Reducing the extensions of CTL with actions and real time

David N. Jansen, Roel J. Wieringa

Universiteit Twente, Faculteit der Informatica, Vakgroep IS.
Postbus 217, NL-7500 AE Enschede.
e-mail: dnjansen,roelw@cs.utwente.nl

Abstract

In this report, we present the logic ATCTL, which combines two known extensions of CTL, namely ACTL and TCTL. ACTL extends CTL with constructs to describe actions and TCTL extends it with constructs to specify real-time properties. ATCTL combines both extensions. We use ATCTL as a language for property specification in which we can express state properties, action properties, and real-time properties.

We show that the result can be reduced to ACTL as well as to TCTL, and therefore also to CTL. This makes model-checking ATCTL possible, because CTL model checkers exist.

*Partially supported by the DEIRDRE project, NWO 612-323-419, and the Esprit Working Group ASPIRE, contract number 22704.
1 Introduction

In this report we define the logic ATCTL, which combines two known extensions of CTL, namely ACTL and TCTL. CTL was defined by Emerson and Clarke [6] as a logic for model checking. ACTL, defined by De Nicola and Vaandrager [5], extends CTL with constructs to describe actions. ACTL contains no proposition symbols other than true. It was mainly designed to bridge the gap between state- and action-oriented specification techniques. TCTL, defined by Alur et al. [1], extends CTL with constructs to specify real-time properties.

ACTL as TCTL can be reduced to CTL, which means that they can be model checked, because model checkers for CTL exist. In addition, specific TCTL and ACTL model checkers have been written: Yovine [15, 2] has implemented Kronos for TCTL. ACTL model checkers can be found in [4, 8, 11].

ATCTL combines both extensions. We show that the result can be reduced to ACTL as well as to TCTL, and therefore also to CTL. This gives us a choice of tools for model checking.

The reason we define ATCTL is that we need a language for property specification in which we can express state properties, action properties, and real-time properties, that we will combine with a lightweight version of the Unified Modelling Language [13], that we call LUML [14]. Diagram-based specification notations such as the UML typically use states, actions and real time to specify systems. The reason for this is that software engineers find it easier to have constructs for all these aspects in a specification language. The drawback of the UML and some other diagram-based languages is that they have a baroque syntax and an often sloppily defined semantics. However, turning to formal specification languages, we find a different drawback: They are difficult to use for engineers. The goal of our research is to combine the best of both worlds and define a formal semantics for a lightweight version of the UML such that the resulting graphical language is easier to use for software engineers than traditional formal specification techniques and also has a well-defined semantics.

To achieve this, we have defined elsewhere an execution semantics for lightweight UML models in terms of a kind of labelled transition system that we call Clocked Labelled Kripke Structure (CLKS) [7]. Basically, a CLKS is a timed automaton in which the states are proposition valuations. A LUML model, consisting of a set of diagrams, will have a CLKS as intended semantics. Desired properties of the system will then be specified in ATCTL formulae. Via a reduction to TCTL, these can be checked using a TCTL model checker such as Kronos.

Comparison with other work. Latella, Majzik and Massink [10] model check UML statecharts using SPIN. Their statechart semantics, however, does not handle real time. Damm et al. [3] also model check Statemate statecharts, but these have a semantics differing from UML statecharts.

There are some minor differences between the ATCTL and both ACTL and TCTL. First, TCTL does not contain a “next state” operator, but ATCTL does. Second, we have extended ACTL with proposition symbols, so that we can give a more uniform presentation of the logics: CTL, ACTL and TCTL are just subsets of ATCTL. Third, Alur et al. [1] presented TCTL with a slightly different syntax: we use a syntax similar to Henzinger et al. [9], which separates clearly the action- and the real-time-oriented parts.
Structure of the report. We present the logics in the reverse order in which they were developed historically. After the preliminaries, we define CLKS in section 4. ATCTL is defined in section 5. Sections 6 and 7 then define TCTL as a subset of ATCTL and the reduction of ATCTL to TCTL. Finally, in sections 8 and 9, we present ACTL as a subset of ATCTL and show how ATCTL (except the clock reset action) can be reduced to ACTL. Since there are model checkers for ACTL [4], this gives us a second way to model check ATCTL. However, this reduction is much less efficient than the first one, so we will use Kronos in our further work.

2 Action logic

Action logic serves to specify constraints on sets of actions. Its atomic formulae are action symbols. Assume given a finite set of action symbols \( A = \{ a, b, \ldots \} \). Action terms \( \alpha, \beta, \ldots \) are then defined by the syntax:

\[
\alpha, \beta ::= \text{any} \mid a \mid -\alpha \mid \alpha + \beta
\]

For the sake of simplicity, we identify action symbols with their interpretation, actions. A set of concurrent atomic actions \( A, B, \ldots \in \mathcal{P}(A) \) is said to satisfy an action term, \( A \models \alpha \), in the following cases:

- \( A \models \text{any} \).
- \( A \models a \) iff \( a \in A \).
- \( A \models -\alpha \) iff \( A \not\models \alpha \).
- \( A \models \alpha + \beta \) iff \( A \models \alpha \) or \( A \models \beta \) (or both).

Speaking informally, any set is allowed by any. The terms a requires a set containing that action; \(-\alpha \) requires a set that is forbidden by \( \alpha \); and \( \alpha + \beta \) requires a set satisfying (at least) one of the constraints. We define: \( \alpha \& \beta := -(\alpha + -\beta) \). For example, \( a \& b \) is satisfied by the set \( \{ a, b \} \).

The definition looks quite similar to a Boolean algebra, and in fact, if one defines \( [\alpha] := \{ A \mid A \models \alpha \} \), the sets \([\alpha]\) become elements of the Boolean algebra \( \mathcal{P}(\mathcal{P}(A)) \) with set union, intersection and complement as its operations.

3 Time model

In our view of the world, time is modelled as the non-negative real numbers. We measure time by clocks which may be reset to zero at any moment. The clocks do not drift, but proceed all at the same rate.

Some state changes in a system may be enabled or disabled depending on time. To describe this, we use clock contraints. Assume given a finite set of clock symbols \( C = \{ t, t', \ldots \} \). Clock constraints \( c, c' \in CC \) are then given by the syntax: (for any \( n \in \mathbb{N} \))

\[
c, c' ::= t \leq n \mid t \geq n \mid -c \mid c \wedge c'
\]

Other operators are defined as usual, e.g. \( m < t := -t \leq m \). A time stamp is a valuation of the clock symbols: \( \chi : C \rightarrow \mathbb{R}_0^+ \). The satisfaction relation \( \chi \models c \) is straightforward; we will
not go into details. Later on, we will also use partial time stamps, which are partial functions $v : C \rightarrow \mathbb{R}_0^+$. Finally, we define two abbreviations, $\chi[C := 0]$ (where $C$ is a set of clock symbols) and $\chi + \delta$ (where $\delta \in \mathbb{R}_0^+$), by:

$$(\chi[C := 0])(t) := \begin{cases} 0 & \text{if } t \in C \\ \chi(t) & \text{otherwise} \end{cases}$$

$$(\chi + \delta)(t) := \chi(t) + \delta$$

4 System model

We base our system models on finite automata or finite state machines (a sort of Kripke structure) with some extensions:

- Real-time aspects come in from timed automata. Basically, the time how long a system is in a certain state is quantified.
- The transitions are labelled similar to labelled transition systems (LTSs).
- We use propositional logic as a model for data. In traditional LTSs, the “state” is reduced to possible behaviour and has no data. This is why we don’t use the term “LTS” for our type of models.

**Clocked Labelled Kripke Structures.** A clocked labelled Kripke structure (CLKS) is a kind of timed automaton. It consists of a set of locations, connected by transitions. Every transition bears a label containing the action set executed, an enabling condition on clocks (a clock constraint) that must be true when this transition is taken, and a (possibly empty) set of clocks to be reset.

