Chapter 1

Timed Automata

[1] In the previous chapter we choose ω-words as observable behavior for untimed reactive systems. Moreover, we choose Büchi-automata as specification formalism. Clearly, the observable behavior of a real-time system must include timing information. This is achieved by coupling a real-valued time with each symbol obtaining *timed words*.

1.1 Timed Languages

Given an alphabet Σ. The behavior of a real-time system corresponds to a timed ω-word over Σ. As we are interested in a *dense-time* mode, we choose the set $\mathbb{R}_{\geq 0}$ of non-negative reals as time domain.

Definition 1.1 A *time sequence* $\tau = (\tau_i)_{i \in \omega}$ is an infinite sequence of time values $\tau_i \in \mathbb{R}_{\geq 0}$ that satisfies the following conditions:

1. *Monotonicity:* τ increases monotonically; that is, $\tau_{i+1} \geq \tau_i$, for every $i \in \omega$.

2. *Divergence:* τ diverges, that is, for every $t \in \mathbb{R}_{\geq 0}$ there is $i \in \omega$ such that $\tau_i \geq t$.

A *timed word* over an alphabet Σ is a pair (σ, τ), where σ is an infinite word in Σ^ω and τ is a time sequence. A *timed language* over Σ is a set of timed words over Σ. $\diamond$

The intuitive interpretation of a timed word (σ, τ) is such that at time point τ_i we observe the symbol σ_i.

It is obvious that each timed language L induces a language of untimed words by projecting a timed word (σ, τ) on its first component.

Definition 1.2 For a timed language L over Σ, *Untime*(L) denotes the ω-language such that

$$\mathit{Untime}(L) = \{\sigma \mid \text{there exists a time sequence } \tau \text{ with } (\sigma, \tau) \in L\}$$

$\diamond$

[1] The material in this chapter is based on [AD94]

1.2 Clock constraints and clock evaluations

Timed automata are finite automata extended by *clocks* which are real-valued variables. Clocks are then used to constraints the possible transitions of the automaton by requiring that a transition can only be taken when the actual values of the variables satisfy the guard of the transition. A transition specifies which of the clocks are reset when it is taken.

Definition 1.3 For a set X of clock variables, the set $\Phi(X)$ of *clock constraints* δ is defined by

$$\delta := x \leq c \mid c \leq x \mid \neg\delta \mid \delta_1 \wedge \delta_2,$$

where $x \in X$ and c is a constant in $\mathbb{Q}$. $\diamond$

A *clock evaluation* ν for a set X of clock variables assigns a non-negative real value to each clock. A clock evaluation ν satisfies a clock constraint δ, denoted by $\nu \models \delta$, if δ evaluates to true using the values given by ν. Formally, we define $\nu \models \delta$ by induction on the structure of δ as follows:

- $\nu \models x \leq c$ iff $\nu(x) \leq c$ and $\nu \models c \leq x$ iff $c \leq \nu(x)$,
- $\nu \models \neg\delta$ iff not $\nu \models \delta$, and
- $\nu \models \delta_1 \wedge \delta_2$ iff $\nu \models \delta_1$ and $\nu \models \delta_2$.

For a clock evaluation ν and $t \in \mathbb{R}_{\geq 0}$, $\nu + t$ denotes the clock evaluation that maps each clock variable x to $\nu(x) + t$, and the clock evaluation $t \cdot \nu$ assigns to each clock x the value $t \cdot \nu(x)$. For $Y \subseteq X$, $\nu[Y \mapsto t]$ denotes the clock evaluation for X that assigns t to each clock in Y and coincides with ν on the rest of the clocks. We denote by $\mathbf{0}$ the clock evaluation that assigns 0 to each clock variable.

