Extending CTL to Specify Quantitative Temporal Requirements

Ammar Mohammed and Ulrich Furbach

Universität Koblenz-Landau, Artificial Intelligence Research Group
D-56070 Koblenz, Germany

Abstract. Computation tree logic (CTL) is expressive to specify those qualitative properties which focus on the temporal order of events. It, however, lacks to specify quantitative temporal requirements, which put time constraints on the occurrence of events. Thus, this paper presents a novel variant of temporal logic, RCTL (Region Computation Tree Logic), that extends CTL by incorporating the notation of time explicitly. To accomplish this aim, the paper uses hybrid automata as a model of computation. The specification language of RCTL allows us to express many properties in a concise and intuitive manner. To bring model checking within the scope of RCTL, the paper focuses on the specification of those properties that can be verified using reachability analysis, which is implemented in a previous work.

1 Introduction

Originally, hybrid automata [11] have been introduced as mathematical formalism for modeling and analysis of systems, whose behaviors are determined with discrete and continuous change. In a previous work [15], it has been presented an approach to model and verify multi-agent systems by means of hybrid automata. In this approach the definition of hybrid automata has been extended with events on each discrete transition. Additionally, the approach has presented a way to construct the composition of automata on-the-fly with the help of those events. Model checking, based on the reachability analysis, has been adopted in this approach. In [16, 20] a tool environment with a constraint logic programming core has been implemented as a prototype of this approach. In order to check properties, however, one needs a formal specification languages, as this language in one of the primary interfaces to the tool. One of the most widely used specification languages is temporal logic, which comes in two views : linear time (LTL) [14, 18] and branching time (CTL) [7] temporal logics. Both views allow to express the qualitative requirements of reactive systems; that is the requirements which focus on the temporal order of occurrence of events. However, these temporal logics are insufficient to specify quantitative temporal requirements, or what is so-called hard real time constraints, which put timing deadlines on the behavior of reactive systems. For example, temporal logic can specify that the *event*$_1$ is always eventually followed by *event*$_2$, but it can not reveal how long the period between the two events takes place. Therefore, temporal logics should be refined, in order to permit such types of quantitative specifications. For accomplishing this aim, there have been proposed several temporal logics

Mohammed A. and Furbach U.
Extending CTL to Specify Quantitative Temporal Requirements.
DOI: 10.5220/0003021500700079
In *Proceedings of the 8th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems (ICEIS 2010)*, page
ISBN: 978-989-8425-12-6

for both linear and computation tree logic with quantitative time (see [3, 6] for a survey). The underlying models of these logics are represented as state transition graphs annotated with time constraints, using either event-based or state based approach. In the former approach, events record the changes of states at particular points of time, whereas in the latter approach, the changes of states are recorded at each point of time. The key difference between both approaches depends on the choice of time domain. In particular, choosing the domain of time to be the set of natural numbers, gives what is so called *Discrete time* model. In this model a transition between states, represented by events, which happen only at the integer time values. The behavior of a discrete time model is described by the timed trace over a set of events that occur during the evolution of the model. Examples of related works those logics, which follow the event based model, are *Timed Propositional Temporal Logic* (TPTL) [4], and *Explicit Clock Temporal Logic* [10, 19, 17] for linar time logics, and *Real-Time Computation Tree Logic* (RTCTL) [9] for computation tree time logic. The main advantage of event based logics together with their underlying discrete time model, is their simplicity to express the quantitative properties, as they abstract lots of details of models. These logics, however, can not specify quantitative properties, which may occur within an interval of time. For example, it might be desirable to state that within some interval of time, say $10 \leq t \leq 20$, a certain property holds. This can not be expressed with events unless the boundaries of the interval coincides with occurrence some events.