More formally, assume given a set $\mathcal{P}$ of proposition symbols, a finite set $\mathcal{A}$ of action symbols and a finite set of clock symbols $\mathcal{C}$. A CLKS is a quintuple $\mathcal{M} = (L, \rightarrow, I, i_t, i_c)$ which satisfies the conditions:

- $L$ is a finite set of locations.
- $\rightarrow \subseteq L \times \mathcal{P}(\mathcal{A}) \times L$ is the transition relation. For any element $(l, A, l')$, the location $l$ is the source, $A$ is its label; $l'$ is the destination.
- $I : \mathcal{P} \rightarrow \mathcal{P}(L)$ is the interpretation of the proposition symbols.
- $i_t : L \rightarrow CC$ assigns a clock constraint to every location, the location invariant.
- $i_c : (\rightarrow) \rightarrow CC \times \mathcal{P}(\mathcal{C})$ assigns to every transition a clock constraint and a set of clocks to be reset.

A location invariant mainly serves to enforce some transitions: the system cannot stay in the location too long, but it will take action before some deadline.

If there is a transition $(l, A, l') \in \rightarrow$ which has clock constraint $c$ and resets the clocks in $C$, (i.e., $i_t(l, A, l') = (c, C)$ ), we write it as $l \xrightarrow{A[c]}_{C := 0} l'$. 

4
States and steps. A system run will be defined as a sequence of states and steps. A state is a pair of a location and a time stamp \((l, \chi)\). A system which is in a particular state may do one of two things: It may just wait; alternatively, it may take action steps, where it does some action to change from one location to a different one. So, we get two types of steps:

A time step is characterized by a delay \(\delta \in \mathbb{R}^+_\). It relates a state \((l, \chi)\) always to \((l, \chi + \delta)\).

For a valid time step, the location invariant of \(l\) has to hold all the time, i.e., for every \(0 \leq \varepsilon \leq \delta\), the timestamp \(\chi + \varepsilon\) satisfies \(i_t(l)\). We denote time steps by: \((l, \chi) \xrightarrow{\delta^{-}\delta^{-}} (l, \chi')\).

An action step belongs to a transition \(l \xrightarrow{A[l]} C = 0\) of the CLKS: \((l, \chi) \xrightarrow{A[l]} (l', \chi[C := 0])\) is a step provided that the timestamp \(\chi\) fulfils the transition’s clock constraint and the location invariant of the source \(l\), and the resulting timestamp fulfils the invariant of the destination \(l'\). In a formula, \(\chi \models c \land i_t(l)\) and \(\chi[C := 0] \models i_t(l')\).

If we talk about steps in general, we may write \((l, \chi) \xrightarrow{A^{-}\delta^{-}} (l', \chi')\) (where either \(\delta^{-} = -\) or \(A^{-} = \tau\)).

Runs. An execution of a system is represented by a run. A run of a system is a (possibly infinite) sequence of steps. We write this as:

\[
(l_0, \chi_0) \xrightarrow{A_{1}^{-}\delta_{1}^{-}} (l_1, \chi_1) \xrightarrow{A_{2}^{-}\delta_{2}^{-}} (l_2, \chi_2) \ldots
\]

where, for each \(i\), either \(\delta_{i}^{-} = -\) or \(A_{i}^{-} = \tau\).

It is possible to take several time steps without doing anything in between, or to take multiple action steps without waiting in between.\(^1\) This allows us to represent, say, a superstep in the StateMate semantics of statecharts as a sequence of steps in a run.

To identify a certain position in a run, we use the format \((i, \delta)\), where \(i \in \mathbb{N}\) is the index and \(0 \leq \delta \leq \delta_{i+1}^{-}\) is the local delay. (If \(\delta_{i+1}^{-} = -\), only \(\delta = 0\) is allowed.) In the run above, at position \((i, \delta)\), the system is in state \((l_i, \chi_i + \delta)\). We denote the time consumed up to position \((i, \delta)\) by: \(\Delta(i, \delta) := \delta + \sum_{j=1}^{i} \delta_{j}^{-}\) (counting the \(\delta_{j}^{-} = -\) as \(0\)).

We say that a position \((j, \varepsilon)\) lies before position \((i, \delta)\) if either \(j < i\) or \((j = i \land \varepsilon < \delta)\).

5 ATCTL

We now present a logic which may be used to specify properties of CLKSs.

Syntax. Assume given a set of proposition symbols \(\mathcal{P} = \{p, q, \ldots\}\), a set of action symbols \(\mathcal{A} = \{a, b, \ldots\}\) and a set of clock symbols: \(\mathcal{C} = \{t, t', \ldots\}\). The syntax of ATCTL is then defined as:

\[
\varphi, \psi ::= \bot \mid p \mid \neg \varphi \mid \varphi \land \psi \mid t \leq n \mid t \geq n \mid \text{reset } t \text{ in } \varphi \mid \exists X_\alpha \varphi \mid \forall X_\alpha \varphi \mid \exists (\varphi_\alpha U \psi) \mid \forall (\varphi \_ U \psi) \mid \exists (\varphi_\alpha U_\beta \psi) \mid \forall (\varphi_\alpha U_\beta \psi)
\]

\(^1\)We do allow for so-called Zeno behaviour. That is, we allow a system to take infinitely many action steps in finite time. We do this because the tool Kronos does so and because it simplifies the reduction to ACTL.
The common formulae have their usual meaning. \( t \leq n \) and \( t \geq n \) hold in a state where the time stamp says that clock \( t \) is in the indicated range. \( \text{reset} \ t \) in \( \varphi \) is a sort of hypothetical reasoning: “If \( t \) were zero (ceteris paribus), then \( \varphi \) would hold.” \( \exists X \alpha \varphi \) states that there is some action step from this state satisfying the constraints \( \alpha \), such that \( \varphi \) holds afterwards. \( \forall X \alpha \varphi \) states that every step from this state is an action step satisfying the constraint \( \alpha \) and \( \varphi \) will hold afterwards. \( \exists (\varphi \alpha \mathcal{U} \psi) \) is valid in a state where it is possible to do several \( \alpha \) steps (during which \( \varphi \) holds), and finally \( \psi \) becomes true. \( \exists (\varphi \alpha \mathcal{U}_{\beta} \psi) \) is valid in a state where it is possible to do several \( \alpha \) steps (during which \( \varphi \) holds), and finally do a \( \beta \) step, such that \( \psi \) is valid immediately afterwards.

We include two different “until” operators for several reasons: In the reduction to TCTL, \( \alpha \mathcal{U} \) is the more natural operator. However, in several examples, \( \alpha \mathcal{U}_{\beta} \) is needed. In ACTL, both are defined.\(^2\)

The simpler modal operators \( \square, \Diamond \) are defined as usual, e.g.:

\[
\exists \Diamond \alpha \varphi := \exists (T \land \mathcal{U} \alpha \varphi) \quad \forall \square \alpha \varphi := \neg \exists \Diamond \alpha \neg \varphi
\]

In some variants of TCTL, slightly different operators are defined which mix up the time- and action-oriented features. We may define them as abbreviations, e.g.: (\( t_{\text{new}} \) denotes a new clock symbol)

\[
\exists (\varphi \alpha \mathcal{U}^{\leq n} \psi) := \text{reset} \ t_{\text{new}} \ \text{in} \ \exists (\varphi \alpha \mathcal{U} (\psi \land t_{\text{new}} \leq n))
\]

Semantics. Assume given an ATCTL-model \( \mathcal{M} = (L, \rightarrow, I, \bar{u}, \bar{\chi}) \). We will interpret a formula in a state of the model \((l, \chi)\) together with a partial time stamp \( v \), which serves to interpret bound clock variables: \( v(t) \) is defined if \( t \) is in the scope of a \( \text{reset} \) \( t \) \( \text{in} \) \( \alpha \).\(^3\)

