}\ \varphi$, the always operator $\Box\ \varphi :\equiv \neg \Diamond \neg\, \varphi$. Because intervals can also be finite, the next operator comes in two flavors: the strong next $\circ\ \varphi :\equiv \text{step};\, \varphi$ requires that there is a next step satisfying φ, the weak next $\bullet\ \varphi :\equiv \neg \circ \neg\, \varphi$ only states that if there is a next step, it must satisfy φ.

The modal operators [.] . and $\langle . \rangle$. can be used to quantify intervals; while $[\varphi]\ \psi$ states that all intervals satisfying φ also satisfy ψ, the dual operator $\langle \varphi \rangle\ \psi :\equiv \neg\, [\varphi] \neg\, \psi$ requires that there exists an interval satisfying both formulas.

Example 1. ITL$^+$ modal operators
Consider a trivial nondeterministic state transition system depicted on the left hand side of Figure 1. This system can be canonically described by the ITL$^+$ formula ψ on the right hand side of Figure 1.



$$\psi \equiv \Box \begin{pmatrix} \begin{pmatrix} state = 1 \wedge \circ\ state = 1 \\ \vee\ state = 1 \wedge \circ\ state = 2 \\ \vee\ state = 2 \wedge \circ\ state = 2 \end{pmatrix} \\ \wedge\ (state = 1 \rightarrow \neg\ safe) \\ \wedge\ (state = 2 \rightarrow safe) \end{pmatrix}$$

Fig. 1. Example state transition system

A typical universal verification problem is:

$$\models_I [\psi] \circ \Box\ safe$$

This property reads: "Every execution of system ψ satisfies that starting with the next state, *safe* is always satisfied." This property does not hold for the given

system. However the existential property

$$\models_I \langle\psi\rangle \circ \Box\ safe$$

does hold. Because there always exists a path, such that starting with the next state always *safe* is satisfied. Note, that in this example no initial state is defined. So the formula is true only if it is true for all intervals and thus for all initial states.

4 Embedding CTL* in ITL$^+$

In order to define an embedding of CTL* into ITL$^+$, we must achieve the following: (i) the model which is used to evaluate CTL* formulas must be translated into an equivalent ITL$^+$ formula and (ii) the temporal CTL* operators must be replaced by equivalent ITL$^+$ operators and finally (iii) CTL* path quantifiers (**A** and **E**) must be expressed by the means of ITL$^+$ modal operators.

4.1 Modeling the Kripke structure

We first construct an ITL$^+$ formula $\psi_\mathcal{K}$ to represent a Kripke structure $\mathcal{K} = (S, R, L)$. We call the set of atomic propositions AP, which are used in the labeling function L. We define one (dynamic) variable *state* for the set of states S.

$$dom(state) = S$$

For every atomic proposition $\alpha \in AP$ we define a (dynamic) boolean variable v_α. This allows to encode a Kripke structure $\mathcal{K} = (S, R, L)$ in the following ITL$^+$ formula $\psi_\mathcal{K}$:

$$\psi_\mathcal{K} :\equiv \Box \left(\begin{array}{l} \bigvee\limits_{(s,t)\in R} state = s \wedge \circ\ state = t \\ \wedge \bigwedge\limits_{s\in S} (state = s \rightarrow \bigwedge\limits_{\alpha\in L(s)} v_\alpha = \text{true} \wedge \bigwedge\limits_{\alpha\notin L(s)} v_\alpha = \text{false}) \end{array} \right)$$

The first line of the above formula assures, that traces of $\psi_\mathcal{K}$ are only made of transitions in R. The second line preserves the labeling (of function L) in the ITL$^+$ formula.

Only for this paper we construct an ITL$^+$-formula for a Kripke structure directly. In practical applications, models are not described by Kripke structures explicitly, but rather as finite automata (e.g. the SMV specification language). These specifications can canonically be modeled by state charts (only syntactical conversions are necessary), which are part of the ITL$^+$ logic [12]. So in practical applications it is easy to construct formula $\psi_\mathcal{K}$ for a given SMV specification.