On the other hand, choosing the time domain to be the set of non-negative real numbers gives what is so called *continuous/dense* model, in which states have to be recorded at each time point. Therefore, the change of states is represented by letting the time to pass between one state to another. Examples of related works of dense logics are *Metric Temporal logic* (MTL) [13] and *Metric Interval Temporal Logic* (MITL)[2] for linear time with dense semantics, and *Timed Computation Tree Logic* (TCTL) [1], and *Integrator Computation Tree Logic* (ICTL) [5] for computation tree time with dense semantics. These logics are powerful and expressive to specify quantitative properties, as they record the state of the model at each point of time. They cope with the limitations of event based logics mentioned previously. They, however, lack to express properties that entirely depend on events in models directly, despite the fact that events are used to communicate and synchronize concurrent parts of a model –i.e. are used to construct the parallel composition of a model. To specify and hence verify a property based on the occurrence of events, an extra work has to be done on the specification of this property by converting its event based specification into a suitable state based representation like what Uppaal [8] and Hytech [12] are doing. For example, to specify and check that it is always the case in a model M that $event_1$ is followed by $event_2$ within t time unit– this property is called a bounded response property–, a traditional solution to this is to translate this specification to a testing transition model A, and then checking whether the parallel composition of A and M can reach a designated state of A.

To this end, this paper proposes Region Computation Tree Logic (RCTL) that extends CTL by incorporating notation of time on states and events. Hybrid automata will be used as the underlying model of RCTL. In particular, formulas of RCTL are interpreted on tree of regions – i.e. the set of all possible runs – generated from the transition system of hybrid automata [15]. To plug the specification of properties into our model checking approach presented in [16, 15, 20], the paper focuses on a fragment of this logic that specifies those properties which can be verified using reachability analysis. The main novelty of RCTL is that it encompass the expressive power of event ans state based approaches. In contrast to other model checking tools, like Uppaal [8] and Hytech [12], the verification of requirements are straightforward checked without unnecessary translation to suitable specifications.

The structure of the paper is as follows. In Sec.2 we introduce the syntax and the semantics of the region computation tree logic will be explained. In Section 3, the specification of several qualitative and quantitative properties in RCTL is given. Moreover, the paper encodes these properties them into suitable constraint logic program quires, and hence can be automatically verified by means of Reachability analysis. Finally, Section 4 concludes the paper.

2 Region Computation Tree Logic RCTL

This section primarily focuses on the definition of the region computation tree logic (RCTL), which extends the qualitative temporal logic of CTL with time on states, events, and constraints of variables. Thus, RCTL brings together, in the same level of specifications, qualitative and quantitative requirements. The formulas of RCTL are interpreted over the possible regions resulted from the run of hybrid automata.

A hybrid automaton H consists of a finite set of control locations Q, a finite set of real-valued variables $\mathbb{X} = \{x_1, x_2, ..., x_n\}$, a finte set of events, and functions that determine the continuous evolution of the variables w.r.t. some constraints inside Q, the jump from locations, and how $\mathbb{X}$ are updated. Additionally, a hybrid automaton contains an initial location and valuations of the variable which determine the starting state of the automaton – Due to the space limitation, we assume the reader is familiar with the syntax and semantics of hybrid automata. More details about this can be find in [15].

Informally speaking, a state $\sigma_i = \langle q_i, v_t, t\rangle$ of a hybrid automaton describes the valuations v_t of the variables $\mathbb{X}$ at time instance t in a control location q. A run $\rho = \sigma_0\sigma_1\sigma_2, \ldots$ of a hybrid automaton is defined in terms of an alternating sequence of states, which they are related to each others using a continuous or a discrete step. If a state σ_1 relates to σ_2 with a discrete setp, then an event should be fired. The continuous change of states in a run generates an infinite number of states. It follows that state-space exploration techniques require a symbolic representation way in order to represent the set of states in an appropriate way. A good way to represent this is to use mathematical intervals called *regions*, which are able to capture all possible continuous states. Thus, the run of a hybrid automaton can be rephrased in terms of reached regions, where the change from one region to another is fired using a discrete step. which are able to capture all possible continuous states. A region Γ is written as $\Gamma = \langle q, V, T\rangle$ captures the possible states that can be reached using continuous transitions in each location $q \in Q$.Therefore, T represents the continuous reached time. Additionally, a region captures the continuous values for each variable $x_i \in X$. These continuous values can be represented as an interval V of real values. Thus, a run of a hybrid automaton can be rephrased in terms of reached regions as $\rho_H = \Gamma_0\Gamma_1, \ldots$ where the change from one region to another is fired using a discrete step – written as $\Gamma_i \xrightarrow[t]{a} \Gamma_{i+1}$.

