First International Conference
Modelling and Development of Intelligent Systems
Sibiu - Romania, 22-25 October, 2009

An Algebraic Specification for CTL with Time Constraints

Laura Florentina Cacovean

Abstract

In this paper we present the algebraic methodology and its utilization for CTL model checker specifications. This allows the possibility of automatic generation of the model checking algorithms for temporal logics into sets of algebraic specifications. We use ANTLRWorks for implement the all macro-operations of CTL model checker. In this paper we extend the CTL model checker and we give the implementation of time constraints of algebraic model checker specification. Next we give an algebraic specification of time of CTL model checker and a case study which proof that specification and our proposed model is correct construct.

1 Introduction

The model checkers are tools which can be used to verify that a given system satisfies a given temporal logic formula. The model is a directed graph where the nodes represent the states of the system and the edges represents the state transitions. The nodes and the edges can be labelled with atomic propositions what describe the states and the transitions of the system. In order to be verified by a given model, a property is written as a temporal logic formula across the labelled propositions from the model. A model checker is an algorithm that determines the states of a model that satisfy a temporal logic formula.

The algebraic methodology and its utilization in developing instruments for model checker specifications as maps in form $C{:}L_s \rightarrow L_t$ [4], where L_s is the source language of temporal logic, L_t is the target language representing sets of states of the model M and $C(f \in L_s)=\{s \in M|s f\}$, where $\vDash$ is satisfaction relation. This allows the possibility of automatic generation of the model checking algorithms for temporal logics into algebraic specifications sets. Extensibility and flexibility of algebraic methodology show how the model checkers for various temporal logics can be generated from algebraic specification. In paper [1] we showed how this algebraic context can be used to the specification of *CTL* (Computation Tree Logic) model checker.

2 CTL model checker

CTL model checker is branching-time logic, meaning that its formulas are interpreted over all paths beginning in a given state of the Kripke structure. A Kripke model M over AP is a triple *M=(S, Rel,*

$P:AP \rightarrow 2^S$) where S is a finite set of states, $Rel \subseteq S \times S$ is a transition relation, $P:S \rightarrow 2^{AP}$ is a function that assigns each state with a set of atomic propositions.

A CTL formula is evaluated on a Kripke model M. A path in M from a state s is an infinite sequence of states from S, denoted in the following with $\pi = [s_0, s_1, \dots, s_{i-1}, s_i, s_{i+1}, \dots]$ such that $s_0 = s$ and $(s_i, s_{i+1}) \in Rel$ holds for all $i \geq 0$. We write $(s_i, s_{i+1}) \subseteq \pi$ and $s_i \in \pi$. If we express a *path* as $\pi = [s_0, s_1, \dots, s_i, \dots, s_j, \dots]$, for $i < j$, we say that s_i is a state earlier than s_j in π as $s_i < s_j$.

Definition 1 *(Syntax Definition of CTL model checker [3]) A CTL has the following syntax given in Backus-Naur form:*

$$f :: \top | \bot | p | (\neg f_1) | f_1 \wedge f_2 | f_1 \vee f_2 | f_1 \subset f_2 | AX f_1 | EX f_1 | AG f_1 | EG f_1 | AF f_1 | EF f_1 | A[f_1 U f_2] \mid E[f_1 U f_2] \qquad (1)$$

where $\forall p \in AP$.

Semantic definition of CTL model checker is provided in [3]. Let M=(S, Rel, $P:AP \rightarrow 2^S$) be a Kripke model for CTL. Given any s in S, in [3] is defined whether a CTL formula f holds in state s. This is denoted this by (M, s) ⊨ f. The satisfaction relation ⊨ is defined by structural induction on fourteen CTL formulas [3].

Many model checking algorithms were developed for different temporal logics [12], thus in this paper is presented a simple universal algorithm based on the algorithm of homeomorphism computation which is used in an algebraic compiler [4]. The generic homeomorphism algorithm is customized by a set of specifications to construct a model checker, implemented as an algebraic compiler $C:L_s \rightarrow L_t$. The specifications consist from a finite set of rules in which each of the rules defines the syntax of some classes constructed in the source language and also the semantic values of these constructs as expressions in the syntax of the target language.

3 Algebraic Structure of CTL

In an algebraic compiler $C:L_s \rightarrow L_t$ the source and the target language used are defined using heterogeneous Σ-algebras and Σ-homeomorphisms [4,5]. The operator scheme of a *Σ-algebras* is a tuple $\Sigma = \langle S, O, \sigma \rangle$ where S is a finite set of states, O is it a finite set of operator names, and $\sigma : O \rightarrow S^* \times S$ is a function which defines the signature of the operators. These signatures are denoted as $\sigma(o) = s_1 \times s_2 \times \dots \dots \times s_n \rightarrow s$, where s, $s_i \in S$, $1 \leq i \leq n$. A Σ-algebra is a tuple $A_\Sigma = \langle \{As\}_{s \in S}, Op(O) \rangle$, where $\{A_s\}_{s \in S}$ is a family of non-empty sets indexed by the states S of Σ, called the carrier sets of the algebra, and $Op(O)$ is a set of operations across the sets in $\{A_s\}_{s \in S}$ such that for each $o \in O$ with signature $\sigma(o) = s_1 \times s_2 \times \dots \times s_n \rightarrow s$, $Op(o)$ there is a function $Op(o) : A_{s1} \times A_{s2} \times \dots \times A_{sn} \rightarrow A_s$. In the following $Op(o)$ is identified with O. A Σ- algebra is a tuple defined through $L = \langle Sem, Syn, \mathcal{L} : Sem \rightarrow Syn \rangle$ [4], where *Sem* is a Σ- algebra called the *semantic language*, *Syn* is a Σ- word or a term algebra called the *syntax language* and $\mathcal{L}$ is a partial mapping called the language learning function.