1.3 Timed transition tables

Definition 1.4 A *timed transition tables* $\mathcal{A}$ is a tuple $(\mathcal{Q}, \mathcal{Q}_0, \mathcal{C}, \mathcal{E})$, where

- $\mathcal{Q}$ is a finite set of control locations,
- $\mathcal{Q}_0 \subseteq \mathcal{Q}$ is a set of initial control locations,
- $\mathcal{C}$ is a **finite** set of clock variables, and
- $\mathcal{E} \subseteq \mathcal{Q} \times \Sigma \times \Phi(\mathcal{C}) \times 2^{\mathcal{S}} \times \mathcal{Q}$ is a set of transitions. A tuple $(q, a, \delta, \lambda, q')$ represents a transition from control location q to control location q' on input symbol a. This transition is guarded by δ and resets the clock variables in λ.

$\diamond$

The main assumption underlying the timed automata model is that executing transitions happens in zero time, and hence, time can only advance when no transition is taken, that is, when control resides at the same control location.

Given a timed word (σ, τ), the timed transition table $\mathcal{A}$ starts in one of its initial states with all clocks initialized to 0. At time τ_i it takes some transition $(q, a, \delta, \lambda, q')$ reading the input σ_i, if the current values of the clock satisfy δ. The effect of the transition is that control jumps to location q' and that the value of the clocks in λ become 0. Between time τ_i and τ_{i+1} control remains at the same location while the value of each clock increases by $\tau_{i+1} - \tau_i$. This behavior is captured by defining *runs* of timed transition tables.

From now on, for a time sequence τ we define $\tau_{-1} = 0$. Then, we define a run as follows:

Definition 1.5 A *run* $r = (q_i, \nu_i)_{i\in\omega}$ of a timed transition table $\mathcal{A} = (\mathcal{Q}, \mathcal{Q}_0, \mathcal{C}, \mathcal{E})$ is an ω-sequence of pairs of control locations and clock evaluations satisfying the following conditions:

- Initiation: $q_0 \in \mathcal{Q}_0$ and $\nu_0 = \mathbf{0}$ and
- Consecution: for every $i \in \omega$ there is a transition $(q_i, \sigma_i, \delta, \lambda, q_{i+1}) \in \mathcal{E}$ such that $\nu_i + (\tau_i - \tau_{i-1})$ satisfies δ and $\nu_{i+1} = (\nu_i + (\tau_i - \tau_{i-1}))[\lambda \mapsto 0]$.

$\diamond$

1.4 Timed regular languages

We obtain timed Büchi automata (TBA for short) from timed transition tables by adding an acceptance set and defining an acceptance condition similar to the acceptance condition in case of automata on ω-words.

Definition 1.6 A *timed Büchi automaton* is a tuple $(\mathcal{Q}, \mathcal{Q}_0, \mathcal{C}, \mathcal{E}, \mathcal{F})$, where $(\mathcal{Q}, \mathcal{Q}_0, \mathcal{C}, \mathcal{E})$ is a timed transition table and $\mathcal{F} \subseteq \mathcal{Q}$ is a set of accepting states.

A run $r = (q_i, \nu_i)_{i\in\omega}$ of a TBA over a timed word (σ, τ) is called an *accepting run*, if $inf(r) \cap \mathcal{F} \neq \emptyset$, where $inf(r) = \{q \mid \exists^\omega i \cdot q_i = q\}$.

For a TBA $\mathcal{A}$, the language $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$ consists of the timed words (σ, τ) such that there exists an accepting run of $\mathcal{A}$ over (σ, τ). $\diamond$

A language L of timed words is called *regular*, if there exists a timed Büchi automaton $\mathcal{A}$ that recognizes L, that is, such that $\mathcal{L}(\mathcal{A}) = L$.

1.5 Properties of timed regular languages

We first consider some closure properties of timed regular languages.

Theorem 1.1 The class of timed regular languages is closed under finite union and intersection. $\square$

Proof: Closure under intersection is an exercise.

The case of union is easy. It suffices to take the disjoint union of the considered TBA's. $\blacksquare$

It is important to note that even in case of timed regular languages the number of symbols in a finite interval of time is **not** necessarily bounded. In fact, the symbols can be arbitrary close to each other. The following example illustrates this points.