The satisfaction relation \((\mathcal{M}, l, \chi) \models_v \varphi \) is defined as:

- \((\mathcal{M}, l, \chi) \models_v \perp \) \( \perp \) never.
- \((\mathcal{M}, l, \chi) \models_v p \) if \( l \in I(p) \).
- \((\mathcal{M}, l, \chi) \models_v \neg \varphi \) if \( (\mathcal{M}, l, \chi) \not\models_v \varphi \).
- \((\mathcal{M}, l, \chi) \models_v \varphi \land \psi \) if \( (\mathcal{M}, l, \chi) \models_v \varphi \) and \((\mathcal{M}, l, \chi) \models_v \psi \).
- \((\mathcal{M}, l, \chi) \models_v t \leq n \) if \( f(t) \leq n \), where \( f(t) = \begin{cases} v(t) & \text{if defined} \\ \chi(t) & \text{otherwise.} \end{cases} \)
- \((\mathcal{M}, l, \chi) \models_v t \geq n \) if \( f(t) \geq n \), where \( f(t) \) is defined as above.
- \((\mathcal{M}, l, \chi) \models_v \text{reset} \ t \ in \ \varphi \) if \((\mathcal{M}, l, \chi) \models_v [t]_{t = 0} \varphi \).
- \((\mathcal{M}, l, \chi) \models_v \exists X \alpha \varphi \) if there is a run \((l, \chi) \xrightarrow{A} (l', \chi') \cdots \) which satisfies \( A \models \alpha \) and \((\mathcal{M}, l', \chi') \models_v \varphi \).
- \((\mathcal{M}, l, \chi) \models_v \forall X \alpha \varphi \) if every run starting in \((l, \chi)\) has the form \((l, \chi) \xrightarrow{A} (l', \chi') \cdots \) and satisfies \( A \models \alpha \) and \((\mathcal{M}, l', \chi') \models_v \varphi \).

\(^2\)The construct \( \exists (\varphi \alpha \mathcal{U}_{\beta} \psi) \) is definable as an abbreviation; for \( \forall (\varphi \alpha \mathcal{U}_{\beta} \psi) \), we couldn’t find any definition.

\(^3\)Henzinger et al. \[9\] call \( v \) a clock environment.
• \((\mathcal{M}, t, \chi) \models_w \exists (\varphi \mathcal{A} \mathcal{U} \psi)\) iff there is a run \((l_0, \chi_0) \xrightarrow{A_1^1} (l_1, \chi_1) \xrightarrow{A_2^2} \cdots\) starting in \((l, \chi) = (l_0, \chi_0)\) which satisfies the simple until condition: there is a position \((i, \delta)\) in the run such that
  - \((\mathcal{M}, l_i, \chi_i + \delta) \models_w \psi;\) and
  - \((\mathcal{M}, l_j, \chi_j + \varepsilon) \models_w \varphi \lor \psi,\) for every position \((j, \varepsilon)\) before \((i, \delta)\); and
  - Either step \(j\) is a time step, or \(A_j^j \models \alpha,\) for every \(j\) with \(0 < j \leq i.\)

• \((\mathcal{M}, t, \chi) \models_w \forall (\varphi \mathcal{A} \mathcal{U} \psi)\) iff every run starting in \((l, \chi)\) satisfies the simple until condition.

• \((\mathcal{M}, t, \chi) \models_w \exists (\varphi \mathcal{A} \mathcal{U}_\beta \psi)\) iff there is a run \((l_0, \chi_0) \xrightarrow{A_1^1} (l_1, \chi_1) \xrightarrow{A_2^2} \cdots\) starting in \((l, \chi) = (l_0, \chi_0)\) which satisfies the double until condition: there is a position \((i, 0)\) in the run \((i > 0)\) such that
  - \((\mathcal{M}, l_i, \chi_i) \models_w \psi;\) and
  - \((\mathcal{M}, l_j, \chi_j + \varepsilon) \models_w \varphi,\) for every position \((j, \varepsilon)\) before \((i, 0)\); and
  - \(A_i^i \models \beta;\) and
  - Either step \(j\) is a time step, or \(A_j^j \models \alpha,\) for every \(j\) with \(0 < j < i.\)

• \((\mathcal{M}, t, \chi) \models_w \forall (\varphi \mathcal{A} \mathcal{U}_\beta \psi)\) iff every run starting in \((l, \chi)\) satisfies the double until condition.

In the simple until condition, the begin of the run has to satisfy \(\varphi \lor \psi.\) This is to allow for conditions like \(\exists(t \leq 5 \text{ any} \mathcal{U} t > 5),\) where the formula \(t > 5\) does not hold in a left-closed interval. For the double until condition of \(\exists (\varphi \mathcal{A} \mathcal{U}_\beta \psi),\) this is not necessary, for there is a clear point where \(\psi\) is tested. (Note that the double until condition requires that the run contain at least one \(\beta\) action.)

6 TCTL

TCTL is a temporal logic without action modalities introduced by Alur et al. [1, 9]. It extends CTL [6] by real time. There is a model checker Kronos [2, 15] for TCTL, so if we can reduce ATCTL to TCTL, we can check ATCTL formulae. Only some combinations where a \(t \geq n\) or \(t \leq n\) is too far away from its \textit{reset} \(t\) in operator can’t be handled by Kronos.

TCTL is based on a real-numbered, continuous time model. We can see TCTL as ATCTL over the empty action symbol set \((A = \emptyset)\) and without the operators \(X_\alpha\) or \(\mathcal{A}_\beta.\) They are omitted because without action labels, there is no unique “next state” notion. Transitions and steps are all unlabelled (or, more precisely, they all are labelled with the action set \(\emptyset\)). So, TCTL syntax is defined as:

\[ \varphi, \psi ::= \bot \mid p \mid \neg \varphi \mid \varphi \land \psi \mid t \leq n \mid t \geq n \mid \text{reset} \ t \text{ in} \ \varphi \mid \exists (\varphi \mathcal{U} \psi) \mid \forall (\varphi \mathcal{U} \psi) \]

The operators have semantics similar to the corresponding ATCTL operators.

Abbreviations like \(\exists (\varphi \mathcal{U} \leq n \psi)\) and \(\exists \phi \varphi\) are defined similarly to the ATCTL abbreviations.
**TCTL models.** TCTL models are timed automata. Roughly speaking, they are CLKSs without action labels. A *timed automaton* (or clocked Kripke structure) is a quintuple $T = (L, \rightarrow, I, i_t, i_t)$ which has the same elements as a CLKS, except that the transition relation is binary:

- $L$ is a finite set of *locations*.
- $\rightarrow \subseteq L \times L$ is the transition relation. For any element $(l, l')$, the location $l$ is the source and $l'$ is the destination.
- $I : \mathcal{P} \to \mathcal{P}(L)$ is the interpretation of proposition symbols.
- $i_t : L \to CC$ assigns a clock constraint to a location, its *location invariant*.
- $i_t : (\rightarrow) \to CC \times \mathcal{P}(C)$ assigns to every transition a clock constraint and a set of clocks to be reset.

We will denote a transition as $l \xrightarrow{[c]}_{C:=0} l'$, similar to the transitions of ATCTL.

States are defined the same way as in ATCTL. Action steps bear no action label. However, we continue to distinguish action steps $(l, \chi) \xrightarrow{a} (l', \chi')$ and time steps. Runs are made up of steps as in ATCTL. For more details, see [9].