In order to define a *CTL* model as a Σ-language, shall define an operator scheme Σ_{ctl} as the tuple $\langle S_{ctl}, O_{ctl}, \sigma_{ctl} \rangle$ where the states $S_{ctl} = \{F\}$ are mapping the formulas f [4] and $O_{ctl} = \{true, false, \neg, \wedge, \vee, \rightarrow, AX, EX, AU, EU, EF, AF, EG, AG\}$ [1]. In this paper it imports us to know just until operation, *AU* and *EU*. Therefore we show the description in Sem_{ctl} algebra of σ_{ctl} just for *AU* formulas.

Operator	*Description in* Sem_{ctl} *algebra*
$\sigma_{ctl}(U) : F \times F \rightarrow F$	*if* $f1, f2 \in Sem_F$ *then* $A\,[f1\,U\,f2] \in Sem_F$

Table 1 Operator scheme Σ_{ctl} in Sem_{ctl} algebra

CTL can be defined as the Σ_{ctl}-language given in the form $L_{ctl}=\langle Sem_{ctl}, Syn_{ctl}, \mathcal{L}_{ctl}:Sem_{ctl}\rightarrow Syn_{ctl}\rangle$. Syn_{ctl} is the word algebra of the operator scheme Σ_{ctl} generated by the operations from O_{ctl} and a finite sets of variables, denoting atomic propositions, *AP*. Sem_{ctl} represents *CTL* semantic algebra defined across the satisfied sets of *CTL* formulas for a given model *M*. $\mathcal{L}_{ctl}$ is a mapping which associates satisfiability sets in Sem_{ctl} from the *CTL* expressions in Syn_{ctl} that they satisfy and ε_{ctl} is a homeomorphism that evaluates *CTL* expressions in Syn_{ctl} to their satisfied sets in Sem_{ctl}. Although the rules for forming *CTL* formulas are independent of any model, the signification of the resulting formulas is dependent upon a given model. Thus in the algebraic definition of *CTL*, Syn_{ctl} is independent from any model while Sem_{ctl} is dependent on the given model *M*. The word algebra Syn_{ctl} is unique into homeomorphism in the class of algebras with operator scheme Σ_{ctl}. This has as the carrier set Syn_F, the collection of *CTL* formulas. This is called terms or words, created by the juxtaposition of variables from *AP* and the operator symbols from O_{ctl} according to the rules shown in Table 1.

3.1 Algebraic structure of model

The CTL model checker algorithm maps the CTL formulas in the syntax algebra Syn_{ctl}. In order to understand these mappings in [4] is structured the model $M=\langle S,Rel,P:AP\rightarrow 2^S\rangle$ as a Σ-language whose syntax algebraic contains the sets of the expressions and whose semantics algebra contains the sets of 2^S. The operator scheme for this language is $\Sigma_{sets}=\langle S_{sets},O_{sets},\sigma_{sets}\rangle$ where $S_{sets}=\{S,B\}$, *S* is the sort for sets, *B* is the sort for the boolean values, $O_{sets}=\{S, \emptyset, C, \cap, \cup, Imply, Urm_{all}, Urm_{some}, LFP_{all}, LFP_{some}, All_{global}, All_{future}, Exist_{global}, Exist_{future}\}$ and σ_{sets} is shown in Table 1. In [1] we used different symbols to denote the operations, because the operators in Sem_{ctl} operate on sets and the operators from Syn_{ctl} operate on terms. The operators from Sem_{ctl} corresponding to the names $\{true, false, \neg, \wedge, \vee, \rightarrow, AX, EX, AU, EU, EF, AF, EG, AG\}$ in O_{ctl} of Σ_{ctl} are respectively named $\{S, \emptyset, C, \cap, \cup, Imply, Urm_{all}, Urm_{some}, LFP_{all}, LFP_{some}, All_{global}, All_{future}, Exist_{global}, Exist_{future}\}$. The model *M* defined as Σ_{sets}-language $L_M=\langle Sem_{sets}, Syn_{sets}, L_{sets}: Sem_{sets}\rightarrow Syn_{sets}\rangle$, where ε_{sets}: $Syn_{sets}\rightarrow Sem_{sets}$ evaluates set expressions to the sets they represent. In this language, Sem_{sets} is the semantic algebra with the carrier sets $Sem_S=2^S$ and $Sem_B=\{true, false\}$. The operators in the algebra and their signatures as defined by σ_{sets} are shown in [1]. We retain that semantic Sem_{sets} and Sem_{ctl} have the carrier sets in the relation $Sem_F\subseteq Sem_S$. This allows due to similarity to show in the scheme all elements of carrier sets of Sem_{ctl} through their occurrences in the carrier sets Sem_{sets}.