4.2 Embedding CTL*

We now define a function ctl2itl which takes a CTL* formula φ and a system description $\psi_{\mathcal{K}}$ as inputs and generates an equivalent ITL$^+$ formula. Atomic propositions are identified with the corresponding boolean variable:

$$\text{ctl2itl}(\alpha, \psi_{\mathcal{K}}) :\equiv v_\alpha = \text{true}, \quad \text{for all } \alpha \in AP$$

Boolean connectives are canonically transferred:

$$\begin{aligned} \text{ctl2itl}(\theta_1 \wedge\ \theta_2, \psi_{\mathcal{K}}) &:\equiv \text{ctl2itl}(\theta_1, \psi_{\mathcal{K}}) \wedge\ \text{ctl2itl}(\theta_2, \psi_{\mathcal{K}}) \\ \text{ctl2itl}(\neg\ \theta, \psi_{\mathcal{K}}) &:\equiv \neg\ \text{ctl2itl}(\theta, \psi_{\mathcal{K}}) \end{aligned}$$

Temporal operators are also easy to embed. The basic CTL* temporal operator **X** and **U** are replaced with ITL$^+$ operators $\circ$ and **until**.

$$\begin{aligned} \text{ctl2itl}(\mathbf{X}\ \theta, \psi_{\mathcal{K}}) &:\equiv \circ\ \text{ctl2itl}(\theta, \psi_{\mathcal{K}}) \\ \text{ctl2itl}(\theta_1\ \mathbf{U}\ \theta_2, \psi_{\mathcal{K}}) &:\equiv \text{ctl2itl}(\theta_1, \psi_{\mathcal{K}})\ \textbf{until}\ \text{ctl2itl}(\theta_2, \psi_{\mathcal{K}}) \end{aligned}$$

Path quantifiers are more difficult to embed. In CTL* the system is only implicitly referred. It is not part of a sequent. As ITL$^+$ is a linear time logic, there is no system outside the sequent which we can use to define path quantifiers. To solve this problem, we use ITL$^+$ system operators and the characterization of the Kripke structure $\mathcal{K}$ as an ITL$^+$ formula $\psi_{\mathcal{K}}$.

$$\text{ctl2itl}(\mathbf{A}\ \theta, \psi_{\mathcal{K}}) :\equiv [\psi_{\mathcal{K}}]\ \text{ctl2itl}(\theta, \psi_{\mathcal{K}})$$

5 Soundness of embedding

In this section, we only present the basic idea and excerpts of the proof. The formal specification and proofs are described in Section 6.

The theorem presented here states that every CTL* formula Φ can be expressed in ITL$^+$ as validity problem.

$$\mathcal{K}, s \models_C \Phi \ \Leftrightarrow\ \models_I \mathit{state} = s \rightarrow \text{ctl2itl}(\Phi, \psi_{\mathcal{K}}) \tag{1}$$

However it is easier to prove the more general statement that not only state formulas but all path formulas can be embedded in ITL$^+$. This is justified, because every state formula is also a path formula. The soundness of embedding path formulas θ in ITL$^+$ is formalized as a predicate, called "embeddable". This property is useful to structure the inductive proof of Theorem 1.

Definition 3. *(Embeddability) Given a Kripke structure $\mathcal{K}$ and a corresponding ITL$^+$ system (description) formula $\psi_{\mathcal{K}}$. A CTL* (path) formula θ is* embeddable *in ITL$^+$ (denoted as* embeddable$(\theta)_{\mathcal{K},\psi_{\mathcal{K}}}$*), if and only if for all paths $p \in \mathcal{K}$*

$$\mathcal{K}, p \models_C \theta \Leftrightarrow \mathit{forall}\ I \in \text{trans}(p, \mathcal{K}) : I \models_I \text{ctl2itl}(\theta, \psi_{\mathcal{K}})$$

This property states, that a Kripke structure $\mathcal{K}$ and a CTL-path p satisfy a formula θ if and only if the translated formula $\text{ctl2itl}(\theta, \psi_{\mathcal{K}})$ is satisfied for all ITL-intervals I which are isomorphic to p. The set of isomorphic intervals is defined by function trans.

