Chapter 6

# Model checking and timed CTL

*Ah! What did I tell you? 88 miles per hour! The temporal displacement occurred at exactly 1:20am and *zero* seconds!* [Dr Emmett Brown]



**Concepts introduced:** TCTL, model checking for TCTL.

A major difference between timed CTL and traditional CTL is that we have clocks. The clocks are found both in the automaton ($X$, a finite set of clock variables), and in the TCTL formula ($Z$, a different finite set of clock variables). We have to take these clocks into account both in the definition of TCTL, and also the model checking relation for TCTL: $\models_\tau$.

In addition, the Kripke structure (or model) used is different, as it corresponds to an RTS (regional transition system) instead of a standard transition system. The (composite) states of the RTS are a pair $\overline{r} = (s, [v]_\approx)$, termed a *region*, where $s$ corresponds to the original state of the transition system, and $[v]_\approx$ is the regional equivalence class for $v$ as defined in Chapter 4.

We begin by formally defining an RTS, and its corresponding model, in terms of the time abstract transition system $\mathrm{TA}_{\mathrm{TTS}}$. Recall that the (possibly infinite) states in $\mathrm{TA}_{\mathrm{TTS}}$ are of the *composite* form $(s, v)$, where $s$ corresponds to the states of the original timed transition system, and $v$ is a valuation of the clocks of that system.

**Definition 33** *Given* $\mathrm{TA}_{\mathrm{TTS}} = (S, S_0, \mathrm{Act}, \rightsquigarrow)$, *then the RTS (regional transition system) is a quotiented transition system* $\mathrm{RTS} = (\overline{R}, \overline{R}_0, \mathrm{Act}, \rightarrow)$. *In this quotiented transition system,*

$$\begin{aligned} \overline{R} &= \{(s, [v]_{\approx}) \mid (s, v) \in S \wedge v \in [v]_{\approx}\}, \text{ and} \\ \overline{R}_0 &= \{(s, [v]_{\approx}) \mid (s, v) \in S_0 \wedge v \in [v]_{\approx}\}, \end{aligned}$$

*and* $(s, [v]_{\approx}) \xrightarrow{a} (s', [v']_{\approx})$ *if and only if there is a transition* $(s, v) \overset{a}{\rightsquigarrow} (s', v')$ *in* $\mathrm{TA}_{\mathrm{TTS}}$. *The elements of the set* $\overline{R}$ *are called the* regions *of the RTS. The notation for identifying a particular region will be* $\overline{r} \in \overline{R}$, *and a (transitory) state with a particular clock valuation within that region will be denoted by* $r = (s, v)$.

Note that since $\equiv_{\mathrm{REG}}$ is a stable equivalence relation of finite index, the RTS is a finite structure. We do not need to differentiate between the RTS and the *zone* based transition system here, instead considering that the zone based transition system is just a more efficient version of the RTS.

The semantics for TCTL is again defined in terms of a Kripke structure or TCTL-model. This model is derived from the RTS.

**Definition 34** *A TCTL model* $\overline{M}$ *over a set* $AP$ *of atomic propositions is a 4-tuple* $(\overline{R}, \Delta, AP, \mathcal{L})$, *where*

1. $\overline{R}$ *is the finite set of* ***regions*** *derived from the* RTS.
2. $\Delta \subseteq \overline{R} \times \overline{R}$ *is a* ***transition relation*** *derived from* $\rightarrow$ *in* RTS. *It must be* total.
3. $AP$ *is a finite set of* ***atomic propositions***.
4. $\mathcal{L} : \overline{R} \rightarrow 2^{AP}$ *is a function which labels each region with the set of atomic propositions true in that region.*

## 6.1 TCTL (Timed CTL)

We define a fragment of TCTL, called TCTL-[1], which we use to specify the formulæ defining the properties to be checked against a TCTL-model. This fragment is sufficient to encode any TCTL formula, and provides the means to specify time-constrained properties.

[1] and pronounced as TCTL-minus.