3.2 Algebraic description of CTL model checker

A *CTL* model checker defined as an algebraic compiler $C:L_{ctl}\rightarrow L_M$ by pair of embedding morphisms $\langle T_C,H_C\rangle$. T_C: $Syn_{ctl}\rightarrow Syn_{sets}$ maps *CTL* formulas from word algebra Syn_{ctl} to set expressions in Syn_{sets}, which evaluate to the satisfiability of sets of the *CTL* formulas, $H_C : Sem_{ctl}\rightarrow Sem_{sets}$, maps sets in Sem_{ctl} by the identity mapping to sets in Sem_{sets} and thus is constructed using the following approach [4,10]:

1. Associate each operation o_{ctl} from algebra Sem_{ctl} with a set expression $d(o_{ctl})$ from algebra Syn_{sets} with the property $H_C(o_{ctl}(s_1, \dots ,s_n))=\varepsilon_{sets}(d(o_{ctl})(d_{ctl}(s_1), \dots, d_{ctl}(s_n)))$.

$o\in Sem_{ctl}$	$D_{ctl}(o)\in Syn_{ctl}$
$LFP_{all}(t_1,t_2)$	$Z:=\emptyset;\ Z':=d_{ctl}(t_2);\ Z'':=d_{ctl}(t_1);$ *while* $(Z\neq Z')$ *do* $Z:=Z';\ Z':=Z'\cup(Z''\cap\{s\in S\mid succ(s)\subseteq Z'\});$ *end while* $d_{ctl}(LFP_{all}(t_1,t_2)):=Z';$

Table 2 The construction of *d* over the generators and operations Sem_{ctl} and Syn_{ctl}

2. The set of expressions from the second column of table 2 defines the operations of an algebra Syn'_{sets} which is similar to the operations from Sem_{ctl} and from Syn_{ctl}.
3. Syn'_{sets} is a sub-algebra of Syn_{sets}, the embedding $T_C : Syn_{ctl} \rightarrow Syn'_{sets}$ is constructed by the composition of T'_C and injection function $I : Syn'_{sets} \rightarrow Syn_{sets}$, given by $T_C=T'_C \circ I$.

The morphisms T_C and H_C thus constructed make the diagram commutative. Commutatively assures the fact that T_C keeps the meaning of the formulas from Syn_{ctl} when mapping them to set expressions in Syn_{sets}. The diagonal mapping $Dctl:Sem_{ctl}\rightarrow Syn_{sets}$ is generated by d_{ctl} defined in table 2 and shows the translation process performed by T_C using *derived operations* [4].

4 Algebraic implementation of CTL

Construction of T_C in the Subsection 3.2 can be entered into an algorithm which implements the *CTL* model checker. This algorithm is universal in the sense that being given operator scheme Σ_{ctl} and a model *M*, the model checker L_{ctl} is automatically generate from the specifications of $\langle\Sigma_{ctl}, D_{ctl}\rangle$. This specification is obtained by associating each operation $o \in O_{ctl}$ with an derived operation $d_{ctl}(o) \in D_{ctl}$. To define derived operations that implement the operations of Σ_{ctl}, from the Syn_{sets} algebra, we use meta-variables that take as values the set expressions of the carrier sets of Syn_{sets}. For each operation $o \in O_{ctl}$ such that $\sigma_{ctl}(o)=s_1\times\ldots\times s_n\rightarrow s$, $d_{ctl}(o)$ takes as the formal parameters the meta-variables denoted by $@_i$, $1\leq i\leq 2$, where $@_i$ denotes the set expression associated with *i*-th argument of $d_{ctl}(o)$; the meta-variable $@_0$ is used to denote the resulting set expression, as example $@_0=d_{ctl}(o)(@_1,\ldots, @_n)$ [1,4].

The *CTL* rules set which are directly specified in the algebra Sin_{ctl} can be ambiguous, therefore it is necessary that the set *F* from Sin_{ctl} to be divided in the non-terminal symbols denote with *Ctlformula, Formula, Factor, Termen* and *Expresie*. Thus, the defined rules deliver non-ambiguous specification in algebra Sin_{ctl}.
In following we show the specification of *AU* formula

```
Ctlformula → ”(” Ctlformula ”au” Ctlformula ”)” ;
Macro: sets Z,Z1,Z2;
        Z:=empty_set;Z1:=@3; Z2:= @1;
        while(Z not_equiv Z1) do
          Z:=Z1; Z1:=Z1 union {Z2 intersect {s in all_setS | (succ(s)subset Z1)}};
        endwhile
        @0:=Z1;
```

The *ANTLR 3* [6] tools are used for the construction of software instruments as translators, compilers, recognition and parser of static/dynamic programs. The ANTLR is a generator of compilers; it receives as input a grammar, with a precise description of a language, and generates the source code and other auxiliary files for lexer and parser. The source code of our *ANTLR* grammar presented in paper [1] must contain the specification of all macro-operations presented in the section 3. We “decorate” the grammar for formula language with actions. *ANTLR* inserts our actions in the generated code for parser, and parser must execute embedded actions after matching the preceding grammar element. This is the mechanism of formula evaluation for a given model. In *ANTLRWorks* was implemented all macro-operations of CTL model checker. The program receives as input the model *M* where are defined the sets *S*, *Rel* and *P*.