$$I \in \text{trans}(p, \mathcal{K}) :\Leftrightarrow (1)\ \forall\ i \in \mathbb{N}_0 : I[i](\mathit{state}) = p[i]$$
$$(2)\ \forall\ i \in \mathbb{N}_0 : \forall\ \alpha \in AP : I[i](v_\alpha) = \text{true} \Leftrightarrow \alpha \in L(p[i])$$

Statement 1 follows from the more general statement of Definition 3. The proof idea is as follows:

$$\begin{array}{ccc} \mathcal{K}, s \models_C \Phi & \Leftrightarrow & \models_I \mathit{state} = s \rightarrow \text{ctl2itl}(\Phi, \psi_{\mathcal{K}}) \\ \Updownarrow \text{(A)} & & \Updownarrow \text{(C)} \\ \mathcal{K}, p \models_C \theta & \overset{\text{(B)}}{\Leftrightarrow} & \text{forall } I \in \text{trans}(p, \mathcal{K}) : I \models_I \text{ctl2itl}(\theta, \psi_{\mathcal{K}}) \end{array}$$

In the following, a proof for the most interesting equivalence (B) is presented. This proof contains the structural induction on the basic operators of both logics. The other two equivalences (A) and (C) can be shown with simple meta logic reasoning.

5.1 Useful lemmas

This section contains some important properties, which will be used later. First of all three properties of trans are listed:

Lemma 1. *(Properties of* trans*)*

1. *if* $I \models_I \psi_{\mathcal{K}}$ *then* $\exists\ p \in \mathcal{K} : I \in \text{trans}(p, \mathcal{K})$
2. *if* $p \in \mathcal{K}$ *and* $I \in \text{trans}(p, \mathcal{K})$ *then* $I \models_I \psi_{\mathcal{K}}$
3. *if* $I \in \text{trans}(p, \mathcal{K})$ *and* $I' \in \text{trans}(p', \mathcal{K})$ *then*

$$p[i] = p'[i] \Leftrightarrow I[i] =_{\text{free}(\psi_{\mathcal{K}})} I'[i]$$

Proof. (Lemma 1.1) Let I be arbitrary with $I \models_I \psi_{\mathcal{K}}$. We have to show that $\exists\ p \in \mathcal{K} : I \in \text{trans}(p, \mathcal{K})$ holds. For this, we construct $p = (s_0, s_1, \ldots)$ such that $I[i](\mathit{state}) = p[i]$. Now we have to show that $p \in \mathcal{K}$ and $I \in \text{trans}(p, \mathcal{K})$.

$$\begin{array}{lll} & I \models_I & \psi_{\mathcal{K}} \\ \Leftrightarrow & I \models_I & \Box(\bigvee_{(s,t)\in R} (\mathit{state} = s \wedge \circ\ \mathit{state} = t) \wedge\ \ldots) \\ \Rightarrow & \forall\ i \in \mathbb{N}_0 : (I[i](\mathit{state}), I[i+1](\mathit{state})) \in R & \\ \Leftrightarrow & \forall\ i \in \mathbb{N}_0 : (p[i], p[i+1]) \in R & \\ \Leftrightarrow & p \in \mathcal{K}. & \end{array}$$

$I \in \text{trans}(p, \mathcal{K})$ as (1) of the definition of trans holds by construction of p and (2) follows directly from $I \models_I \psi_{\mathcal{K}}$. Since I was arbitrary the proof is complete. □

Proof. (Lemma 1.2) Let p, I be arbitrary with $p \in \mathcal{K}$ and $I \in \text{trans}(p, \mathcal{K})$. We have to show that $I \models_I \psi_\mathcal{K}$ holds. The proof is similar to the proof for Lemma 1.1. □

Proof. (Lemma 1.3) Let $p, p', I, I', i \in \mathbb{N}_0$ be arbitrary with $I \in \text{trans}(p, \mathcal{K})$ and $I' \in \text{trans}(p', \mathcal{K})$.