**Definition 35** *Given a proposition $p \in AP$ (a finite set of atomic propositions), $x \in X$ (a finite set of clock variables in the model), $z \in Z$ (a finite set of clock variables in the property formula) and $\phi \in \Phi(X \cup Z)$ (a clock constraint over both the clocks in the model, and the ones in the TCTL property formula), then $p$ and $\phi$ are both TCTL- formulæ, and if $\psi_1$ and $\psi_2$ are TCTL- formulæ, then*

1. $\neg\psi_1$ *is a TCTL- formula*
2. $\psi_1 \wedge \psi_2$ *is a TCTL- formula*
3. $\psi_1 \vee \psi_2$ *is a TCTL- formula*
4. $z \,\mathrm{in}\, \psi_1$ *is a TCTL- formula*
5. $\mathrm{A}(\psi_1 \,\mathrm{U}\, \psi_2)$ *is a TCTL- formula*
6. $\mathrm{E}(\psi_1 \,\mathrm{U}\, \psi_2)$ *is a TCTL- formula*

The only peculiar part of this definition is the FREEZE definition in line 4. The meaning attached to "$z \,\mathrm{in}\, \psi$" is that $\psi$ holds when the property formula clock $z$ is reset to 0. This corresponds to the clock reset performed during transitions of the automata, and is useful for specifying properties that are bounded in time.

From the above definition it is not completely clear how to provide bounds on the time for something to happen. To do this, temporal operators are subscripted with time constraints. For example, the notation

$$\mathrm{A}(\psi_1 \,\mathrm{U}_{\leq 5}\, \psi_2)$$

expresses the idea that $\psi_1$ holds until within 5 time units, $\psi_2$ becomes true. This may be defined in TCTL- using the FREEZE operator:

$$z \,\mathrm{in}\, \mathrm{A}((\psi_1 \wedge z \leq 5) \,\mathrm{U}\, \psi_2)$$

Note also that in TCTL there is no EX or AX, as there is no (non-timed) *next* operator for timed systems - we do not *jump* from one region to the next.

### 6.1.1 The definition of $\models_\tau$

The model checking relation $\models_\tau$ for timed systems is defined for each atomic proposition $p$, and any clock constraint $\phi$. In $\overline{M}, (r, f) \models_\tau \psi$ we might say that $\psi$ *holds* or is *satisfied* at state $r = (s, v)$ in the case of the (formula) clock valuation $f$. The regions are of the form $\overline{r} = (s, [v]_\approx)$, where $v \in [v]_\approx$.

The model checking relation $\models_\tau$ for each TCTL- formula $\psi_1$, $\psi_2$ is defined as:

$$
\begin{array}{lcl}
\overline{M}, (r, f) \models_\tau p & \Leftrightarrow & p \in L(\overline{r}) \\
\overline{M}, (r, f) \models_\tau \phi & \Leftrightarrow & v \cup f \models \phi \\
\overline{M}, (r, f) \models_\tau \neg\psi_1 & \Leftrightarrow & \text{iff it is not the case that } \overline{M}, (r, f) \models_\tau \psi_1 \\
\overline{M}, (r, f) \models_\tau \psi_1 \wedge \psi_2 & \Leftrightarrow & \text{iff } \overline{M}, (r, f) \models_\tau \psi_1 \text{ and } \overline{M}, (r, f) \models_\tau \psi_2 \\
\overline{M}, (r, f) \models_\tau \psi_1 \vee \psi_2 & \Leftrightarrow & \text{iff } \overline{M}, (r, f) \models_\tau \psi_1 \text{ or } \overline{M}, (r, f) \models_\tau \psi_2 \\
\overline{M}, (r, f) \models_\tau z \,\text{in}\, \psi_1 & \Leftrightarrow & \text{iff } \overline{M}, (r, z \,\text{in}\, f) \models_\tau \psi_1 \\
\overline{M}, (r, f) \models_\tau \mathrm{A}(\psi_1 \,\mathrm{U}\, \psi_2) & \Leftrightarrow & \text{iff for every path } \overline{\pi} = s_0\ s_1\ \ldots \text{ from } r, \\
 & & \text{where for some } j, \overline{M}, \overline{\pi}(j) \models_\tau \psi_2, \\
 & & \text{and } \forall i < j, \overline{M}, \overline{\pi}(i) \models_\tau \psi_1 \vee \psi_2 \\
\overline{M}, (r, f) \models_\tau \mathrm{E}(\psi_1 \,\mathrm{U}\, \psi_2) & \Leftrightarrow & \text{iff there is a path } \overline{\pi} = s_0\ s_1\ \ldots \text{ from } r, \\
 & & \text{where for some } j, \overline{M}, \overline{\pi}(j) \models_\tau \psi_2, \\
 & & \text{and } \forall i < j, \overline{M}, \overline{\pi}(i) \models_\tau \psi_1 \vee \psi_2
\end{array}
$$

In this definition, the progression of time is defined in reference to the states of the original $\mathrm{TS}_{\mathrm{TTS}}$. In particular, a path from one state $r$ is an infinite sequence of states $\overline{\pi} = s_0\ s_1\ \ldots$ such that $s_0 = r$ and $s_i \rightarrow s_{i+1}$. A particular $i$-th element of $\overline{\pi}$ is $\overline{\pi}(i)$.