The detailed specification of *AU* operation defined in table 2 as *actions* in *ANTLR* grammar are presented in the following:

```
ctlFormula returns [HashSet set]
```

```
@init {System.out.println("Incepe..."); init();}
      :  '(' c1=ctlFormula 'au' c2=ctlFormula ')'
                { HashSet rez = new HashSet(); //Z:=∅;
                  HashSet rez1 = new HashSet($c2.set); // Z':= dctl(t2);
                  HashSet rez2 = new HashSet($c1.set); // Z'':= dctl(t1);
                  while (!rez.equals(rez1)) {
                    rez.clear();rez.addAll(rez1);
                    HashSet tmp = new HashSet();
                    boolean include;
                    for (int i=0; i<MAX_STARI; i++) {
                      include = true;
                      for (int j=0; j<MAX_STARI; j++)
                        if (rel[i][j]==1)
                          if (!(rez1.contains(new Integer(j))))
                            include = false;
                      if (include) tmp.add(new Integer(i));
                    }
                    tmp.retainAll(rez2); rez1.addAll(tmp);
                  }
                  trace("ctlFormula",1);
                  printSet(" ( " + $c1.text + " au " + $c2.text + " ) ",rez1);
                  $set = rez1;
                }
```

The behavior of the model checker algorithm demonstrated in [1] consists of identifying the sets of states of a model *M* which satisfy each sub formula of a given *CTL* formula *f* and constructing the set of states, from these sets, that satisfy the formula *f*. This is certainly the behavior of the algorithm for the homeomorphism computation performed by an algebraic compiler. Thus is evaluated an expression by repeatedly identifying its sub expressions and replacing them with their images in the target algebra. In the case of the model-checking algorithm, sub expressions are *CTL* sub formulas and their images are the sets of states in the model satisfy the sub formulas.

5 Algebraic specification of Real-Time CTL extension of CTL model checker

Many times we need to specify when an event is necessary to be happened. For this we need a clock can measure the time. The main idea is to add the feasible constraint clock to states and transition.

Formal verification methods have been developed to reason about the correctness of a system with respect to a given specification. In particular, model checking [7] of temporal logics has become one of the most successful verification techniques. Using this technique requires to adequately model a system by a finite state transition system so that specifications given in temporal logics can be checked for that model.

Real-time systems must perform certain actions within limited time bounds or should start actions only after some point of time. It is therefore natural to label the transitions of the abstract transition system by numbers that denote the time required to move from one state to another one. In general, a transition from state s_1 to state s_2 with label $k \in N$ means that at any time x, where we are in state s_1, we can perform an atomic action that requires k units of time. The action terminates at time $x+k$, where we are in state s_2.

The development of discrete real-time extensions of *CTL* has been initiated in [8], where the temporal operators have been extended by time bounds to limit the number of fixpoint iterations

required to evaluate the considered temporal expression. The models used in [8] were still traditional finite-state transition systems where each transition requires a single unit of time. In order to represent real-time systems in a more compact way, [9] introduced timed transition systems, where transitions are labelled by natural numbers that denote the time consumption of the action associated with the transition.

For come to an *CTL* model checking which generate a real time logic is necessary to extend the *CTL* model with an until bounded operator. This operator contains an interval with a lower bound, denoted with *mi*, and an upper bound, denoted with *ms*, of time step number who allow the transition from one *mi* time to an *ms* time until an event to must happen. For this extension of *CTL* model checking is necessary to extend and modify the target language of sets to handle the various temporal logics.

A real time *CTL* model checker, denoted CTL_T is extending via adds a single specification rule. In this case the until operator *U* have attached an interval of time denoted by *[mi,ms]* which represent when the clock begin to measure and when is stop for that event happened.

Be f_1 and f_2 two CTL_T formula holds on a path $\pi = [s_0, s_1, ..., s_i, ... , s_j, ...]$ if f_2 holds on some states s_i, $mi \leq i \leq ms$ and f_1 holds on all states s_j, $0 \leq j \leq i$.

Definition 2 *(Syntax Definition of CTL_T model checker) If f_1 and f_2 are CTL_T formula and $mi,ms \in N$ then the syntax of CTL_T can be given in Backus-Naur form:*

$$f :: \top | \bot | p | (\neg f_1) | f_1 \wedge f_2 | f_1 \vee f_2 | f_1 \subset f_2 | AX_{[mi,ms]} f_1 | EX_{[mi,ms]} f_1 | AG_{[mi,ms]} f_1 | EG_{[mi,ms]} f_1 | AF_{[mi,ms]} f_1 | EF_{[mi,ms]} f_1 | A[f_1 U_{[mi,ms]} f_2] \mid E[f_1 U_{[mi,ms]} f_2] \quad (2)$$

where $\forall p \in AP$.