Example 1.1 The TBA A_{conv} given in Figure 1.2 recognizes the following language:

$$L_{conv} = \{((ab)^\omega, \tau) \mid \forall i \cdot \tau_{2i+1} = i + 1 \wedge \tau_{2i+1} - \tau_{2i} < \tau_{2i+3} - \tau_{2i+2}\}$$

□

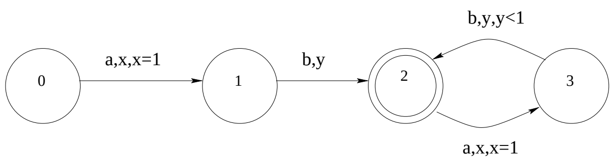

Figure 1.1: The TBA T_{conv}

If we require all the time values τ_i to be multiples of some $\epsilon > 0$, the language accepted by A_{conv} will be empty. Thus, taking as time domain the set of non-negative reals is indeed different from the discrete-time domain. On the other hand, timed automata do not distinguish between reals and rational numbers.

Theorem 1.2 Let L be a timed regular language. For every word σ, $\sigma \in \mathit{Untime}(L)$ iff there exists a time sequence τ such that $(\sigma, \tau) \in L$ and $\tau_i \in \mathbb{Q}$, for every $i \in \omega$. □

Proof: The implication from right to left is trivial.

Consider a timed regular language L that is recognized by a timed automaton $\mathcal{A}$. Let (σ, τ) be a timed word in L. We construct a time sequence τ' with $\tau'_i \in \mathbb{Q}$ and $(\sigma, \tau') \in L$.

Let ϵ be a rational number such that for every constant c appearing in a constraint in $\mathcal{A}$ there exists $n \in \omega$ such that $c = n \cdot \epsilon$. Let $\tau_{-1} = \tau'_{-1} = 0$. We define τ'_i, for $i \geq 0$, recursively. If there exists $j < i$ such that $\tau_i - \tau_j = n \cdot \epsilon$, for some $n \in \omega$, then $\tau'_i = \tau'_j + n \cdot \epsilon$. Otherwise, choose τ'_i such that for every $j < i$ and $n \in \omega$, $\tau'_i - \tau'_j < n \cdot \epsilon$ iff $\tau_i - \tau_j < n \cdot \epsilon$. It can be proved that such a τ'_i must satisfy a finite set of equation of the form $\tau'_i \prec c \cdot \epsilon + c'$, where $\prec \in \{<, \geq\}$ and $c, c' \in \mathbb{Q}$. Such a set of equations either admits no solution or a solution set that contains an interval in $\mathbb{R}_{\geq 0}$. Since $\mathbb{Q}$ is dense in $\mathbb{R}$, we find a solution in $\mathbb{Q}$. It can then be proved that if there exists a run that accepts (σ, τ) then there exists a run that follows the same transitions that accepts (σ, τ'). ■

1.6 The emptiness problem for timed automata

In this section we present an algorithm by Alur and Dill for checking emptiness of timed automata [AD94].

1.6.1 Restriction to integer constants

Lemma 1.3 Let $\mathcal{A}$ be a timed transition table, (σ, τ) a timed word, and $t \in \mathbb{Q}_{\geq 0}$. Then, $(q_i, \nu_i)_{i \in \omega}$ is a run of $\mathcal{A}$ over (σ, τ) iff $(q_i, t \cdot \nu_i)_{i \in \omega}$ is a run $t \cdot \mathcal{A}$ over $(\sigma, t \cdot \tau)$, where $t \cdot \mathcal{A}$ is the timed transition table obtained from $\mathcal{A}$ by multiplying all constants in the guards by t. □

1.6.2 Clock regions

For any $t \in \mathbb{R}_{\geq 0}$, *fract*$(t)$ denotes the fractional part of t, and $\lfloor t \rfloor$ denotes its integral part.