Here too, we interpret a formula $\varphi$ over a state $(l, \chi)$ of a timed automaton $T$ and a partial time stamp $v$. The definition of the satisfaction relation $(T, l, \chi) \models_v \varphi$ is a simple adaptation of the corresponding definition for ATCTL:

- $(T, l, \chi) \models_v \bot$ never.
- $(T, l, \chi) \models_v p$ iff $l \in I(p)$.
- $(T, l, \chi) \models_v \neg \varphi$ iff $(T, l, \chi) \not\models_v \varphi$.
- $(T, l, \chi) \models_v \varphi \land \psi$ iff $(T, l, \chi) \models_v \varphi$ and $(T, l, \chi) \models_v \psi$.
- $(T, l, \chi) \models_v t \leq n$ iff $f(t) \leq n$, where $f(t) = \begin{cases} v(t) & \text{if defined} \\ \chi(t) & \text{otherwise} \end{cases}$.
- $(T, l, \chi) \models_v t \geq n$ iff $f(t) \geq n$, where $f(t)$ is defined as above.
- $(T, l, \chi) \models_v \text{reset } t \text{ in } \varphi$ iff $(T, l, \chi) \models_{v\{t\}:=0} \varphi$.
- $(T, l, \chi) \models_v \exists (\varphi U \psi)$ iff there is a run $(l_0, \chi_0) \xrightarrow{\delta_1} (l_1, \chi_1) \xrightarrow{\delta_2} (l_2, \chi_2) \cdots$ starting in $(l, \chi) = (l_0, \chi_0)$ which satisfies the *until condition*: there is a position $(i, \delta)$ in the run such that
  - $(T, l_i, \chi_i + \delta) \models_{v + \Delta(i, \delta)} \psi$; and
  - $(T, l_j, \chi_j + \varepsilon) \models_{v + \Delta(j, \varepsilon)} \varphi \lor \psi$, for every position $(j, \varepsilon)$ before $(i, \delta)$.
- $(T, l, \chi) \models_v \forall (\varphi U \psi)$ iff every run starting in $(l, \chi)$ satisfies the until condition.
7 Reduction of ATCTL to TCTL

We now define a reduction of ATCTL formulae and models to TCTL. It extends the reduction defined by De Nicola and Vaandrager [5]: Most of the operators of ATCTL have a direct correspondence in TCTL. However, we had to find a workaround for the operators $X_a$ and $\alpha U_b$.

Further, only the action modalities have to be translated: For every transition, add an action location; translate actions to special propositions valid in some of the action locations. An additional proposition act will serve to distinguish the action locations from non-action locations. We add time constraints to ensure that actions are instantaneous. Transition clock constrains and clock resets are translated into constraints and resets on the transition into an action location.

Translation of models. We base the models in this section on an action set $\mathcal{A}$ and a proposition symbol set $\mathcal{P}$. Assume given a ATCTL model $\mathcal{M} = (L, \rightarrow, I, i_t, i_r)$. We define the TCTL model $\mathcal{M}^T = (L^T, \rightarrow^T, I^T, i_t^T, i_r^T)$ as

- $\mathcal{P}^T = \mathcal{P} \cup \mathcal{A} \cup \{\text{act}\}$ is the underlying set of proposition symbols.

- We add one new clock $t_{new}$ to the clocks of $\mathcal{M}$. We need this clock to ensure that actions are instantaneous.

- We add an action location for every transition: $L^T = L \cup \{\rightarrow\}$.

- For every transition $tr = (l, A, l')$ of $\mathcal{M}$ with $i_t(tr) = (c, C)$, we define two new transitions in $\mathcal{M}^T$, namely $l \xrightarrow{c} C \cup \{t_{new} = 0\} \rightarrow^T tr$ and $tr \xrightarrow{t_{new} = 0} l'$.

- The reset of $t_{new}$ and the clock constraint $t_{new} = 0$ jointly take care that the system does not spend time in the action location. See Fig. 1 for an example.

- The interpretation of the proposition symbols is:

\[
I^T(p) := I(p) \\
I^T(\text{act}) := \rightarrow \\
I^T(a) := \{ (l, A, l') \in \rightarrow | a \in A \}
\]

- The location invariant function $i_t^T$ is:

\[
i_t^T(l) := i_t(l) \\
i_t^T(l, A, l') := (t_{new} = 0)
\]

Translation of states, steps and runs. Now, continue with states: as the translated model has one new clock, there is a slight difference between states of $\mathcal{M}$ and their translation. In $(l, \chi^T)$, one may choose the value of $t_{new}$ nearly arbitrarily.

In the translation of steps, we distinguish action and time steps

- $(l, \chi) \xrightarrow{A} (l', \chi')$ is translated to $(l, \chi^T) \xrightarrow{c} ((l, A, l'), \chi^T) \xrightarrow{t_{new} = 0} (l', \chi^T)$. Set $\chi^T(t_{new}) = 0$. 

\footnote{We assume that the mentioned sets are disjoint.}
\footnote{If there are no other clocks, even $t_{new}$ may be suppressed, as it is no more possible to measure the time consumed by an action.}
\[
\left(\begin{array}{c}
p \\
\{a_2, b\}
\end{array}\right)^T \rightarrow \left(\begin{array}{c}
p \\
q
\end{array}\right) = \left(\begin{array}{c}
p \\
\text{ACT,} \\
a_2, b
\end{array}\right) \rightarrow \left(\begin{array}{c}
p \\
q
\end{array}\right)
\]

Figure 1: Translation of models. (Time constraints are omitted.)

- \((l, \chi) \xrightarrow{\delta} (l', \chi')\) is translated to \((l, \chi^T) \xrightarrow{\delta} (l', \chi'^T)\). Set \(\chi'^T(t_{\text{new}}) = \chi^T(t_{\text{new}}) + \delta\).

Action steps are split up into two parts, corresponding to the split transitions. Time steps are translated directly. A run, then, is translated stepwise. As the translation of each individual step is one or two step(s) in \(M^T\), their composition is a run again.

This mapping is injective and essentially surjective. Only for the new clock symbol \(t_{\text{new}}\), not every value is hit. One could say, in a loose manner of writing: The extended mapping \(\{\text{runs of } M\} \times \mathbb{R}_0^+ \rightarrow \{\text{runs of } M^T\} \text{ starting in non-action locations}\) is a bijection.

### 7.1 Translation of formulae

**Action terms.** We reinterpret action terms as a special kind of propositional formulae, for we have reinterpreted action symbols as a special kind of proposition symbols:

\[
\begin{align*}
\text{any}^T &= \text{ACT} \\
(-\alpha)^T &= -\alpha^T \\
\alpha^T + \beta^T &= \alpha^T \lor \beta^T
\end{align*}
\]

**ATCTL formulae.** Propositional formulae are translated straightforward. ATCTL operators are mostly translated to the corresponding TCTL operators:

\[
\begin{align*}
\bot^T &= \bot \\
(-\varphi)^T &= -\varphi^T \\
(t \leq n)^T &= t \leq n \\
(t \geq n)^T &= t \geq n \\
(reset t \text{ in } \varphi)^T &= reset t \text{ in } \varphi^T \\
(\exists X \alpha \varphi)^T &= \exists (\neg \text{ACT } U^0 \exists (\alpha^T U^0 \varphi^T)) \\
(\forall X \alpha \varphi)^T &= \forall (\neg \text{ACT } U^0 \exists (\alpha^T U^0 \varphi^T)) \\
(\exists (\varphi \alpha U \psi)^T &= \exists (\neg \text{ACT } \land \alpha^T U \neg \text{ACT } \land \psi^T) \\
(\forall (\varphi \alpha U \psi)^T &= \forall ((\neg \text{ACT } \land \alpha^T) \lor (\neg \text{ACT } \land \psi^T))
\end{align*}
\]

\(\exists(\varphi U^0 \psi)\) operator is an abbreviation for: \(\text{reset } t_{\text{new}}' \text{ in } \exists (\varphi U (t_{\text{new}}' = 0 \land \psi))\), where \(t_{\text{new}}'\) denotes a new clock symbol.

### 7.2 Equivalence theorem

**Theorem 1** Let \(M = (L, \rightarrow, I, i_t, i_s)\) be an ATCTL model, \((l, \chi)\) one of its states, \(\varphi\) an ATCTL formula and \(\nu\) a partial time stamp.