Definition 3 *(Semantic Definition of CTL_T model checker) Let $M=(S, Rel, P:AP \rightarrow 2^S)$ be a Kripke model for CTL_T, and s in S, then the semantics of the logic is recursively defined as follows:*

- $(M, s) \models \top$ and $M, s \not\models \bot$ for all $s \in S$.
- $(M, s) \models p$ iff $p \in P(s)$.
- $(M, s) \models \neg f$ iff $(M, s) \not\models f$.
- $(M, s) \models f_1 \wedge f_2$ iff $(M, s) \models f_1$ and $(M, s) \models f_2$.
- $(M, s) \models EX_{[mi,ms]} f$ iff for some s_1 such *that* $s \rightarrow s_1$, $(M, s_1) \models f$.
- $(M, s) \models E[f_1 U_{[mi,ms]} f_2]$ holds iff some a *path* $[s_0, s_1, s_2, ...]$, where $s_0 = s$, and some i with ($mi<i<ms$) and $s_i \models f_2$ *and all* j $[0 \leq j < i$ *then* $s_j \models f_1]$.
- $(M, s) \models A[f_1 U_{[mi,ms]} f_2]$ holds iff all *paths* $[s_0, s_1, s_2, ...]$, where $s_0 = s$, and some i with ($mi<i<ms$) and $s_i \models f_2$ *and all* j $[0 \leq j < i$ *then* $s_j \models f_1]$.

Algebraically the *CTL* algebra Syn_{CTL} is extend to Syn_{CTL_T} which is a heterogeneous algebra with carrier sets *F* and *N* where *F* is the carrier set of Syn_{CTL_T} formulas and *N* is the carrier set of positive integer constants used in the until bounded operator. The until operation is defined by $U{:}F \times F \times N \times N \rightarrow F$. The construct formulas use the bounded until operator and is represented in the algebraic specification of the model checker as *BNF* rules. The algebraic specifications used to generate model checking implementation for CTL_T are the same with *CTL* with the addition of bound.

In following we show the specification of *AU* formula in CTL_T model checker

```
Ctlformula → "(" Ctlformula "au" "[" mi "," ms "]" Ctlformula ")" ;
Macro: sets Z,Z1; unsigned integer ms,mi,count;
        Z= empty_set; Znew:= empty_set; Z1:= @2; Z2=@9;
```

```
mi=@5;ms=@7;count=mi;
while ((Z1 not_equiv empty_set) and (count≤ms)) do
  Z:=Z1;
  Znew:= Z2 intersect {s in all_setS | (succ(s) subset Z1)};
  if(count≥mi and count≤ms)
     Z1:=Z1 union Znew;
  endif
    count:=count+1;
endwhile
@0:=Z1;
```

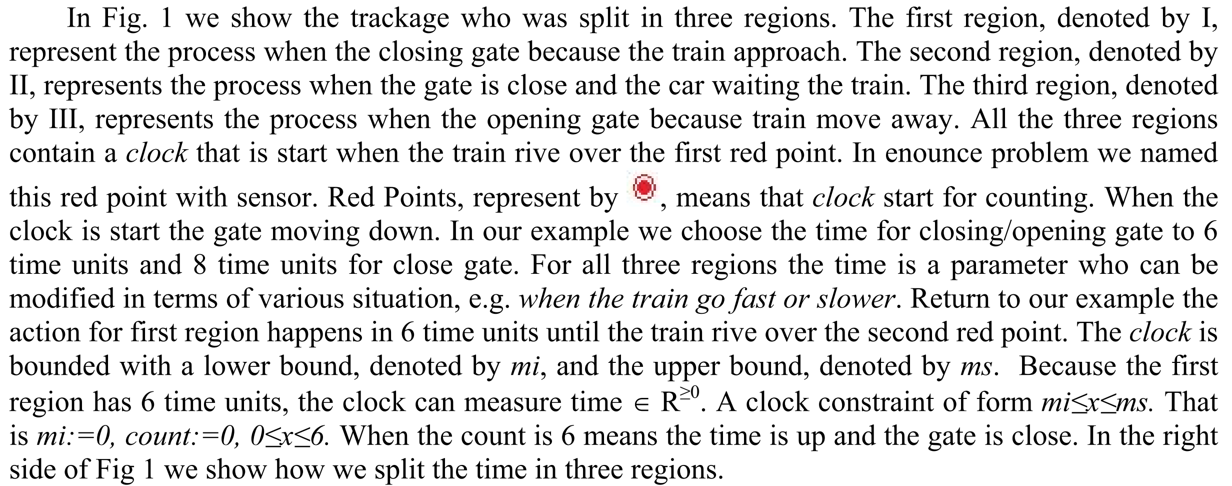
Correctness of a system depends in some cases on the exact timing of events. As a consequence, the models must include the time at which events occur. A usually used formalism to model and reason about timed systems is timed automata [11]. In next section we show an example where we used a timed automaton which is an extension of finite state automata that define a set of real-valued clock variables.