Definition 1.7 Let $\mathcal{A} = (\mathcal{Q}, \mathcal{Q}_0, \mathcal{C}, \mathcal{E})$ be a timed transition table. For each $x \in \mathcal{C}$, let c_x be the largest integer c such that the constraint $x \leq c$ or the constraint $c \leq x$ appears in a guard of some transition in $\mathcal{E}$.

The equivalence relation $\sim$ is defined over the set of all clock evaluation of $\mathcal{C}$; $\nu \sim \nu'$ iff the following conditions are satisfied:

1. For all $x \in \mathcal{C}$, either $\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor$ or $\nu(x) > c_x$ and $\nu'(x) > c_x$.

2. For all $x, y \in \mathcal{C}$ with $\nu(x) \leq c_x$ and $\nu(y) \leq c_y$, *fract*$(\nu(x)) \leq$ *fract*$(\nu(y))$ iff *fract*$(\nu'(x)) \leq$ *fract*$(\nu'(y))$.

3. For all $x \in \mathcal{C}$ with $\nu(x) \leq c_x$, *fract*$(\nu(x)) = 0$ iff *fract*$(\nu'(x)) = 0$.

The relation $\sim$ is an equivalence relation.

Lemma 1.4 The relation $\sim$ is an equivalence relation. □

The equivalence class that contains valuation ν is denoted by $[\nu]$. A *clock region* for $\mathcal{A}$ is an equivalence class of clock evaluations induced by $\sim$. ◇

Clock regions can be effectively represented by specifying

1. for every clock x, one constraint from the set

$$\{x = c \mid c = 0, \cdots, c_x\} \cup \{c - 1 < x < c \mid c = 1, \cdots, c_x\} \cup \{x > c_x\},$$

2. for every x and y such that $c - 1 < x < c$ and $d - 1 < y < d$ appear above whether *fractx* $<$ *fractx*, *fractx* $=$ *fractx*, or *fractx* $>$ *fractx*.

It is not difficult to see that there is a finite number of clock regions.

Given a timed transition table $\mathcal{A} = (\mathcal{Q}, \mathcal{Q}_0, \mathcal{C}, \mathcal{E})$, let $\tilde{\Phi}(\mathcal{C})$ denote the set consisting of constraints over $\mathcal{C}$ in which each variable x is only compared to constants c with $c \leq c_x$. We say that a clock region α satisfies a constraint $\delta \in \tilde{\Phi}(\mathcal{C})$, if there exists a clock valuation $\nu \in \alpha$ which satisfies δ. We have the following lemma.

Lemma 1.5 Let α be a clock region and $\delta \in \Phi(\tilde{\mathcal{C}})$. Then, for every $\nu, \nu' \in \alpha$, $\nu \models \delta$ iff $\nu' \models \delta$. □

Given a clock region α and $\lambda \subseteq \mathcal{C}$, the region $\alpha[\lambda \mapsto 0]$ is the region $\nu[\lambda \mapsto 0]$, where $\nu \in \alpha$.

1.6.3 The region automaton

In this section we define a transition table $R(\mathcal{A})$, called *region transition table*, associated to the timed transition table $\mathcal{A}$ which mimics the runs of $\mathcal{A}$.

To do so we define the *time-successor* of a clock region.

Definition 1.8 A clock region α' is a *time-successor* of a region α iff for every valuation $\nu \in \alpha$ there exists $t \in \mathbb{R}_{\geq 0}$ such that $\nu + t \in \alpha$. $\diamond$

Then, we have the following lemma.

Lemma 1.6 There is a recursive function that associates to every clock region α the set consisting of the time-successors of α. $\square$

Proof: Exercise. $\blacksquare$

Definition 1.9 For a timed transition table $\mathcal{A} = (\mathcal{Q}, \mathcal{Q}_0, \mathcal{C}, \mathcal{E})$ the corresponding *region transition table* $R(\mathcal{A})$ is a transition table over the alphabet of $\mathcal{A}$ such that

1. the states of $R(\mathcal{A})$ are pairs (q, α) where $q \in \mathcal{Q}$ and α is clock region,
2. the initial states of $R(\mathcal{A})$ are the states in $\mathcal{Q}_0 \times \{[\nu_0]\}$, and
3. $((q, \alpha), \sigma, (q'\alpha'))$ is an edge of $\mathcal{A}$ iff there exists an edge $(q, \sigma, \delta, \lambda, q')$ of $\mathcal{A}$ and a time-successor α'' of α such that α'' satisfies δ and $\alpha' = \alpha''[\lambda \mapsto 0]$.