Regions are the key essence of RCTL, such that RCTL can be viewed as a state based quantitative temporal logics in a sense that regions capture the changes of states, and as event based quantitative temporal logics in a sense that events mark the instantaneous exist from region to another. Thus, RCTL brings together, in the same framework, the advantages of both approaches. In the following we show the syntax and semantics of RCTL.

Let $\mathbb{X}$ be a set of real variables, $\Phi(\mathbb{X})$ be a set of a linear constraints with free variables from $\mathbb{X}$, $\mathbb{Q}$ be a set of proposition denotes a set of locations , and *Event* be a set of propositions denotes the occurrence of events.

Definition 1 (Syntax). *The formula Ψ of RCTL are inductively defined as*

$$\Psi ::= p \mid a \mid \phi \mid \neg\Psi \mid \Psi_1 \wedge \Psi_2 \mid t.\Psi \mid \exists(\Psi_1 U \Psi_2) \mid \forall(\Psi_1 U \Psi_2)$$

for $t \in \mathbb{R}^{\geq 0}$, $p \in \mathbb{Q}$, $a \in Event$, and $\phi \in \Phi(\mathbb{X})$

Let a region Γ Conventionally takes the form $\Gamma = (q, V, T)$, with q is its location, and V and T are the interval of continuous valuation of variables with their respective time at which the region is admissible. A sub-region $\beta \subseteq \Gamma$, and $\beta \neq \emptyset$ means that $\beta = (q, V', T')$ with $T' \subseteq T$ and $V' \subseteq V$. A state $\sigma \in \Gamma$ means that $\sigma = (q, v_t, t)$, with $v_t \in V$ and $t \in T$. σ satisfies a constraint $\phi \in \Phi(\mathbb{X})$, written as $\sigma \models_v \varphi$, iff $v_t \models \varphi$ –i.e., the valuation v_t satisfies the constraint φ . In the following, we show the semantics of RCTL formulas on the set of all possible runs Π_H

Definition 2 (Semantics). *Let Ψ is a RCTL formula, and Π_H be the possible runs of a hybrid automaton with a region $\Gamma = (q, V, T) \in \Pi_H$. The satisfaction relation $\langle \Pi_H, \Gamma \rangle \models_T \Psi$, which means that Ψ is satisfied in the region Γ within a time interval T, is defined inductively as follows*

- $\langle \Pi_H, \Gamma \rangle \underset{T}{\models} p$ *iff* $p = q$.
- $\langle \Pi_H, \Gamma \rangle \underset{T}{\models} a$ *iff there is* $t' \in T$ *with* $\Gamma \xrightarrow[t']{a} \Gamma'$.
- $\langle \Pi_H, \Gamma \rangle \underset{T}{\models} \phi$ *iff there is* $\beta \subseteq \Gamma$, *for each* $\sigma_k \in \beta, \sigma_k \models_v \phi$.
- $\langle \Pi_H, \Gamma \rangle \underset{T}{\models} \neg\Psi$ *iff* $\langle \Pi_H, \Gamma \rangle \underset{T}{\not\models} \Psi$.
- $\langle \Pi_H, \Gamma \rangle \underset{T}{\models} t.\Psi$ *iff* $\langle \Pi_H, \Gamma \rangle \underset{T:=t}{\models} \Psi$.
- $\langle \Pi_H, \Gamma \rangle \underset{T}{\models} \Psi_1 \wedge \Psi_2$ *iff* $\langle \Pi_H, \Gamma \rangle \underset{T}{\models} \Psi_1$ *and* $(\Pi_H, \Gamma) \underset{T}{\models} \Psi_2$.
- $\langle \Pi_H, \Gamma \rangle \underset{T}{\models} \exists(\Psi_1 U \Psi_2)$ *iff there is a run* $\Pi \in \Pi_H, \Pi = \Gamma_0, \Gamma_1, \cdots$, *with* $\Gamma = \Gamma_0$, *for some* $j \geq 0$, $\langle \Pi_H, \Gamma_j \rangle \underset{T_j}{\models} \Psi_2$, *and* $\langle \Pi_H, \Gamma_k \rangle \underset{T_k}{\models} \Psi_1$ *for* $0 \leq k < j$.
- $\langle \Pi_H, \Gamma \rangle \underset{T}{\models} \forall(\Psi_1 U \Psi_2)$ *iff for every run* $\Pi \in \Pi_H, \Pi = \Gamma_0, \Gamma_1, \cdots$, *with* $\Gamma = \Gamma_0$, *for some* $j \geq 0$, $\langle \Pi_H, \Gamma_j \rangle \underset{T_j}{\models} \Psi_2$, *and* $\langle \Pi_H, \Gamma_j \rangle \underset{T_k}{\models} \Psi_1$ *for* $0 \leq k < j$.