Then, \((M, l, \chi) \models \varphi \iff (M^T, l, \chi^T) \models \varphi^T\).

**Proof:** By induction over the formulae.
• \((\mathcal{M}, l, \chi) \models_v \bot \iff (\mathcal{M}^T, l, \chi^T) \models_v \bot^T\): trivially.

• \((\mathcal{M}, l, \chi) \models_v p \iff (\mathcal{M}^T, l, \chi^T) \models_v p^T\): The left side holds iff \(l \in I(p)\), and the right side holds iff \(l \in I^T(p^T) = I(p)\).

• \((\mathcal{M}, l, \chi) \models_v \neg \varphi \iff (\mathcal{M}^T, l, \chi^T) \models_v (\neg \varphi)^T\): By induction.

• \((\mathcal{M}, l, \chi) \models_v \varphi \land \psi \iff (\mathcal{M}^T, l, \chi^T) \models_v (\varphi \land \psi)^T\): By induction.

• \((\mathcal{M}, l, \chi) \models_v t \leq n \iff (\mathcal{M}^T, l, \chi^T) \models_v (t \leq n)^T\): The left side holds iff \(f(t) \leq n\), where

\[
f(t) = \begin{cases} 
  v(t) & \text{if defined} \\
  \chi(t) & \text{otherwise} 
\end{cases}
\]

The right side holds iff \(f^T(t) \leq n\), where \(f^T(t)\) is defined similarly for \(v\) and \(\chi^T\). It turns out that \(f(t) = f^T(t)\).

• \((\mathcal{M}, l, \chi) \models_v t \geq n \iff (\mathcal{M}^T, l, \chi^T) \models_v (t \geq n)^T\): Analogous.

• \((\mathcal{M}, l, \chi) \models_v \text{reset } t \text{ in } \varphi \iff (\mathcal{M}^T, l, \chi^T) \models_v (\text{reset } t \text{ in } \varphi)^T\): simple.

• \((\mathcal{M}, l, \chi) \models_v \exists X_{\alpha} \varphi \iff (\mathcal{M}^T, l, \chi^T) \models_v (\exists X_{\alpha} \varphi)^T\): The left side holds iff there is a run \((l, \chi) \xrightarrow{A} (l', \chi') \rightarrow \cdots\) such that \(A \models \alpha\) and \((\mathcal{M}, l', \chi') \models_v \varphi\). Then, there is a run of \(\mathcal{M}^T\), namely \((l, \chi^T) \rightarrow ((l, A, l'), \chi'^T) \rightarrow (l', \chi'^T) \rightarrow \cdots\). It satisfies \((\exists X_{\alpha} \varphi)^T \equiv \exists(-\alpha \text{ACT} \cup 0 \exists(\alpha \text{ACT} \land \alpha^T \cup \varphi^T))\). This is the right-hand side.

The converse (the right-hand side implies the left-hand side) uses the correspondence between runs of \(\mathcal{M}\) and runs of \(\mathcal{M}^T\) starting in non-action locations in a similar way.

• \((\mathcal{M}, l, \chi) \models_v \forall X_{\alpha} \varphi \iff (\mathcal{M}^T, l, \chi^T) \models_v (\forall X_{\alpha} \varphi)^T\): Analogous.

• \((\mathcal{M}, l, \chi) \models_v \exists(\varphi \alpha \text{U} \psi)\) holds iff there is a run \((l_0, \chi_0) \xrightarrow{A_1^1} (l_1, \chi_1) \xrightarrow{A_2^2} (l_2, \chi_2) \cdots\) which satisfies the simple until condition. Let \((i, \delta)\) be the position used in the definition. Translate this run to a run of \(\mathcal{M}^T\), and let \((i^T, \delta)\) be the corresponding position in the translated run. It satisfies the translated condition, \((\mathcal{M}^T, l, \chi^T) \models_v \exists(\varphi \alpha \text{U} \psi)^T\), that is:

\[
\begin{align*}
- (\mathcal{M}^T, l_{i^T}, \chi_{i^T}^T + \delta) & \models_v \Delta(i^T, \delta) \rightarrow -\alpha \text{ACT} \land \psi^T; \\
- (\mathcal{M}^T, l_{j^T}, \chi_{j^T}^T + \varepsilon) & \models_v \Delta(j^T, \varepsilon) (\alpha \text{ACT} \land \chi^T) \land (\neg \alpha \text{ACT} \land \varphi^T) \land (\neg \alpha \text{ACT} \land \psi^T), \text{ for every position } (j^T, \varepsilon) \text{ before } (i^T, \delta),
\end{align*}
\]

These two conditions are easily verified by induction. Note that \((\alpha \text{ACT} \land \alpha^T)\) is satisfied by the action locations inserted into the translation.

The only if part holds because the mapping of runs is essentially surjective.

• \((\mathcal{M}, l, \chi) \models_v \forall(\varphi \alpha \text{U} \psi) \iff (\mathcal{M}^T, l, \chi^T) \models_v (\forall(\varphi \alpha \text{U} \psi))^T\): Analogous.

• \((\mathcal{M}, l, \chi) \models_v \exists(\varphi \alpha \text{U}_\beta \psi) \iff (\mathcal{M}^T, l, \chi^T) \models_v (\exists(\varphi \alpha \text{U}_\beta \psi))^T\): The left side holds iff there is a run \((l_0, \chi_0) \xrightarrow{A_1^1} (l_1, \chi_1) \xrightarrow{A_2^2} (l_2, \chi_2) \rightarrow \cdots\) which satisfies the double until condition.

11
As with the simple until, translate this run to a run of $\mathcal{M}^T$; the translation will satisfy the translated formula. The only difference is that we have to reach a state where $\exists (\text{ACT} \land \beta^T \ U^0 \rightarrow \text{ACT} \land \psi^T)$ holds. This is the action state immediately before the position $(i, 0)$ of the original run where $\psi$ becomes true.

- $(\mathcal{M}, l, \chi) \models \forall (\varphi \alpha U^\beta \psi) \iff (\mathcal{M}^T, l, \chi^T) \models \forall (\varphi \alpha U^\beta \psi)^T$: Analogous.

### 7.3 Abbreviations

It might be helpful to give direct translations of some of the abbreviations. Later on, we will use Kronos, and as this tool has some of the abbreviations defined in section 5 as its primitives, direct translations will be more efficient.

\[
\exists (\varphi \alpha U^\beta \psi)^T = \exists (\text{ACT} \land \alpha^T) \lor (\neg \text{ACT} \land \varphi^T) \ U^0 \exists (\text{ACT} \land \beta^T \ U^0 \rightarrow \text{ACT} \land \psi^T) \\
\forall (\varphi \alpha U^\beta \psi)^T = \forall (\text{ACT} \land \alpha^T) \lor (\neg \text{ACT} \land \varphi^T) \ U^0 \exists (\text{ACT} \land \beta^T \ U^0 \rightarrow \text{ACT} \land \psi^T)
\]

**Lemma 1** The translations defined here and the original translations are equivalent.

For more abbreviations, we just can abbreviate both sides of the original definitions to give a more concise translation, e.g.: $(\exists H \alpha \varphi)^T = \exists H \exists (\text{ACT} \land \alpha^T \ U \rightarrow \text{ACT} \land \varphi^T)$.

### 8 ACTL

ACTL is a logic with action modalities, but (originally) with $\perp$ as only atomic proposition and without real time, introduced by De Nicola and Vaandrager [5]. It adapts CTL [6] to action-oriented situations. We have added proposition symbols to the original ACTL, so we are able to describe it just as an extension of CTL. Another addition we have made is that the transitions are labelled with a set of actions rather than a single action.