"⇒": $p[i] = p'[i] \Rightarrow I[i] =_{\text{free}(\psi_\mathcal{K})} I'[i]$

Evidently $\text{free}(\psi_\mathcal{K}) = \{state\} \cup \{v_\alpha \mid \alpha \in AP\}$.

$$\begin{aligned} & I \in \text{trans}(p, \mathcal{K}) \text{ and } I' \in \text{trans}(p', \mathcal{K}) \\ \Rightarrow \quad & I[i](state) = p[i] \text{ and } I'[i](state) = p'[i] \\ & \text{and } (I[i](v_\alpha) = \text{true} \Leftrightarrow \alpha \in L(p[i])) \\ & \text{and } (I'[i](v_\alpha) = \text{true} \Leftrightarrow \alpha \in L(p'[i])) \\ \Rightarrow \quad & I[i](state) = I'[i](state) \text{ and } I[i](v_\alpha) = I'[i](v_\alpha) \end{aligned}$$

"⇐": $p[i] = p'[i] \Leftarrow I[i] =_{\text{free}(\psi_\mathcal{K})} I'[i]$

$$\begin{aligned} & I[i] =_{\text{free}(\psi_\mathcal{K})} I'[i] \\ \Rightarrow \quad & I[i](state) = I'[i](state) \\ \Rightarrow \quad & p[i] = p'[i] \end{aligned}$$

□

An important lemma of ITL$^+$ is the coincidence lemma. This lemma states that the evaluation of a formula only depends on the valuation of its free variables.

Lemma 2. *(Coincidence) Given an ITL$^+$ formula φ. For arbitrary intervals I_1, I_2 with $|I_1| = |I_2|$, if*

$$\forall\ i \in 0..|I_1| : \forall\ v \in \text{free}(\varphi) : I_1[i](v) = I_2[i](v)$$

then

$$I_1 \models_I \varphi \Leftrightarrow I_2 \models_I \varphi$$

The proof of this lemma is too large to present here.

5.2 Central theorem

The following theorem states, that every CTL* path formula θ (for a Kripke structure $\mathcal{K}$) can be embedded into ITL$^+$.

Theorem 1. *For all Kripke-structures $\mathcal{K}$ and all CTL* path formulas θ,*

$$\text{embeddable}(\theta)_{\mathcal{K}, \psi_\mathcal{K}}$$

holds (for property embeddable *see Definition 3).*

This theorem shows, that CTL* formulas may be verified using the ITL$^+$ calculus. It also states, that certain ITL$^+$ formulas (i.e. those formulas which can be described as translations of CTL formulas) can be verified with CTL verification methods. This result is of high practical significance. It allows to use model checking techniques for finite state abstractions of interactive verification problems.

In the following we will give a draft of the proof. The basic concept is to use structural induction over the formula θ. The induction step is to show that every CTL*-constructor is *embeddable*:

$$\text{embeddable}(\theta)_{\mathcal{K},\psi_\mathcal{K}} \Rightarrow \text{embeddable}(\text{ctlOperator}(\theta))_{\mathcal{K},\psi_\mathcal{K}}$$

for all basic CTL*-operators. We only present some interesting parts of the proof, namely the operators $\wedge$ (and), **U** (until) and **A** (allpath).

5.3 Embeddability of A

With the induction hypothesis

$$\text{embeddable}(\theta)_{\mathcal{K},\psi_\mathcal{K}}$$

show

$$\text{embeddable}(\mathbf{A}\ \theta)_{\mathcal{K},\psi_\mathcal{K}}$$

$$\begin{aligned}
& \text{embeddable}(\mathbf{A}\ \theta)_{\mathcal{K},\psi_\mathcal{K}} \\
\Leftrightarrow\ & \text{(Def. embeddable)} \\
& \mathcal{K}, p \models_C \mathbf{A}\ \theta \Leftrightarrow \forall\ I : I \in \text{trans}(p,\mathcal{K}) : I \models_I \text{ctl2itl}(\mathbf{A}\ \theta, \psi_\mathcal{K}) \\
\Leftrightarrow\ & \text{(Sem. } \mathbf{A}\text{, Def. ctl2itl)} \\
& (\forall\ p' = (p[0], s_1, s_2, \ldots) \in \mathcal{K} : \mathcal{K}, p' \models_C \theta) \Leftrightarrow \\
& \qquad \forall\ I \in \text{trans}(p,\mathcal{K}) : I \models_I [\psi_\mathcal{K}]\ \text{ctl2itl}(\theta, \psi_\mathcal{K}) \\
\Leftrightarrow\ & \text{(Sem. [.] .)} \\
& (\forall\ p' : p' = (p[0], s_1, s_2, \ldots) \in \mathcal{K} : \mathcal{K}, p' \models_C \theta) \Leftrightarrow^* \\
& \quad \forall\ I \in \text{trans}(p,\mathcal{K}) : \forall\ I' : I'[0] = I[0] \text{ and } I' \models_I \psi_\mathcal{K} \Rightarrow \\
& \qquad I \models_I \text{ctl2itl}(\theta, \psi_\mathcal{K})
\end{aligned}$$

We separately show the two directions of $\Leftrightarrow^*$.