$\diamond$

Next we establish a correspondence between the runs of $\mathcal{A}$ and the runs of $R(\mathcal{A})$.

Definition 1.10 For a run $r = (q_i, \nu_i)_{i\in\omega}$ of $\mathcal{A}$ we define its projection $\bar{r}$ to be the sequence $(q_i, [\nu_i])_{i\in\omega}$. $\diamond$

It can easily be proved that if r is a run of $\mathcal{A}$ then $\bar{r}$ is a run of $R(\mathcal{A})$. the converse is, however, in general not true, that is not every run of $R(\mathcal{A})$ is a projection of a run of $\mathcal{A}$. This is the case because runs of $\mathcal{A}$ must satisfy the divergence condition; no such condition is requested for the runs of $R(\mathcal{A})$. To formulate such a condition we assume that every timed automaton contains a special clock z such that z is reset by every transition. Notice that this assumption does not restrict the class of languages recognized by timed automata. Now, we introduce the following definition.

Definition 1.11 A run $\bar{r} = (q_i, [\nu_i])_{i\in\omega}$ of $R(\mathcal{A})$ is called *progressive*, if the following conditions are satisfied:

1. for every clock $x \neq z$ there are infinitely many positions $i \in \omega$ such that α_i satisfies $x = 0 \vee x > c_x$ and
2. there are infinitely many positions $i \in \omega$ such that α_i satisfies $z > 0$.

$\diamond$

We will show below that progressive runs of $R(\mathcal{A})$ correspond exactly to the projected runs of $\mathcal{A}$. Moreover, the following example shows that this is not case if we remove the second condition.

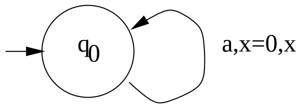

Figure 1.2: Counter example

Example 1.2 □

Lemma 1.7 If $\bar{r} = (q_i, [\nu_i])_{i\in\omega}$ is a progressive run of $R(\mathcal{A})$ then there exists a time sequence $(\tau_i)_{i\in\omega}$ such that $r' = (q_i, \tau_i)_{i\in\omega}$ is a run of $\mathcal{A}$ and $\bar{r'} = \bar{r}$. □

1.6.4 The untiming construction

Let $\mathcal{A} = (\mathcal{Q}, \mathcal{Q}_0, \mathcal{C}, \mathcal{E}, \mathcal{F})$ be a timed automaton. By abuse of notation, we denote by $R(\mathcal{A})$ the region transition table associated to $(\mathcal{Q}, \mathcal{Q}_0, \mathcal{C}, \mathcal{E})$.

Now, for a timed automaton $\mathcal{A}$ we can extend $R(\mathcal{A})$ by an acceptance conditions to obtain a Büchi automaton $\mathcal{R}(\mathcal{A})$ called the *region automaton* of $\mathcal{A}$ which recognizes $Untime(\mathcal{L}(\mathcal{A}))$.

Theorem 1.8 [G] iven a timed automaton $\mathcal{A} = (\mathcal{Q}, \mathcal{Q}_0, \mathcal{C}, \mathcal{E}, \mathcal{F})$ there exists a Büchi automaton $\mathcal{R}(\mathcal{A})$ which recognizes $Untime(\mathcal{L}(\mathcal{A}))$. □

Proof: An extended Büchi automaton $B = (\mathcal{Q}', \mathcal{Q}'_0, \mathcal{E}')$ consists of a transition table and a sequence $(F_1, \cdots, F_n)$ of acceptance sets, i.e. $F_i \subseteq \mathcal{Q}'$. A run r of B is called accepting, if $\inf(r) \cap F_i \neq \emptyset$, for every $i = 1, \cdots, n$. It is not difficult to sea that every extended Büchi automaton can be transformed into a Büchi automaton which recognizes the same language. Therefore, to prove the theorem it suffices to give an extended Büchi automaton that recognizes $Untime(\mathcal{L}(\mathcal{A}))$.