5.1. Train Gate Controller Example

In this subsection we construct two figures which represent the description of Train Gate Controller.

Problem description for Train Gate Controller is: Consider a trackage crossing whose physical layout is represented in Fig.1. There have a road crossing a trackage. Trains and cars cross the passageway area in turns. The crossing involves a gate who keeps the barrier up while the train not coming. The trackage have four sensors detecting when a train enters respectively exits the crossing. When the train approach and cross the first sensor the gate begin to close and also the clock begin to measure. The system consists of three main components, the trackage, the road, the controller, and its behaviour.

Based on these sensor signals, a controller should signal the gate to open/close. In following we prove the properties for the system: *When a train is in the crossing, the gate is closed.*

In Fig. 1 we show the trackage who was split in three regions. The first region, denoted by I, represent the process when the closing gate because the train approach. The second region, denoted by II, represents the process when the gate is close and the car waiting the train. The third region, denoted by III, represents the process when the opening gate because train move away. All the three regions contain a *clock* that is start when the train rive over the first red point. In enounce problem we named this red point with sensor. Red Points, represent by , means that *clock* start for counting. When the clock is start the gate moving down. In our example we choose the time for closing/opening gate to 6 time units and 8 time units for close gate. For all three regions the time is a parameter who can be modified in terms of various situation, e.g. *when the train go fast or slower*. Return to our example the action for first region happens in 6 time units until the train rive over the second red point. The *clock* is bounded with a lower bound, denoted by *mi*, and the upper bound, denoted by *ms*. Because the first region has 6 time units, the clock can measure time $\in R^{\geq 0}$. A clock constraint of form $mi \leq x \leq ms$. That is $mi:=0$, $count:=0$, $0 \leq x \leq 6$. When the count is 6 means the time is up and the gate is close. In the right side of Fig 1 we show how we split the time in three regions.

Definition 4 *The clock are usually written by x,y,…, sets of clock are $Clock_1$, $Clock_2$, … A clock-constraint, denoted with Clock_Con(Clock), over clocks Clock is $g \in$Clock_Con(Clock) where* $g::=x<c|x\leq c|x\geq c|x>c|g\wedge g$ *where $g \in$Clock_Con(Clock) and $c \in N$.*

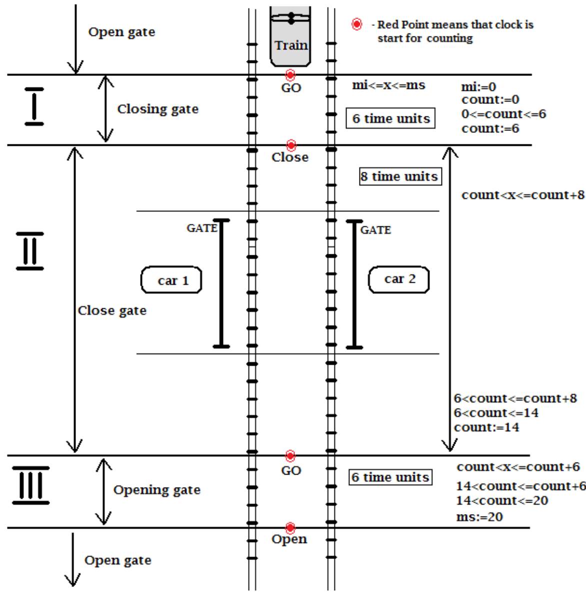


Fig 1. Description time for Train Gate Controller

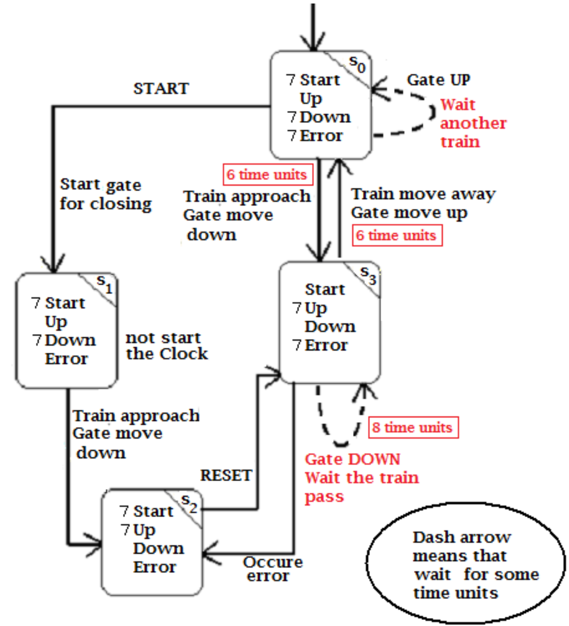


Fig 2. The CTL structure of Train-gate Controller

As example we present in Fig. 2 a scenario for a train gate control system. The state-transition diagram showed in Fig. 2 has four locked-state events. These locked-state events occur because the Gate Train Controller, in most instances, takes one action and then awaits a response before moving for a new state. In fact, only three event flows, *Move Up Request*, *Move Down Request* and *Start the clock* (when we denote with *Up* and *Down* the *move* state when the request exist) do not qualify as locked-state events, because each of them can arrive any time a train and the clock start. The remaining events can occur while the train on-coming to the gate.