"$\Rightarrow$": Let I and I' be arbitrary with $I \in \text{trans}(p,\mathcal{K})$, $I'[0] = I[0]$, and $I' \models_I \psi_\mathcal{K}$. We have to show that $I \models_I \text{ctl2itl}(\theta, \psi_\mathcal{K})$.

$I' \models_I \psi_\mathcal{K}$ implies $\exists\ p' \in \mathcal{K} : I' \in \text{trans}(p',\mathcal{K})$ (because of Lemma 1.1). With $I' \in \text{trans}(p',\mathcal{K})$ and $I'[0] = I[0]$ follows $p[0] = p'[0]$ (Lemma 1.3). Because of the universal quantifier in the precondition $(\forall\ p' : p' = (p[0], \ldots)$ follows $\mathcal{K}, p' \models_C \theta$.

Because of the induction hypothesis, formula θ is embeddable. Since $I' \in \text{trans}(p',\mathcal{K})$ holds, $I' \models_I \text{ctl2itl}(\theta, \psi_\mathcal{K})$ is fulfilled. Since I and I' were arbitrary this direction of the proof is complete.

"$\Leftarrow$": Let p' be an arbitrary CTL-path with $p' = (p[0], s_1, s_2, \ldots) \in \mathcal{K}$. We have to show that $\mathcal{K}, p' \models_C \theta$ holds. Take an ITL-interval $I \in \text{trans}(p, \mathcal{K})$. Let I' be an arbitrary ITL-interval with $I' \in \text{trans}(p', \mathcal{K})$. Such intervals exist by definition of trans for all p and all $\mathcal{K}$.

Because of Lemma 1.2, $I' \models_I \psi_\mathcal{K}$. Take I'' with $I'' = (I[0], I'[1], I'[2], \ldots)$. Instantiate $\forall\ I$ and $\forall\ I'$ in the precondition with I and I'' respectively to receive

$$I \in \text{trans}(p, \mathcal{K}) \text{ and } I''[0] = I[0] \text{ and } I'' \models_I \psi_\mathcal{K} \Rightarrow I'' \models_I \text{ctl2itl}(\theta, \psi_\mathcal{K}) \ .$$

$I \in \text{trans}(p, \mathcal{K})$ and $I''[0] = I[0]$ are apparently true.

$I \in \text{trans}(p, \mathcal{K}), I' \in \text{trans}(p', \mathcal{K})$ and $p[0] = p'[0]$ with Lemma 1.3 implies $I'[0] =_{\text{free}(\psi_\mathcal{K})} I[0]$. This implies $I''[0] =_{\text{free}(\psi_\mathcal{K})} I'[0]$ (transitivity of "="). For all other states, I'' and I' are by construction equal. Together this means: $I'' =_{\text{free}(\psi_\mathcal{K})} I'$. With the coincidence lemma $I'' \models_I \psi_\mathcal{K}$ follows, because $I' \models_I \psi_\mathcal{K}$.

Consequently, $I'' \models_I \text{ctl2itl}(\theta, \psi_\mathcal{K})$. The definition of $\psi_\mathcal{K}$ apparently implies that $\text{free}(\theta) \subseteq \text{free}(\psi_\mathcal{K})$ and therefore $\text{free}(\text{ctl2itl}(\theta, \psi_\mathcal{K})) \subseteq \text{free}(\psi_\mathcal{K})$. In that case we know that

$$I' =_{\text{free}(\text{ctl2itl}(\theta, \psi_\mathcal{K}))} I''$$

because $I' =_{\text{free}(\psi_\mathcal{K})} I''$. The coincidence lemma is applied a second time to receive $I' \models_I \text{ctl2itl}(\theta, \psi_\mathcal{K})$.