In the previous semantics, the quantifiers $\forall$, and $\exists$ are called paths quantifiers. The formula $t.\Psi$ gives the current time, at which the formula Ψ occurs. $\langle \Pi_H, \Gamma \rangle \underset{T:=t}{\models} \Psi$ means that the formula Ψ is satisfied in the region Γ when the time T is restricted to the time point t. In case Ψ represents a proposition of an event, then $t.\Psi$ gives the time at which the event has occurred. This can be used to specify various quantitative properties, like time bound response properties, as we will see in what follows. However, if Ψ represents a constraint formula, then $t.\Psi$ evaluates the time interval at which the constraint Ψ is satisfied. This allows to specify quantitative properties, which could not be specified using events.

In addition to the definition of formulas, There are standard abbreviations in RCTL similar to CTL. For example, Eventually: $\Diamond\Psi = true\, U \Psi$, and Always: $\Box\Psi = \neg\Diamond\neg\Psi$. Like $\Box$ and $\Diamond$, both $\forall$ and $\exists$ are dual. Thus, the formulas $\neg(\forall\Diamond\Psi)$ and $\exists\Box\neg\Psi$, for example, are semantically equivalent.

Definition 3 (Satisfiability). *Let H be hybrid automaton with Π_H as its possible runs. We say that H satisfies the RCTL formula Ψ, written as $H \models \Psi$, iff $(\Pi_H, \Gamma_0) \models \Psi$, where Γ_0 is the initial region of Π_H.*

3 Model Checking as Reachability

For the purpose of verification by means of model checking, we need to describe the properties. Generally, the qualitative properties are often classified into reachability, safety and liveness properties. However, when the time becomes a critical factor to react in the environment, then the concept of safety and liveness properties should be refined. In the following these types of properties will be reviewed with their specifications by means of RCTL. For the purpose of model checking, these properties will be encoded into suitable queries in Constraint logic program (CLP) following our CLP model presented in [15]. However, in order to put model checking within our framework, we will concentrate only on the reachability requirements. Indeed, many properties of interest can be specified as a form of reachability, as we will see in the sequel. Therefore, we will start with specifying of reachability.

3.1 Reachability Properties

The reachability of a property Ψ means starting from the initial state of an intial region, there is a region along some run in which Ψ is satisfiable. This can be specified in RCTL as follows:

$$init \rightarrow \exists\Diamond\Psi$$

where *init* is the predicate characterizing the set of initial states and it expresses that the run to be considered are those that start from the initial state . In terms of the CLP model, the reachability of a certain region that satisfies the formula Ψ is done by performing forward reachability analysis from the system's initial state, and then checking whether the conjunction of Ψ with the possible reached regions is satisfied. For example assuming *init* has been assigned to the set of initial state, the following is the CLP query to check the safety requirements.

```
%%% reachable(+init,-Region).
%%% perform reachability starting from the intial states
```

```
?- reachable(init,Reached),
      member(Ψ,Reached),ϕ.
```