We can see ACTL as ATCTL over the empty clock set: $C = \emptyset$. Then, models, transitions and steps don’t bear any real-time information. (or, stated differently, every real number is acceptable as delay of a time step.) Time steps in ACTL models are similar to internal steps denoted by $\tau$ in other systems. The clock constraints $\cdot \leq n$, $\cdot \geq n$ and the operator reset $\cdot$ in $\varphi$ cannot be used in ACTL.

De Nicola et al. [4] have constructed a model checker for ACTL which translates ACTL formulae and models to CTL and relies on a CTL model checker to do the rest. So, we may choose to translate ATCTL to ACTL and use this model checker. Although there are ATCTL formulae that can only be checked this way, model checking is less efficient this way because the state space is blown up heavily in the reduction.

**ACTL syntax.** We repeat the constructs of ATCTL which make up ACTL:

\[
\varphi, \psi ::= \perp \mid p \mid \neg \varphi \mid \varphi \land \psi \\
\exists \alpha X \varphi \mid \forall \alpha X \varphi \mid \exists (\varphi \alpha U^\beta \psi) \mid \forall (\varphi \alpha U^\beta \psi) \mid \exists (\varphi \alpha U^\beta \psi)^T \\
\forall (\varphi \alpha U^\beta \psi)^T
\]

---

*There is some asymmetry in our definition: In most systems, $\tau$ steps may change the state of the system thoroughly. In our setting, only transitions may become enabled or disabled (because of their clock constraints) after an internal step; no propositional formula changes its truth value.*
The operators have semantics similar to the corresponding ATCTL operators. Abbreviations like $\exists \phi \alpha \varphi$ are defined similarly to the ATCTL abbreviations.

**ACTL models.** Labelled Kripke structures (LKS) are ACTL models. (Originally, ACTL models were called labelled transition systems.) They are similar to ATCTL models without clocks. Assume given a set of proposition symbols $P$ and a finite set of action symbols $A$. A labelled Kripke structure is a triple $L = (L, \to, I)$ which satisfies the conditions:

- $L$ is a finite set of locations.
- $\to \subseteq L \times (P(A) \cup \{\top\}) \times L$ is the transition relation. For any element $(l, A^T, l')$, the location $l$ is the source, $A^T$ is the action set, and $l'$ is the destination.
- $I : P \to \mathbb{P}(L)$ is the interpretation of proposition symbols.

We will denote a transition as $l \xrightarrow{A^T} l'$, similar to the transitions of ATCTL. A state in ATCTL is a pair of a location and a time stamp, but as there is no relevant time stamp in ACTL, states and locations are the same here. Steps $l \xrightarrow{A^T} l'$ are defined as in ATCTL. (Be careful to distinguish transitions from steps.) Time steps have disappeared; we will see that they are translated to $\top$ steps below. Finally, runs are defined the same way as in ATCTL. For more detailed information, see [4, 5].

**ACTL semantics.** Assume given an ACTL-model $L = (L, \to, I)$. We will interpret a formula in a state $l$ of the model. The satisfaction relation $(L, l) \models \varphi$ is similar to the satisfaction relation of ATCTL, but doesn’t include time stamps:

- $(L, l) \models \bot$ never.
- $(L, l) \models p$ iff $l \in I(p)$.
- $(L, l) \models \neg \varphi$ iff $(L, l) \not\models \varphi$.
- $(L, l) \models \varphi \land \psi$ iff $(L, l) \models \varphi$ and $(L, l) \models \psi$.
- $(L, l) \models \exists X \alpha \varphi$ iff there is a run $l \xrightarrow{A^T} l' \xrightarrow{A^T} \cdots$ with $A \models \alpha$ and $(L, l') \models \varphi$.
- $(L, l) \models \forall X \alpha \varphi$ iff every run starting in $l$ has the form $l \xrightarrow{A^T} l' \xrightarrow{A^T} \cdots$ and satisfies $A \models \alpha$ and $(L, l') \models \varphi$.
- $(L, l) \models \exists (\varphi \alpha U \psi)$ iff there is a run $l_0 \xrightarrow{A_l^1} l_1 \xrightarrow{A_l^1} l_2 \cdots$ starting in $l = l_0$ which satisfies the simple until condition: there is a position $i \in \mathbb{N}$ such that
  - $(L, l_i) \models \varphi$; and
  - $(L, l_j) \models \varphi$, for every position $j$ with $0 < j < i$; and
  - $A_l^j$ is $\top$ or $A_l^j \models \alpha$, for every $j$ with $0 < j \leq i$.
- $(L, l) \models \forall (\varphi \alpha U \psi)$ iff every run starting in $l$ satisfies the simple until condition.
- $(L, l) \models \exists (\varphi \alpha U_\beta \psi)$ iff there is a run $l_0 \xrightarrow{A_l^1} l_1 \xrightarrow{A_l^1} l_2 \rightarrow \cdots$ starting in $l = l_0$ which satisfies the double until condition: there is a position $i \in \mathbb{N}$ in the run such that
- \((\mathcal{L}, l_i) \models \psi\); and
- \((\mathcal{L}, l_j) \models \varphi\), for every position \(j\) with \(0 \leq j < i\); and
- \(A^\mathcal{L}_j \models \beta\); and
- \(A^\mathcal{L}_j\) is \(\top\) or \(A^\mathcal{L}_j \models \alpha\), for every \(j\) with \(0 < j < i\).

- \((\mathcal{L}, l) \models \forall (\varphi A^\mathcal{L}_\alpha \psi)\) iff every run starting in \(l\) satisfies the double until condition.

9 Reduction of ATCTL to ACTL

The reduction to ACTL, eliminating real time, is a bit more complicated than the first reduction. It is similar to the reduction given by Alur et al. [1].

We construct a region automaton. This is an automaton where a state is an equivalence class of states in the ATCTL model. It is, in general, a very large automaton which may be too large for existing model checkers.

The correspondence between an ATCTL model and its region automaton is a bit looser than the correspondence between an ATCTL model and its translated TCTL model: As the clock information in the translated model is lost, the mapping is no more injective. In our translation, we didn’t find a satisfactory mapping of the \texttt{reset} \cdot \varphi \texttt{operator} yet.

**Equivalent states.** Assume given a CLKS \(M = (L, \rightarrow, I, i; i)\) over a set of action symbols \(A\), a set of proposition symbols \(P\), and a set of clock symbols \(C\). We first will define an equivalence relation of time stamps \(\chi\) of the CLKS; that leads us to an equivalence relation of states.

Note that all clock constraints \(t \leq n\) and \(t \geq n\) contain only natural numbers \(n\). In an ATCTL formula, one may express which clock is the first to change its integer part. So, time stamps with the same integer parts and the same ordering of the fractional parts cannot be distinguished by ATCTL formulae and are declared equivalent. Further, assume given an \(N\) such that that in every interesting formula \(t \leq n\) or \(t \geq n\), we have \(n \leq N\).\(^7\) Time stamps whose clocks exceed \(N\) are declared equivalent. The equivalence relation is called \(N\)-similarity.

So, given two time stamps \(\chi\) and \(\chi'\), let \(R_\chi = \{ t \in C \mid \chi(t) \leq N\}\), and define \(R_\chi'\) similarly. The two time stamps are \(N\)-similar, denoted \(\chi \sim_N \chi'\), if:

- The same clocks are in the relevant area: \(R_\chi = R_\chi'\); and
- For every \(t \in R_\chi\), the integer parts are the same: \([\chi(t)] = [\chi'(t)]\); and
- For every pair of \(t, t' \in R_\chi\), the orderings of the fractional parts are the same: \([\chi(t) - \chi(t')] = [\chi'(t) - \chi'(t')]\);\(^8\) and
- For every \(t \in R_\chi\), the fractional parts are both zero or both non-zero: \(\text{fract}(\chi(t)) = 0\) iff \(\text{fract}(\chi'(t)) = 0\).