The induction hypothesis $\text{embeddable}(\theta)_{\mathcal{K}, \psi_\mathcal{K}}$ states that

$$\mathcal{K}, p' \models_C \theta \Leftrightarrow \forall\ I' \in \text{trans}(p', \mathcal{K}) : I' \models_I \text{ctl2itl}(\theta, \psi_\mathcal{K}) \ .$$

The right part of the equivalence is true since $I' \in \text{trans}(p', \mathcal{K})$ was arbitrary and $I' \models_I \text{ctl2itl}(\theta, \psi_\mathcal{K})$ holds. Thus, $\mathcal{K}, p' \models_C \theta$. Since $p' = (p[0], s_1, s_2, ..)$ was arbitrary the proof is complete. □

6 Formal specification and proof in KIV

All of the proofs were constructed with the interactive theorem prover KIV. For this purpose, both logics were formalized as structured algebraic specifications using higher order logic. The complete specification contains over 100 modules. During this project the soundness of the ITL$^+$ calculus has also been verified.

Formal specifications in KIV are represented as directed acyclic graphs called *development graphs* where nodes correspond to specification components and edges form the specification structure. A single specification component specifies a signature, a generation principle and a set of axiom. The signature defines *sorts, constants*, *functions*, *predicates* and *variables*. The generation principle restricts models to term-generated models and provides structural induction. The semantics is loose, i.e., the set of all term-generated algebras that satisfy axioms.

For example, in Fig. 2, we specify the semantics of the CTL operator $\mathbf{A}\ \varphi$ (compare to Tab. 1) and define the translation to ITL$^+$. In Figure 3, we introduce

```
sem-allpath:      (sys × I) × val ⊨ AllPath φ
              ↔ ∀ I0. I0 ∈ sys∧ I0[0] = I[0] → (sys × I0) × val ⊨ φ;
ctl2itl-allpath: ctl2itl(AllPath φ, ψ) = box(_true, ψ _→ ctl2itl(φ, ψ))
```

Fig. 2. Formal specification of CTL operator **A** φ and its translation into ITL$^+$.

```
enrich next with
  functions box: xpr × xpr → xpr;
  ...
  axioms
  ...
  sem-box:      (sys × I) × val ⊨ box(φ, ψ)
            ↔ ∀ sys0, I0, val0.      val0[I0[0]] = val[I[0]]∧ I0 ∈ sys0
                                   ∧ (sys0 × I0) × val0 ⊨ φ
                                  → (sys0 × I0) × val0 ⊨ ψ;
  ...
  dia: dia(φ, ψ) = _¬ box(φ, _¬ ψ);
end enrich
```

Fig. 3. Formal specification of ITL$^+$ operators $[\varphi]\ \psi$ and $\langle\varphi\rangle\ \psi$.

the modal operators *box* and *diamond* of ITL$^+$ (compare to Table 2). Note that the ITL$^+$ formula ψ is evaluated by $(sys \times I) \times val \models \psi$. The triple on the left represents the model, where the *sys* component is important only for branching time operators. It would be sufficient to evaluate ITL$^+$ formulas over the interval represented by the tuple $I \times val$. If we reason about ITL$^+$ formulas ψ the *sys* component in the triple can be automatically eliminated and we need not bother about it. Also note that we separate the transition relation I on states σ from the valuation *val* of variables in a state.

Figure 4 shows the formalization of the *embeddable* property. Informally, this property describes the semantic equivalence of the CTL* formula φ and its translation $ctl2itl(\varphi, \psi)$ into ITL$^+$. The important assumption on ITL$^+$ formula

```
axioms
embeddable:
   embeddable(φ)
↔ ∀ ψ, sys, I, val.
          is-fo-itl+(ψ) ∧ injective(val)
        ∧ (∀ I0, val0.
               I0 × val0 ⊨ ψ
             ↔ (∃ I1. I1 ∈ sys∧ int2vint(I0, val0) = int2vint(I1, val)))
        ∧ (is-foctl*(φ) ∨ is-path-formula-foctl*(φ))
      → ((sys × I) × val ⊨ φ ↔ (sys × I) × val ⊨ ctl2itl(φ, ψ))
```

Fig. 4. Formalization of *embeddable*.