In this query, `reachable` is a predicate which takes the initial states of a model and returns the possible reached state in a list of regions, `member` is the standard Prolog predicate. ϕ is a constraint formula over the variables, which might appear in the formula Ψ. To demonstrate the previous specification in a concrete example, we take a train gate controller example described in [15]. Assume that one wants to check the possibility of reaching a region whose a state satisfy that the *train* is at *near* within distance less than 10 *meters*, and the *gate* is *closed*. First the initial states is given by

$$init : train.far \wedge gate.open \wedge controller.idel \wedge x = 1000 \wedge g = 0 \wedge z = 0.$$

The intended formula is specifed as

$$init \rightarrow \neg\exists\Diamond(x \leq 10 \wedge train.near \wedge gate.closed)$$

Now, in CLP the previous formula is verified by asking the following query. The successful answer to this query gives insight for the satisfaction of the reachability of the specified formula.

```
?-reachable((far,[1000]),(open,[0]),(idle,[0]),Reached),
      member((near,close,_,Time,_,X,),Reached), X $=< 10.
```

It is often that in certain cases we may be interested in the reachability of certain state either before or after a time deadline has expired, which we call *Time bounded reachability*. For example, the possibility of a formula Ψ to be reached within the bounded time α is specified in RCTL as

$$init \rightarrow \exists\Diamond\ (t.\Psi \wedge t \leq \alpha)$$

Demonstrating this on the previous example with $\alpha = 19$, in the following the query, which check the reachability of the previous example within 19 unite of time.

```
?-reachable((far,[1000]),(open,[0]),(idle,[0]),Reached),
    member((near,close,_,Time,_,X,),Reached),
    X $=< 10, Time $=<19.
```

3.2 Safety as Reachability

A safety property states that *something bad must never happen*. The bad thing represents a critical property that should never occur. Let Ψ represents this critical property, then the safety property is specified using RCTL as:

$$init :\rightarrow \forall\Box\neg\Psi.$$

Generally, a safety property can be violated within bounded time, which means that the exhibition of the previous formula by a single state within a region, suffices to show that the safety property does not hold. For this reason, safety property can be reduced to reachability property as the following

$$init :\rightarrow \neg\exists\Diamond\Psi.$$

To illustrate the safety property in the train gate example, assuming one wants to check that the state, where the train is near at distance $X = 0$ and the gate is open, is a forbidden state. This safety requirement can be specified as

$$init \rightarrow \neg\exists\Diamond(x = 0 \wedge train.near \wedge gate.open)$$

This formula asserts that during the run of the system, starting from the initial state, there is no reached state where the train is near at distance $x = 0$ and the gate is at open state. It turns out, to check the safety property, one checks the unreachability of the following query:

```
?-reachable((far,[1000]),(open,[0]),(idle,[0]),Reached),
      member((open,down,_,Time,_,X,),Reached), X $= 0.
```

3.3 Quantitative Properties

We showed that safety properties can be reduced to reachability problem. The safety properties assert what may or may not occur, but do not require that anything ever does happen. For example, in the train gate example, closing the gate permanently can maintain the safety of the system, but it is unacceptable for the waiting cars or pedestrians in the front of the gate. For this reason, the liveness property is needed to specify such requirements, which asserts that some property of interest will always eventually occur. In other words, there exists a time point in which the system is always in a good state. It should be noted these type of properties can not be falsified in bounded time. We can not say that the liveness property is violated because of occurring some state does not say how long it will take for this state to occur. For this reason, these types of properties are not strong enough in the context of specifying quantitative properties. Here one would like to see a time bound when the good state occurs. This brings the next kind of properties.