---

\(^7\)In a practical situation, one only deals with a finite set of interesting formulae (or, properties that must be checked); in this set, clock constraints are always bound by some \(N \in \mathbb{N}\).

\(^8\)This clause is equivalent to:

- For every pair of \(t, t' \in R_\chi\), we have \(\text{fract}(\chi(t)) < \text{fract}(\chi(t'))\) iff \(\text{fract}(\chi'(t)) < \text{fract}(\chi'(t'))\).
In this definition, \([r]\) denotes the integer part of the real number \(r\), such that \(r - 1 < [r] \leq r \land [r] \in \mathbb{Z}\). The fractional part is \(\text{frac}(r) = r - [r]\).

The equivalence relation is then extended to states by: two states \((l, \chi)\) and \((l', \chi')\) are \(N\)-similar, if \(l = l'\) and \(\chi \sim_N \chi'\). The equivalence classes of this extended relation are \(\text{regions}\), written as \([[l, \chi]]_\sim\).

### 9.1 Translation of models

The region automaton is the translation of an ATCTL model. Its locations are the regions; its transitions correspond to the transitions in the ATCTL model. Define an ACTL model \(\mathcal{M}^A = (I^A, \rightarrow^A, I^A)\) (this is the region automaton) by:

- \(\mathcal{P}^A = \mathcal{P} \cup \{t^\leq n, t^\geq n \mid t \in \mathcal{C} \text{ and } n \leq N\}\) is the underlying set of proposition symbols.
- States are the equivalence classes constructed above:
  \[L^A = \{[[l, \chi]]_\sim \mid (l, \chi) \text{ is a state of } \mathcal{M}\}\]
- For every action step \((l, \chi) \xrightarrow{A} (l', \chi')\), there is a transition \([[l, \chi]]_\sim \xrightarrow{A} [[l', \chi']]_\sim\).
- For every time step \((l, \chi) \xrightarrow{\tau} (l, \chi')\) with \(\delta > 0\), there is a transition \([[l, \chi]]_\sim \xrightarrow{\tau} [[l, \chi']]_\sim\).
- The interpretation of the proposition symbols is:
  \[I^A(p) := \{[[l, \chi]]_\sim \in L^A \mid l \in I(p)\}\]
  \[I^A(t^\leq n) := \{[[l, \chi]]_\sim \in L^A \mid \chi(t) \leq n\}\]
  \[I^A(t^\geq n) := \{[[l, \chi]]_\sim \in L^A \mid \chi(t) \geq n\}\]

### 9.2 Translation of states, steps and runs

States are translated very easily: any state \((l, \chi)\) is translated to its corresponding equivalence class \([[l, \chi]]_\sim\).

Translate a step \((l, \chi) \xrightarrow{A^T} (l', \chi')\) to \([[l, \chi]]_\sim \xrightarrow{A^T} [[l', \chi']]_\sim\).

Runs of \(\mathcal{M}\), then, are translated to runs of \(\mathcal{M}^A\) state- and stepwise, by composing the translation of single steps. Let \((l_0, \chi_0) \xrightarrow{A^T_1} (l_1, \chi_1) \xrightarrow{A^T_2} (l_2, \chi_2) \to \cdots\) be a run of \(\mathcal{M}\). Then, the translated run is \([[l_0, \chi_0]]_\sim \xrightarrow{A^T_1} [[l_1, \chi_1]]_\sim \xrightarrow{A^T_2} [[l_2, \chi_2]]_\sim \to \cdots\). Even here, we allow Zeno runs.\(^9\)

**Lemma 2** The construction above maps runs of \(\mathcal{M}\) onto runs of \(\mathcal{M}^A\).

\(^9\)One could translate non-Zenones, using the property that a clock that isn’t reset grows above any bound. So, either a clock is reset in infinitely many steps, or it exceeds \(N\) in infinitely many steps. This could be translated to a family of “fairness sets:” for every clock \(t\), a run has to visit the set \([[l, \chi]]_\sim \mid \chi(t) = 0 \lor \chi(t) > N\) infinitely often. In addition, we have to require that there are infinitely many \(\tau\) steps in the run. A run satisfying both conditions is the translation of some non-Zeno run of the original model.
Proof: Let \((l_0, \chi_0) \xrightarrow{A_t} (l_1, \chi_1) \xrightarrow{A_t} (l_2, \chi_2) \rightarrow \cdots\) be a run of \(\mathcal{M}\). First, we have to show that the result of the construction is a run. This is clear because for every step of \(\mathcal{M}\), there is a transition in \(\mathcal{M}^A\). So, in particular, for the step \((l_{i-1}, \chi_{i-1}) \xrightarrow{A_t} (l_i, \chi_i)\), there is a transition which is taken in the step \([([l_{i-1}, \chi_{i-1}]_\sim \xrightarrow{A_t} ([l_i, \chi_i])_\sim\). Second, we have to show that every run of \(\mathcal{M}^A\) is hit. Let \([([l_0, \chi_0])_\sim \xrightarrow{A_t} [([l_1, \chi_1])_\sim \xrightarrow{A_t} [([l_2, \chi_2])_\sim \rightarrow \cdots\) be a run of \(\mathcal{M}^A\). We construct a run of \(\mathcal{M}\) which maps to it inductively. To begin, choose some arbitrary \((l_0, \chi_0') \in [([l_0, \chi_0])_\sim\). Then assume that the run has been constructed up to \((l_{i-1}, \chi'_{i-1}) \xrightarrow{A_t}_{\delta_{i-1}} (l_i, \chi'_{i-1}[C := 0])\). Look at the type of \(A_t\).

- Translating an action step back is easy because the locations of \(\mathcal{M}\) and the action set \(A_t\) are preserved by the translation. Look at the transition \(l_{i-1} = \xrightarrow{A_t} l_i\) of \(\mathcal{M}\). \(i_t(tr)\) gives us its set of clocks to be reset \(C\). Then, \((l_{i-1}, \chi'_{i-1}) \xrightarrow{A_t}_{\delta_i} (l_i, \chi'_{i-1}[C := 0])\) is the step to be added next.

- When translating a time step back to \((l_{i-1}, \chi'_{i-1}) \xrightarrow{A_t}_{\delta_i} (l_i, \chi'_i)\), we have to look for a valid value for \(\delta_i\) such that \((l_{i-1}, \chi'_{i-1} + \delta_i) \in [(l_i, \chi_i)]_\sim\). (Remember that \(l_{i-1} = l_i\).) Clearly, there are \((l, \chi) \in [(l_{i-1}, \chi_{i-1})]_\sim\) and \((l, \chi') \in [(l_i, \chi_i)]_\sim\) and \(\delta \in \mathbb{R}_+\) such that \((l, \chi) \xrightarrow{\delta} (l, \chi')\). Without loss of generality, \([(l_i, \chi_i)]_\sim\) is an immediate successor of \([(l_{i-1}, \chi_{i-1})]_\sim\), i.e., there is no \(0 < \varepsilon < \delta\) such that \((l_{i-1}, \chi_{i-1} + \varepsilon) \not\in [(l_{i-1}, \chi_{i-1})]_\sim\) \(\cup [(l_i, \chi_i)]_\sim\).

A no-waiting class is an equivalence class \([([l, \chi])_\sim\) which satisfies the condition that by waiting an arbitrarily small time \(\varepsilon > 0\), one reaches a different class: \((l, \chi + \varepsilon) \not\in [([l, \chi])_\sim\). The reader will easily convince himself that of two immediately succeeding classes, one is a no-waiting class. (Remember that a time stamp where some fraction part is 0 isn’t \(N\)-similar to one where this isn’t the case.)

Remember that \(C\) is the set of clock symbols. So, either \([(l_{i-1}, \chi_{i-1})]_\sim\) is a no-waiting class; then choose \(0 < \delta_i < \min\{(1 - \text{frac}(\chi'_{i-1}(t)) \mid t \in C)\}\). Or, \([(l_i, \chi_i)]_\sim\) is a no-waiting class; then let \(\delta_i = \min\{(1 - \text{frac}(\chi'_{i-1}(t)) \mid t \in C)\}\).