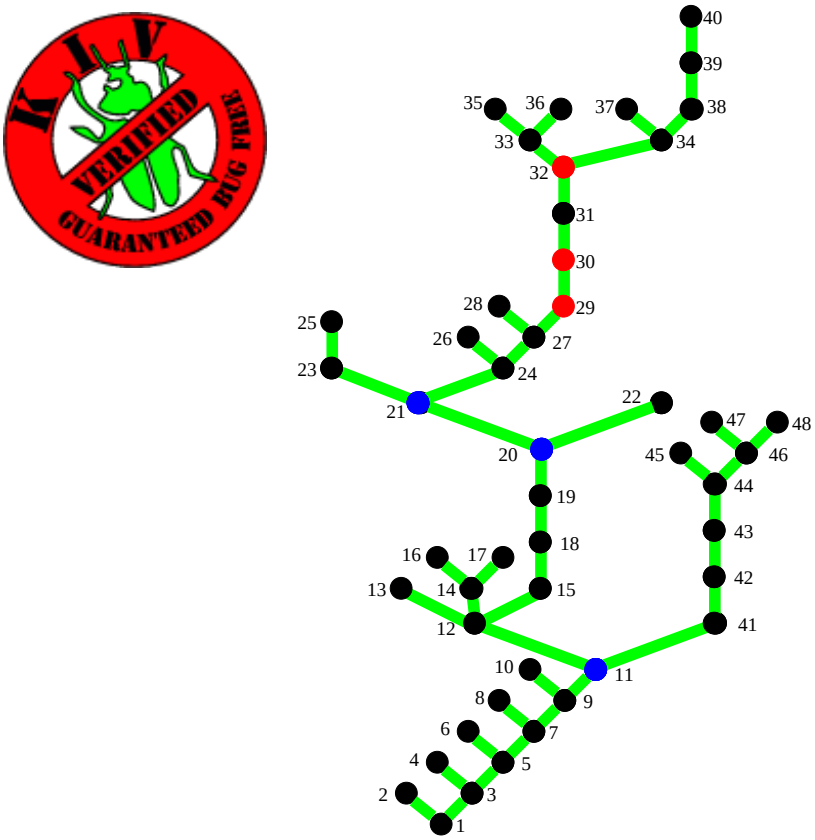


Fig. 5. Formal proof tree for the theorem *embeddable-allpath*.

ψ is its semantic equivalence to the model represented by the Kripke structure. Also, some further assumption $injective(val)$ on valuation val is needed. It states that equal valuation of states implies equality of states.

We also formalized the central theorem which states that all CTL formula can be embedded in ITL$^+$. This theorem is proven by structural induction. The corresponding formal proof for the case of the $AllPath$ operator is shown in Figure 5 (compare to the sketch of the proof in Section 5.3). In the first steps 1-10, we unfold the terms on the sequent using definitions of *embeddable* and $ctl2itl$. Also, we apply the axioms $sem - box$ and $sem - allpath$ defining the semantics of box and $AllPath$. Then, in the node 11, we apply the *case distinction* proof rule in order to prove separately both directions of the equivalence in the theorem. The proof of the right hand side (nodes 41-48) is easily accomplished by several quantifier instantiations and simplification. The left hand side (nodes 12-40) requires a bit more effort but is also not very complicated. The general idea of the formal proof directly corresponds to the informal one of Section 5.3.

The whole case study in KIV is available online[1]. Due to the very good tool support of the KIV theorem prover, it has been easier to do the proofs in KIV than on paper. We have corrected the semantics of the box operator several times before the equivalence was finally verified.

7 Outlook

We have shown how a given verification problem $\mathcal{K}, s \models_C \theta$ can be translated into an equivalent proof obligation in ITL$^+$. However, only *model based verification* problems can be translated, i.e. the translation requires a fixed model $\mathcal{K}$. Furthermore, we did not consider fair Kripke structures. Support for fair Kripke structures should be straightforward: fairness conditions can be added to the temporal formula $\psi_{\mathcal{K}}$ of Section 4.