Let B be the extended Büchi automaton whose transition table is $R(\mathcal{A})$ and which includes the following acceptance conditions:

- the set $\{(q, \alpha) \mid q \in \mathcal{F}\}$,
- for every clock $x \neq z$, the set $\{(q, \alpha) \mid \alpha \models x = 0 \lor x > c_x\}$ and
- the set $\{(q, \alpha) \mid \alpha \models z > 0\}$.

■

Chapter 2

Model-Checking for CTL

Let $\mathcal{P}$ be a finite set of propositions.

Definition 2.1 *A* Kripke structure *is given by a triple* $(\mathcal{Q}, R, \mu)$*, where*

- $\mathcal{Q}$ *is a finite of states,*
- $R \subseteq \mathcal{Q} \times \mathcal{Q}$ *is a transition relation, and*
- $\mu : \mathcal{Q} \longrightarrow 2^{\mathcal{P}}$ *is a labeling function.*

$\diamond$

Computation Tree Logic (CTL) [CES86] is a temporal logic that allows to specify branching properties of concurrent systems. It syntax is given by the following rules:

- Every proposition in $\mathcal{P}$ is a formula.
- If f and g are formulas then so are $\neg f$, $f \wedge g$, $\forall \bigcirc f$, $\exists \bigcirc f$,$\forall f \mathcal{U} g$ and $\exists f \mathcal{U} g$.

CTL formulas are interpreted aver pairs of Kripke structures and states. In order to define the interpretation of CTL formulas we need the following definitions:

Definition 2.2 *A path* π *starting at* q *of a Kripke structure* $\mathcal{K} = (\mathcal{Q}, R, \mu)$ *is an infinite sequence* $q_0, q_1, \ldots$ *such that* $q_0 = q$ *and* $(q_i, q_{i+1}) \in R$*, for every* $i \geq 0$.
Given a path $\pi = (q_i)_{i \geq 0}$*, we write* $\pi(j)$ *for* q_j. $\diamond$

A path π of a Kripke structure $\mathcal{K}$ satisfies the formula $f \mathcal{U} g$, where f and g are CTL formulae, if there exists $j \geq 0$ such that $\mathcal{K}, \pi(j) \models g$ and $\mathcal{K}, \pi(i) \models g$, for every $i < j$.

The *satisfaction relation* $\mathcal{K}, q \models f$ is defined on the structure of f. Thus, assume we are given a Kripke structure $\mathcal{K} = (\mathcal{Q}, R, \mu)$.

- $\mathcal{K}, q \models P$ iff $P \in \mu(q)$.
- $\mathcal{K}, q \models \neg f$ iff not $\mathcal{K}, q \models f$.
- $\mathcal{K}, q \models f \wedge g$ iff $\mathcal{K}, q \models f$ and $\mathcal{K}, q \models g$.

- $\mathcal{K}, q \models \forall\bigcirc f$ iff $\mathcal{K}, q' \models f$, for every q' with $(q, q') \in R$.
- $\mathcal{K}, q \models \exists\bigcirc f$ iff $\mathcal{K}, q' \models f$, for some q' with $(q, q') \in R$.
- $\mathcal{K}, q \models f \forall\mathcal{U} g$ iff for every path π starting at q, $\pi \models f\mathcal{U}g$.
- $\mathcal{K}, q \models f \exists\mathcal{U} g$ iff for some path π starting at q, $\pi \models f\mathcal{U}g$.