Bounded Response Properties. A bounded response property is one of the most important classes of quantitative timing requirements, which can be used to specify many important application. It asserts that something will happen within a certain amount of time. A typical application of bounded response property is the specification of worst case performance; that is the specification of an upper bound α on the termination of a system S: if started at time t, then S is guaranteed to reach a final state no later than $\alpha + t$ unit time. In the train gate example, a desired property is to specify that once the approach of a train is detected, the gate needs to be closed within a certain time bound in order to halt car and pedestrian traffic before the train reaches the crossing the intersection. Formally, the bounded response property can be specified in RCTL as:

$$init \rightarrow \forall\Box(t_1.event \rightarrow \forall\Diamond(t_2.event_2 \wedge t_2 \leq \alpha + t_1)).$$

The previous formula states that whenever there is a request $event_1$ occurs at time t_1, then it is followed by a response $event_2$, at time t_2, such that t_2 is at most $\alpha + t_1$.

It should be mentioned that this property can be falsified within time bound, therefore this property can be specified as a kind of safety requirement, and hence can be represented as reachability. For this reason, proving the previous property means proving that it is not possible to reach a state where the $event_2$ is not reached from $event_1$, within $t_2 \leq \alpha + t_1$. In other words, starting from $event_1$, finding a reachable state satisfies $event_2$, within α time bound, is sufficient to check the reachability of the property. In terms of the CLP, this previous can be encoded into the following steps. First, getting all possible reachable states from $event_1$ within $t_1 + \alpha$ as L, then check that reachability of $event_2$ has not occurred. A positive answer gives a negative answer to the original problem, and vis versa. The following is the CLP query encode, the previous specification

```
?- reachable($\Psi_0$,Reached),
   reached_from(L,$event_1$,Reached),
   reached_within(Target, $\alpha$,L),
   \+ memeber((_,..,_,$event_2$),Target)
```

We should emphasize that the traditional way to verify this type of properties in real time system tools, like UPPAAL [8] and Hytech [12], is to translate this property to a suitable state based specification. In Hytech, for example, to specify that the $event_2$ is a response to $event_1$ within α time unit, one has to augment the model under consideration by an automaton A, whose *idle*, *wait*, and *violate* as its control locations and t as its integrator. Initially, the control location in the *idle*. When a trigger $event_1$ occurs, control pass to *wait* location and the integrator t is reset. The response $event_2$ causes the control to return to the *idle* location. The location *violate* is only enabled when $t \geq \alpha$. Now, with the parallel composition of the original model with the automaton A, the specification of bounded response property can be specified as the un-reachability of the location *violate*. As we said , the reason behind this translation is that there is no direct use of events in the model. The use of events are limited to construct only the parallel composition of automata. In contrast to our presented approach, the direct use of events with the model, allows us to avoid this translation process. This shows that RCTL is more expressiveness, particularly in our setting, than any other quantitative temporal logics.

Bounded invariance Properties. Like the bounded response property, bounded invariance property is one of the most important classes of quantitative timing requirements. It asserts that once an event has been triggered, a certain condition will hold continuously for a certain amount of time. It is often used to specify that something will not happen for a certain amount of time. Formally, specifying that a certain property hold continuously for a certain amount of time in RCTL is like the following

$$init \rightarrow \forall\Box(t_1.event \rightarrow \forall\Box(t_2.\Psi \wedge t_2 \leq \alpha + t_1)).$$

where α is the duration at which the formula Ψ must be continuously hold. An example of such type of properties is to specify that whenever the train approaches the gate, the distance of the train is always greeter than 100 for the duration of 20 time unit. The property $\Psi = X \geq 100$ in this case represents the distance of the train, and *app* is the triggered event. Again, the bounded invariance property can be checked as a safety property. Starting from time t_1, finding a non-reachable violating state for the formula Ψ, within α time bound, is sufficient to check the reachability of the property. Thus, we specify this with CLP as the following

```
?- reachable(Ψ0,Reached),
   reached_from(L,event1,Reached),
   reached_within(Target, α,L),
   memeber((_,..,X,_,Target), X$<100.
```

4 Conclusions

Due to the lack of qualitative temporal logics to specify quantitative properties. We showed in this paper how to extend the qualitative logic CTL to the quantitative logic RCTL by adding time notation on linear constraints, states, and events. Hybrid automata have been used as the interpretation model of RCTL. The formulas of RCTL are interpreted on the possible regions produced form the run of hybrid automata. With regions, RCTL combined the expressive power of both state based and event based quantitative temporal logics, which have been proposed already to extend the qualitative temporal logics. RCTL allows to express many properties in a concise and intuitive manner. To bring model checking within the scope of RCTL, we concentrated on the specification of those properties that can be verified using reachability analysis. Furthermore, the paper showed how to encode these properties into suitable quires implemented with constraints logic programming.