An approach to verify *validity* of CTL* formulas in ITL$^+$ remains for future investigation. Verifying the validity of CTL* formulas (i.e. $\models_C \theta$) is known as *proof based verification*. In this context, there is an important difference between the modal operators of ITL$^+$ and the path operators of CTL*: while formulas in ITL$^+$ are valid, if they are satisfied by all intervals I, formulas in CTL* are valid only, if they are satisfied by all paths $p \in \mathcal{K}$ for all models $\mathcal{K}$. The idea of translating CTL formulas to ITL$^+$ for a given model has been to construct a suitable formula $\psi_{\mathcal{K}}$. A naive approach to embed

$$\models_C \theta$$

in ITL$^+$ would be to prove

$$\models_I \forall\ \psi.\ \mathrm{ctl2itl}(\theta, \psi)$$

This, however, requires variables for system description formulas as part of the logic. We would like to investigate, whether the strategy of symbolic execution is still applicable for variable system descriptions.

8 Conclusion

We presented a possibility to embed model based verification problems in branching time logic in a linear time logic (with modal operators). We think, that this result is of high theoretical relevance as it basically states that for a *given* model branching time is linear time extended with modal operators known from dynamic logic. In particular the proof calculus of a linear time logic (ITL$^+$) can be used for interactive verification.

For the praxis of verification, there is another useful consequence: the intuitive, interactive proof strategy of symbolic execution in ITL$^+$ can be combined with fully automatic model checking in CTL. In particular we were able to combine the interactive verifier KIV and the model checking tool SMV. The

[1] `http://www.informatik.uni-augsburg.de/swt/projects`

translation of the system model is straightforward, as finite automata (of SMVs input language) can be directly mapped to state charts which are directly supported by KIV. The opposite translation (from ITL$^+$ to CTL) is possible for an important subset of ITL$^+$ formulas.

Together, this is in our opinion a big step forward towards bringing together model checking and interactive verification approaches. We also see the possibility of transferring this approach to other interactive verification tools and model checkers.

References

[1] M. Balser. *Verifying Concurrent System with Symbolic Execution – Temporal Reasoning is Symbolic Execution with a Little Induction*. PhD thesis, University of Augsburg, Augsburg, Germany, 2005.

[2] W. Damm, B. Josko, H. Hungar, and A. Pnueli. A compositional real-time semantics of STATEMATE designs. In W.-P. de Roever, H. Langmaack, and A. Pnueli, editors, *COMPOS' 97*, volume 1536 of *LNCS*, pages 186–238. Springer, 1998.

[3] D. A. Peled E. M. Clarke Jr., O. Grumberg. *Model Checking*. The MIT Press, 1999.

[4] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 996–1072. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands, 1990.

[5] David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic Logic*. MIT Press, 2000.

[6] KIV homepage. http://www.informatik.uni-augsburg.de/swt/kiv.

[7] Wolfgang May. A tableau calculus for a temporal logic with temporal connectives. In *International Conference on Analytic Tableaux and Related Methods (TABLEAUX'99)*, volume LNCS. Springer, 1999.

[8] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1990.

[9] B. Moszkowski. A temporal logic for multilevel reasoning about hardware. *IEEE Computer*, 18(2):10–19, 1985.

[10] F. Ortmeier, W. Reif, and G. Schellhorn. Formal safety analysis of a radio-based railroad crossing using deductive cause-consequence analysis (DCCA). In *Proceedings of 5th European Dependable Computing Conference EDCC*, volume 3463 of *LNCS*. Springer, 2005.

[11] A. Thums, F. Ortmeier, W.Reif, and G. Schellhorn. Interactive verification of statecharts. In H. Ehrig, editor, *Integration of Software Specification Techniques for Applications in Engineering*, pages 355 – 373. Springer LNCS 3147, 2004.

[12] Andreas Thums. *Formale Fehlerbaumanalyse*. PhD thesis, Universität Augsburg, Augsburg, Germany, 2004. (in German).

[13] Peter H. Schmitt Wolfgang May. A tableau calculus for first-order branching time logic. In *International Conference on Formal and Applied Practical Reasoning (FAPR'96)*, volume LNCS. Springer, 1996.