Suppose that we have a train gate control which includes in first case a process for normal moving of gate (like, $\{s_0,s_3\}$) and in second case a faulty process (like, $\{s_0,s_1,s_2\}$). In first case for the normal moving of gate process, doesn't shall appear the errors, so the gate is closing and opening normal. The *Clock* is start to counting and cars are shall be stop when the gate is moving or is down. The second process is the faulty process, when the gate doesn't moving when the *Clock* is start to counting. We construct this form of model, to find where the faulty process is, because the objective of model is to correct the event which contains the faulty process. *CTL* structure for the train-gate control is presented in the Fig. 2 and states of the system are denote with $s_0, s_1, \ldots, s_3$.

The Kripke model has four states and the propositional variables are from the set {*Start*, *Up*, *Down*, *Error*}. *Start* represented the *start Clock* when start moving up or down the gate train, *Up* represent the *Up gate*, *Down* is the *Down gate* and *Error* means occur some *error*.

The formal definition of the Kripke structure of the train-gate control is given by: $M = (S, Rel, P)$, where $S=\{s_0,s_1,s_2,s_3\}$, $Rel=\{(s_0,s_0),(s_0,s_1), (s_0,s_3), (s_1,s_2), (s_2,s_3), (s_3,s_0), (s_3,s_2), (s_3,s_3)\}$, $AP=\{Start,Up, Down, Error\}$, P assigns state s_0 in M with *not Start, Up, not Down* and *not Error*, that is set $\{\neg Start, Up, \neg Down, \neg Error\}$. P assigns state s_1 in M with $\{\neg Start, Up, \neg Down, Error\}$, the state s_2 in M with $\{\neg Start, \neg Up, Down, Error\}$, the state s_3 in M with $\{Start, \neg Up, Down, \neg Error\}$.

If the path $\pi = s_0 \rightarrow_{\tau 1} s_1 \rightarrow_{\tau 2} s_2 \rightarrow_{\tau 3} s_3 \ldots \rightarrow_{\tau n} s_n$ is a time-divergent compressed path then $\pi \vDash f_1 U_{[mi,ms]} f_2$ if and only if there is some i such that $s_i +d \vDash f_2$ for some $d \in [0,d_i]$ with $d+\sum_{k=0,\ldots,i-1} d_k \in [mi,ms]$ and for all j≤I and all $d' \in [0,d_j]$ such that $d'+\sum_{k=0,\ldots,j-1} d_k \leq d+\sum_{k=0,\ldots,i-1} d_k$ the relation $sj +d' \vDash f_1 \vee f_2$ is valid. This represents the semantics of CTL_T and is inspired from [2]. For our example the path can be write like $\pi = s_0 \rightarrow_{\tau 1} s_3 \rightarrow_{\tau 2} s_3 \rightarrow_{\tau 3} s_0$ where we define the execution like $ExecutionTime(\pi) = \sum_{\tau i \in R}^{\geq 0}$. That is $\pi = s_0 \rightarrow_6 s_3 \rightarrow_8 s_3 \rightarrow_6 s_0 \equiv s_0 \rightarrow_6 s_0 +6 \rightarrow_a s_3 \rightarrow_8 s_3 +8 \rightarrow_b s_0 \rightarrow_6 s_0 +6$.

In our example we have three regions. Here we have three clock constraints consists of atoms x < | ≤ | ≥ | > | c for some $c \in N$ by *definition 4*. Consider clocks x,y,z and regions $Reg = "x \in (0,6] \wedge y \in (6,14] \wedge z \in (14,20]"$ like in Fig 3.

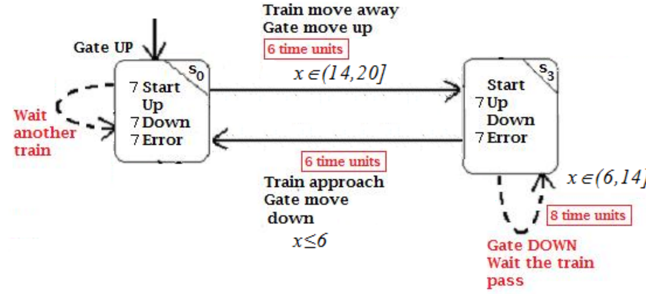


Fig 3. Transition in Region Transition System

Remarks on Region Transition System [2] are $Reg \vDash g$ if and only if for all $\alpha \in Reg$ with $\alpha \vDash g$ if and only if there exists $\alpha \in Reg$ with $\alpha \vDash g$ then there is no ambiguity in the labelling. *Clock* constraints of f CTL_T formula become atomic propositions in Region Transition System, RTS(TA,f), where TA is the *timed automata* defined in [2] and $TA \vDash f$ represent the semantics CTL_T.