References

1. R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Inf. Comput.*, 104(1):2–34, 1993.
2. R. Alur, T. Feder, and T. A. Henzinger. The benefits of relaxing punctuality. *J. ACM*, 43(1):116–146, 1996.
3. R. Alur and T. Henzinger. Logics and models of real time: A survey. *Real Time: Theory in Practice, Lecture Notes in Computer Science*, 600:74–106, 1992.
4. R. Alur and T. Henzinger. A really temporal logic. *Journal of the ACM (JACM)*, 41(1):203, 1994.
5. R. Alur, T. A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22(3):181–201, 1996.
6. P. Bellini, R. Mattolini, and P. Nesi. Temporal logics for real-time system specification. *ACM Comput. Surv.*, 32(1):12–42, 2000.
7. M. Ben-Ari, A. Pnueli, and Z. Manna. The temporal logic of branching time. *Acta Informatica*, 20(3):207–226, 1983.
8. J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi. Uppaal—a tool suite for automatic verification of real-time systems. In *Proceedings of the DIMACS/SYCON workshop on Hybrid systems III : verification and control*, pages 232–243, Secaucus, NJ, USA, 1996. Springer-Verlag New York, Inc.

9. E. A. Emerson, A. K. Mok, A. P. Sistla, and J. Srinivasan. Quantitative temporal reasoning. *Real-Time Syst.*, 4(4):331–352, 1992.
10. E. Harel, O. Lichtenstein, and A. Pnueli. Explicit clock temporal logic. In *Proceedings, Fifth Annual IEEE Symposium on Logic in Computer Science, 4-7 June 1990, Philadelphia, Pennsylvania, USA*, pages 402–413. IEEE Computer Society, 1990.
11. T. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual Symposium on Logic in Computer Science*, pages 278–292, New Brunswick, NJ, 1996. IEEE Computer Society Press.
12. T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. In *CAV '97: Proceedings of the 9th International Conference on Computer Aided Verification*, pages 460–463, London, UK, 1997. Springer-Verlag.
13. R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.
14. Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems: Specification*. Springer, 1992.
15. A. Mohammed and U. Furbach. Multi-agent systems:modeling and verification using hybrid automata. In J.-P. B. Lars Braubach and J. Thangarajah, editors, *Post-Proceedings of 7th International Workshop on Programming Multi-Agent Systems at 8th International Joint Conference on Autonomous Agents and Multi-Agent Systems*, LNAI 5919, pages 49–66. Springer, Berlin, Heidelberg, 2010.
16. A. Mohammed and C. Schwarz. Hieromate: A graphical tool for specification and verification of hierarchical hybrid automata. In M. H. B. Mertsching and Z. Aziz, editors, *KI 2009: Advances in Artificial Intelligence, Proceedings of the 32nd German Conference on Artificial Intelligence*, LNAI 5803, pages 695–702. Springer, 2009.
17. J. Ostroff and W. Wonham. A framework for real-time discrete event control. *IEEE Transactions on Automatic Control*, 35(4):386–397, 1990.
18. A. Pnueli. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57, 1977.
19. A. Pnueli and E. Harel. Applications of temporal logic to the specification of real-time systems. In *Systems, Proceedings of a Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 84–98, London, UK, 1988. Springer-Verlag.
20. C. Schwarz, A. Mohammed, and F. Stolzenburg. A tool environment for specifying and verifying multi-agent systems. In J. Filipe, A. Fred, and B. Sharp, editors, *Proceedings of the 2nd International Conference on Agents and Artificial Intelligence*, volume 2, pages 323–326. INSTICC Press, 2010.