The notation found in the definition for the FREEZE operator ($\overline{M}, (r, z \,\text{in}\, f) \models_\tau \psi_1$) indicates that $\overline{M}, (r, f) \models_\tau \psi_1$ if all occurences of $z$ in $f$ are reset to 0.

An interesting element of the definition is found in the definitions for $\mathrm{E}(\psi_1 \,\mathrm{U}\, \psi_2)$ and $\mathrm{A}(\psi_1 \,\mathrm{U}\, \psi_2)$, where at some $j$, $\overline{M}, \overline{\pi}(j) \models_\tau \psi_2$, but for all $i < j$, $\overline{M}, \overline{\pi}(i) \models_\tau \psi_1 \vee \psi_2$. If you compare this with the similar definition from CTL, you find in that case the condition "for all $i < j$, $M, \overline{\pi}(i) \models \psi_1$" (i.e. $\psi_1$ instead of $\psi_1 \vee \psi_2$).

We can see the need for the expression $\psi_1 \vee \psi_2$ instead of just $\psi_1$ by considering the *big* step from a particular valuation in $\overline{r}_1$ to another in $\overline{r}_2$ seen in Figure 6.1. For all points in the two regions we want $\mathrm{A}(\psi_1 \,\mathrm{U}\, \psi_2)$, but for the two points connected by the line, $\psi_1$ is not true just before the new point.

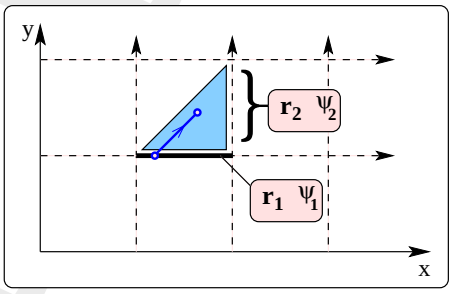



Figure 6.1: AU for timed CTL

## 6.2 Model checking algorithm for TCTL

In a similar manner to the strategy for CTL, we have another labelling algorithm `rsat` (for regional satisfiability):

```
set_of_Regions rsat(Property ψ) =
  if ψ ∈ AP then {(r, f) | ψ ∈ L(r̄)}
  else case ψ of
    true:        R̄
    false:       ∅
    φ:           {(r, f) | v ∪ f ⊨ φ}
    ¬ψ:          R̄ − rsat(ψ)
    ψ1 ∧ ψ2:     rsat(ψ1) ∩ rsat(ψ2)
    ψ1 ∨ ψ2:     rsat(ψ1) ∪ rsat(ψ2)
    z in ψ1:     {(r, f) | (r, z in f) ∈ rsat(ψ1)}
    A(ψ1 U ψ2):  lfp(g(Z) = rsat(ψ2) ∪ (rsat(ψ1) ∩ {(r, f) | ∀r' ∈ (r, f)↑ ∩ Z}))
    E(ψ1 U ψ2):  lfp(h(Z) = rsat(ψ2) ∪ (rsat(ψ1) ∩ {(r, f) | ∃r' ∈ (r, f)↑ ∩ Z}));
```

where $(r, f)^{\uparrow}$ represents the possible future states of $(r, f)$.

### 6.2.1 Practical model checking for TCTL

In practice the model checking problem for TCTL is still hard, and real-time model checkers such as Uppaal generally restrict themselves to analysis for reachability only. This restriction turns out to not be all that limiting, and the resultant model checkers can operate efficiently on *region* or *zone* representations.

In Uppaal, the clock constraints for the formula must be $\emptyset$, and the syntax of the language accepted is restricted to only the following two temporal operators:

$$\mathrm{AG}(\psi)$$
$$\mathrm{EF}(\psi)$$

and $\psi$ is of the form

$$\begin{aligned} \psi \ ::= \quad & a \\ \mid \ & x \ \mathrm{op} \ n \\ \mid \ & \neg\psi \\ \mid \ & \psi_1 \wedge \psi_2 \end{aligned}$$

where $a$ is a location, and $\mathrm{op}$ is a simple comparison operator.