Return to our model we check up the following properties: *Is the gate closed for less then 8 minute?* The formula is, $TA \vDash A((Up \wedge \neg Error)\ U_{[7,14]}\ (Down \wedge \neg Error))$.

We showed at first of this section *AU* formula specification in CTL_T model checker. Beginning of this the $A((Up \wedge \neg Error)\ U_{[7,14]}\ (Down \wedge \neg Error))$ formula executing like bellow:

We initialize all sets with $Z := empty_set;\ Z_{new} := empty_set;\ Z1 := @2;\ Z2 = @9;$ where $Z1$ is set with state $Up \wedge \neg Error$. This state is $\{s_0\}$. $Z2$ is set with state $Down \wedge \neg Error$. This state is $\{s_3\}$. The Z_{new} set is constructing with all state s from all_setS which have the successor in $Z1$ and intersect with $Z2$ set. We initialize all positive integers with $mi=@5, ms=@7, count=mi$. That is $mi=7; ms=14; count=mi;$

```
while (({s0} ≠ ∅) and (7≤14)) do
  Z:=Z1={s0};
  Znew:= {s3}∩{s0,s3}={s3};
  if(7≥7 and 7≤14)
     Z1:={s0}∪{s3}={ s0,s3};
  endif
   count:=7+1=8;
  Return to while in next step
     while (({s0,s3} ≠ ∅) and (8≤14)) do
     Z:=Z1={s0,s3};
     Znew:= {s3}∩{s0,s2,s3}={s3};
     if(8≥7 and 8≤14)
        Z1:= {s0,s3}∪{s3}={ s0,s3};
     endif
```

```
count:=8+1=9;
Return to while in next step until count:=14
```

Fig 4. Interpretation step by step for *AU* formula in CTL_T model checker

If count is between *mi* and *ms* bound of until operator these nodes are added to the set of states satisfying formula. The loop terminates when no new nodes are added or the number of steps exceeds the upper limit. The correctness of implementation is given in Fig. 4 for our proposed model.

6 Conclusion

The behaviour of the real time model checker algorithm demonstrated in the section 5 consists of identifying the sets of states of a model *M* which satisfy each sub formula of a given CTL_T formula *f* and constructing the set of states, from these sets, that satisfy the formula *f* over the bound. This is certainly the behaviour of the algorithm for the homeomorphism computation performed by an algebraic compiler. Thus is evaluated an expression by repeatedly identifying its sub expressions and replacing them with their images in the target algebra. In the case of the real time model-checking algorithm, sub expressions are CTL_T sub formulas and their images are the sets of states in the model satisfy the sub formulas.

References

[1] Laura Cacovean, Florin Stoica, *Algebraic Specification Implementation for CTL Model Checker Using ANTLR Tools*, 2008 WSEAS International Conferences, Computers and Simulation in Modern Science - Volume II, Bucharest, Romania, Nov. 2008, ISSN: 1790-5117, ISBN: 978-960-474-032-1

[2] http://www-i2.informatik.rwth-aachen.de/i2/fileadmin/user_upload/documents/AMC09/amc_lec15.pdf

[3] M. Huth and M. Ryan, Logic in Computer Science: *Modelling and Reasoning about Systems*, Cambridge University Press, 2000.

[4] E. Van Wyk, Specification Languages in Algebraic Compilers, Theoretical Computer Science, 231(3):351--385, 2003

[5] P.J. Higgins. *Algebras with scheme of operators*. Mathematische Nachrichten, No.27, 1963/64

[6] Terence Parr , *The Definitive ANTLR Reference, Building Domain-Specific Languages*, version: 2007

[7] E. Clarke and E. Emerson. Design and Synthesis of Synchronization Skeletons using Branching Time Temporal Logic. In D. Kozen, editor, Workshop on Logics of Programs, volume 131 of Lecture Notes in Computer Science, 1981.

[8] E. Emerson, A. Mok, A. Sistla, and J. Srinivasan. Quantitative Temporal Reasoning. Journal of Real-Time Systems, 1992.

[9] S. Campos and E. Clarke. *Real-Time Symbolic Model Checking for Discrete Time Models*. Theories and Experiences for Real-Time System Development, AMAST Series in Computing.World Scientific Press, AMAST Series in Computing, 1994.

[10] E.M. Clarke, E.A. Emerson, and A.P. Sistla. *Automatic verifications of finite-state concurrent systems using temporal logic specifications*. ACM Transactions on Programming Languages and Systems, Vol. 8, No. 2, April 1986

[11] R. Alur and D. L. Dill. *A theory of timed automata*. Theoretical Computer Science, 126(2):183–235, 25 April 1994.

LAURA F. CACOVEAN
Lucian Blaga University of Sibiu, Faculty of Sciences
Department of Computer Science
Str. Dr. Ion Ratiu 5-7, 550012, Sibiu
ROMANIA
laura.cacovean@ulbsibiu.ro