The reader will easily see that \(\delta_i\), chosen this way, has the required properties.

### 9.3 Translation of formulae

\[
\begin{align*}
\bot^A &= \bot \\
\top^A &= \top \\
p^A &= p \\
\exists x \phi^A &= \exists x \phi^A \\
\forall x \phi^A &= \forall x \phi^A \\
(t \leq n)^A &= t^n \\
(t \geq n)^A &= t^n \\
(\neg \phi)^A &= \neg(\phi^A) \\
(\phi \land \psi)^A &= \phi^A \land \psi^A \\
(\text{reset } t \text{ in } \phi)^A &= (\text{Open problem})
\end{align*}
\]
9.4 Equivalence theorem.

We can then formulate an equivalence theorem similar to theorem 1:

**Theorem 2** Let $\mathcal{M} = (L, \rightarrow, I, i, i)$ be an ATCTL model, $(l, \chi)$ one of its states and $\varphi$ an ATCTL formula without \texttt{reset} or \texttt{in} operators.

Then, $(\mathcal{M}, l, \chi) \models \varphi$ iff $(\mathcal{M}^A, [(l, \chi)]_\sim) \models \varphi^A$.

**Proof:** By induction over the formulae:

- $(\mathcal{M}, l, \chi) \models \bot \iff (\mathcal{M}^A, [(l, \chi)]_\sim) \models \bot^A$: trivial.

- $(\mathcal{M}, l, \chi) \models p$ iff $(\mathcal{M}^A, [(l, \chi)]_\sim) \models p^A$: The left-hand side holds iff $l \in I(p)$, and the right-hand side holds iff $[(l, \chi)]_\sim \in I^A(p)$. Trivial.

- $(\mathcal{M}, l, \chi) \models \neg \varphi$ iff $(\mathcal{M}^A, [(l, \chi)]_\sim) \models (\neg \varphi)^A$: By induction.

- $(\mathcal{M}, l, \chi) \models \varphi \land \psi$ iff $(\mathcal{M}^A, [(l, \chi)]_\sim) \models (\varphi \land \psi)^A$: By induction.

- $(\mathcal{M}, l, \chi) \models t \leq n$ iff $(\mathcal{M}^A, [(l, \chi)]_\sim) \models (t \leq n)^A$: The left-hand side holds iff $\chi(t) \leq n$, and the right-hand side holds iff $[(l, \chi)]_\sim \in I^A(t \leq n)$. By definition.

- $(\mathcal{M}, l, \chi) \models t \geq n$ iff $(\mathcal{M}^A, [(l, \chi)]_\sim) \models (t \geq n)^A$: Analogous.

- $(\mathcal{M}, l, \chi) \models \exists X \alpha \varphi$ iff $(\mathcal{M}^A, [(l, \chi)]_\sim) \models (\exists X \alpha \varphi)^A$: The left-hand side holds iff there is a run $(l, \chi) \xrightarrow{\alpha^A} (l', \chi') \rightarrow \cdots$ such that $A \models \alpha$ and $(\mathcal{M}, l', \chi') \models \varphi$. Then, there is a run of $\mathcal{M}^A$, namely $[(l, \chi)]_\sim \xrightarrow{\alpha^A} [(l', \chi')]_\sim \rightarrow \cdots$. It satisfies $\exists X \alpha \varphi^A$, which is the right-hand side.

- $(\mathcal{M}, l, \chi) \models \forall X \alpha \varphi$ iff $(\mathcal{M}^A, [(l, \chi)]_\sim) \models (\forall X \alpha \varphi)^A$: Analogous.

- $(\mathcal{M}, l, \chi) \models \exists (\varphi \alpha U \psi)$ holds iff there is a run $(l_0, \chi_0) \xrightarrow{A_1} (l_1, \chi_1) \xrightarrow{A_2} \cdots$ which satisfies the simple until condition. Let $(i, \delta)$ be the position used in the definition. This translates to the ACTL simple until condition: iff $(\mathcal{M}^A, [(l, \chi)]_\sim) \models \exists (\varphi \alpha U \psi)^A$:
  - $(\mathcal{M}, [(l_i, \chi_i + \delta)]_\sim) \models \varphi^A$; and
  - $(\mathcal{M}, [(l_j, \chi_j + \varepsilon)]_\sim) \models \varphi^A \lor \psi^A$, for every position $(j, \varepsilon)$ before $(i, \delta)$; and
  - $A_j^+ \text{ is } t \text{ or } A_j^- \models \alpha^A$, for every $0 < j \leq i$.

These three conditions are easily verified by induction.

- $(\mathcal{M}, l, \chi) \models \forall (\varphi \alpha U \psi)$ iff $(\mathcal{M}^A, [(l, \chi)]_\sim) \models (\forall (\varphi \alpha U \psi))^A$: Analogous.

- $(\mathcal{M}, l, \chi) \models \exists (\varphi \alpha U \beta \psi)$ iff $(\mathcal{M}^A, [(l, \chi)]_\sim) \models (\exists (\varphi \alpha U \beta \psi))^A$: The proof is similar to the simple until.

- $(\mathcal{M}, l, \chi) \models \forall (\varphi \alpha U \beta \psi)$ iff $(\mathcal{M}^A, [(l, \chi)]_\sim) \models (\forall (\varphi \alpha U \beta \psi))^A$: Analogous.
10 Comparison of the reductions

We have defined the logics ATCTL and (our variants of) ACTL and TCTL. In our framework, we can give a uniform presentation of ATCTL and the other languages: TCTL is “ATCTL without actions” and ACTL is “ATCTL without clocks.”

Complexity. The translation of formulae is, for both reductions, linear. The reduction to ACTL -^ is somewhat simpler. However, this advantage disappears when we look at the translation of models: The reduction to TCTL -^ (just add action locations) is linear in the number of transitions, but the reduction to ACTL (construct the region automaton) is exponential in the number of clocks and the largest clock constraints. (For exact numbers, see e.g. [1].) This is why we decided to implement only the translation to TCTL and let an optimised tool handle the more complicated parts.

Problems. TCTL lacks some of the operators we have defined for ATCTL: X_ and _^U_. We have found a workaround for the translation. ACTL, on the other hand, lacks the time-related operators. We have not looked for a translation of reset t in -^, because we have abandoned plans to implement this translation. The state explosion problem (because of the exponential translation) needs heavy optimisations; we prefer to reuse work already done by others for TCTL.

11 Conclusion

We have defined an extension ATCTL of CTL with actions and real time, and we have shown that the resulting logic is reducible to TCTL, so that many ATCTL formulae can be model checked by Kronos. An alternative reduction to ACTL allows us to model check all ATCTL formulae not including a clock reset action. However, this second reduction is much less efficient. By means of an example, we have shown that model checking works at least for small cases. All reductions are essentially surjective, so that it is possible to translate paths generated by a model checker back to ATCTL.

We have implemented a prototype translator from statecharts to CLKSs, so that we can undertake larger case studies, in which we intend to look at the interplay between diagram-based and textual (formal) specification and the added value that arises by combining these two approaches. For example, we want to look at mapping paths through a CLKS to paths through a collection of object-oriented statecharts. Closely related to this are the intriguing questions whether object-oriented sequence diagrams are runs through a CLKS and what consistency requirements we can derive from this between a collection of object-oriented statecharts and sequence diagrams. Finally, we want to look at possible semantics of object-oriented statecharts that are closer to the vaguely described UML semantics, by dropping the restriction that actions take no time and that we have unlimited computational resources.

Acknowledgements. This work profited from the helpful comments of Ed Brinksma, Pascal van Eck, Maarten Fokkinga, Joost-Pieter Katoen and Frits Vaandrager.
References