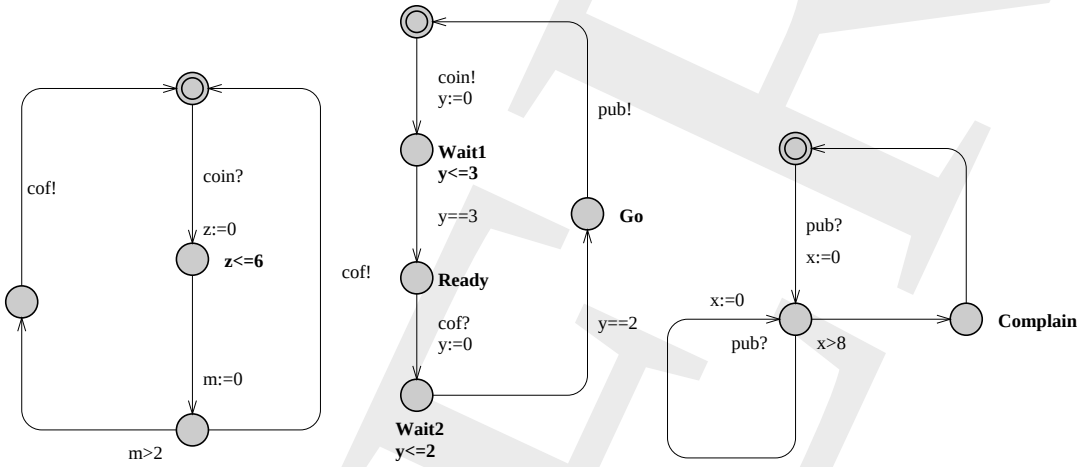
Even though this appears quite restrictive, it is possible to express any sort of reachability query using this subset of TCTL. An algorithm to see if a particular starting state $(s_0, r_0)$ can result in a particular final state $s_f$ given a particular clock assignment $\phi$ is shown below. This algorithm operates over the RTS:

```
set_of_regions checked = ∅, toCheck = {(s0, ro)};
set_of_states final = {sf};
boolean reachable( clock_assignment φ ) =
  while ( toCheck≠ ∅ ) do
    foreach ( (s,r) ∈toCheck ) do
      if ( s = sf ∧ r ∩ φ ≠ ∅ ) then return TRUE;
      if ( ∀(s,r') ∈ checked,r ⊈ r' ) then
        checked = checked+(s,r);
        foreach ( (s',r'),(s,r) → (s',r') ) do
          toCheck = toCheck+(s',r');
  return FALSE;
```

### 6.2.2 Coffee machine example in Uppaal

The problem is to model the behaviour of a system with three components, a coffee *Machine*, a *Person* and an *Observer*. The person repeatedly tries to insert a coin, tries to extract coffee after which (s)he will make a publication. Between each action the person requires a suitable time-delay before being ready to participate in the next one. After receiving a coin the machine should take some time for brewing the coffee. The machine should time-out if the brewed coffee has not been taken before a certain upper time-limit. The observer should complain if at any time more than 8 time-units elapses between two consecutive publications.

The automata are shown in Figure 8.3, and (partially) model the specified system. Why partially? In the specification there is a worrying phrase: “The Machine should time-out if the brewed coffee has not been taken before a certain upper time-limit”. This phrase is worrying because it is an under-specification of the system. For example: “What does the machine do if it times out?”. If it times out and then dumps the coffee, the system will deadlock, as the Person automata must pay and then drink. So - rather than modifying the specified Person automaton, the machine specified here times out and then synchronizes on the dispensing of coffee.




(a) Machine (b) Person (c) Observer

Figure 6.2: The three automata

- **Machine:** The coffee machine accepts a coin and then delays for some time (above it is 6 time units). It then sets a timeout timer, and either (to the right) dispenses coffee, or (to the left) times out and then dispenses coffee. The extra state on the left is because Uppaal does not allow both guards and synchronizing elements to appear on the same transition.
- **Observer:** The observer has an 8 time unit timeout. If the publications keep coming in more often than 8 time units, then the system stays in the middle state. However, if the timer times out, we visit (briefly) the Complain state.
- **Person:** The person was already specified, and it just puts in a coin and then drinks coffee before publishing.

In UPPAAL, the path operators ◊ and □ are written as `<>` and `[]`, so to test the model, the temporal query `E<> Observer.Complain` is used, which corresponds to the CTL formula `EF Observer.Complain`, specifying that:

- for at least one computation path, at some time state `Observer.Complain` is reached.

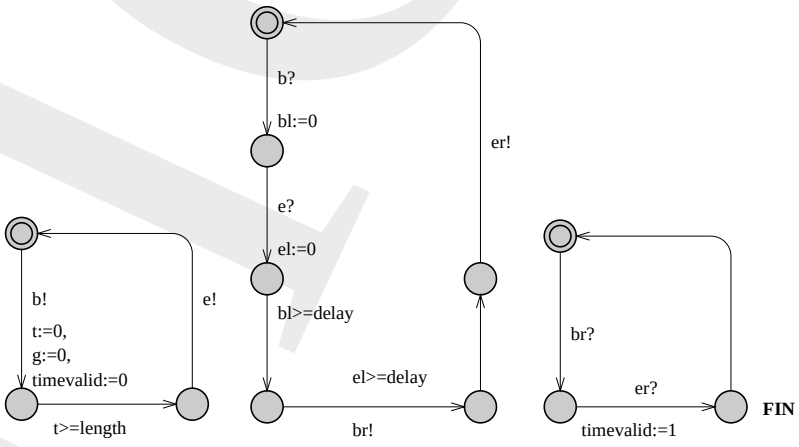
In addition the system is tested with `A[] not deadlock`. The results of the testing are as follows:

- System is deadlock free
- `Observer.Complain` is reached if the coffee timeout is 7 or more
- `Observer.Complain` is never reached if the coffee timeout is 6 or less

The last two tests were done by trial and error - setting the value in the coffee machine model to different values, and rerunning the model checker.

### 6.2.3 A simple protocol example in Uppaal

The problem is to model a simple protocol, with a communication *Medium*, a *Sender*, and a *Receiver*. The sender sends messages of a fixed length `length`, which is the time between the beginning and the end of a message. The medium has a transmission delay `delay`.




(a) Sender (b) Medium (c) Receiver

Figure 6.3: The three automata

- **Sender:** The sender just synchronizes on the beginning and end of the message, ensuring a time of length between the two synchronizations. The `timevalid` and `g` variables are global variables used to time the total transit time of the message.
- **Receiver:** The receiver just synchronizes on the beginning and end of the message after it arrives from the medium, setting `timevalid` as the `FIN` state is entered.
- **Medium:** The medium uses two local clocks, `bl` and `el`, to delay a message enroute to the receiver.

The first step is to model the system, assuming `length<delay`. The model in Figure 6.3 shows this model.

A quick test with `A[] not deadlock` shows that it is deadlock free. To find out the total time between begin send and end receive, a global clock variable $g$ is reset by the sender at the beginning of a message, and its value in state `Receiver.FIN` tells us the total time between the beginning of sending the message and the end of receiving the message. To test this a global variable `timevalid` was added to the system, and if the receiver is in the `Receiver.FIN` state, and the time is valid, then we can run various tests - for example the query

```
E<> (Receiver.FIN and timevalid==1 and g<maxtime)
```

(where `maxtime` is `length+delay`) is always unsatisfied, which tells us there is no time sequence shorter than `length+delay`. The query

```
A[] (Receiver.FIN and timevalid==1 imply g>=maxtime)
```

is satisfied, which tells us that the time will always be greater than or equal to `length+delay`.

The preceding model could only handle systems in which the length of the message was less than the medium delay time. We can extend the medium to also handle

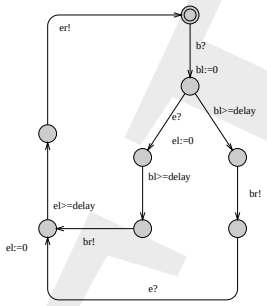



Figure 6.4: The new specification of the medium

messages with `length>=delay`. The only thing that needs changing is the definition of medium, given in Figure 6.4. The two queries before both still produce the expected results, no matter what the relationship between length and delay:

```
E<> (Receiver.FIN and timevalid==1 and g<maxtime)
A[] (Receiver.FIN and timevalid==1 imply g>=maxtime)
```

## 6.3 Summary of topics

In this section, we introduced the following topics:

**Theoretical foundations.** *Formal foundations for TCTL, syntax and semantics.*
**Model checking algorithms.** *Algorithms for checking timed CTL systems.*
**Examples.** *Two worked Uppaal examples.*