Given a Kripke structure $\mathcal{K}$ and a CTL formula f, a *model-checking algorithm* determines which of the states of $\mathcal{K}$ satisfy f, i.e. for which states q we have $\mathcal{K}, q \models f$. The *enumerative* model-checking alogorithm for CTL operates by labeling each state q of $\mathcal{K}$ by all the subformulas of f which are true in q. The algorithms works recursively on the structure of f. In the sequel we present an enumerative model-checking alogorithm for CTL. We need first a few definitions.

Let $sub^+(f)$ denote the set of all subformulas of f of the form $\forall f_1 \mathcal{U} f_2$, $\exists f_1 \mathcal{U} f_2$, $\forall\bigcirc f_1$, $\exists\bigcirc f_1$, and $f_1 \wedge f_2$. For every $g \in sub^+(f)$, let $\|g\|$ denote the maximal depth of the operators in g above i.e. $\|\forall f_1 \mathcal{U} f_2\| = 1$, if f_1 and f_2 are propositions and $\|\forall f_1 \mathcal{U} f_2\| = \max(\|f_1\|, \|f_2\|) + 1$, otherwise, and similarly for the other operators.

The enumerative algorithm works by induction on i and associates with each $q \in \mathcal{Q}$ the set of formulas $g \in sub^+(f)$ with $\|g\|$ such that $\mathcal{K}, q \models g$. Initially, each state q is labeled by $\mu(q)$, that is, $lab(q) = \mu(q)$. After termination of the algorithm we have $g \in lab(q)$ iff $\mathcal{K}, q \models g$, for every $g \in sub^+(f)$. Here we only present how the algorithm handles a formula of the form $\forall f_1 \mathcal{U} f_2$. The complete formulation of the algorithm is an exercise. Thus, consider a formula $\forall f_1 \mathcal{U} f_2$ and assume that for every $q \in \mathcal{Q}$ and $i = 1, 2$, $q \in lab(f_i)$ iff $\mathcal{K}, q \models f_i$. The following procedure labels a set q with $f = \forall f_1 \mathcal{U} f_2$ iff $\mathcal{K}, q \models f$:

```
FOR q ∈ Q DO
 IF f2 ∈ lab(q) THEN lab(q) := lab(q) ∪ {f}
 ENDIF
ENDFOR
FOR j = 1 TO |Q| DO
  FOR q ∈ Q DO
    IF f1 ∈ lab(q) ∧ ∀q' ∈ Q · (q, q') ∈ R ⇒ f ∈ lab(q')
    THEN lab(q) := lab(q) ∪ {f}
    ENDIF
  ENDFOR
ENDFOR
```

Figure 2.1: Labeling for the $\forall\mathcal{U}$ operator.

2.1 Model-Checking for Fair CTL

Definition 2.3 *A* fair *Kripke structure is Kripke structure $(\mathcal{Q}, R, \mu)$ extended with a set $\mathcal{F}$ of non-empty subsets of $\mathcal{Q}$.* ◇

A fair path $\pi = (q_i)_{i\geq 0}$ of a fair Kripke structure $(\mathcal{Q}, R, \mu, \mathcal{F})$ is a path of $(\mathcal{Q}, R, \mu)$ such that $\inf(\pi) \cap F \neq \emptyset$, for every $F \in \mathcal{F}$.

CTL formulas can be interpreted over fair Kripke structure by restricting path quantification to fair paths. The satisfaction relation is denoted by $\models_F$ and defined as follows:

- $\mathcal{K}, q \models_F P$ iff $P \in \mu(q)$.
- $\mathcal{K}, q \models_F \neg f$ iff not $\mathcal{K}, q \models_F f$.
- $\mathcal{K}, q \models_F f \wedge g$ iff $\mathcal{K}, q \models_F f$ and $\mathcal{K}, q \models_F g$.
- $\mathcal{K}, q \models_F \forall\bigcirc f$ iff $\mathcal{K}, q' \models_F f$, for every q' with $(q, q') \in R$.
- $\mathcal{K}, q \models_F \exists\bigcirc f$ iff $\mathcal{K}, q' \models_F f$, for some q' with $(q, q') \in R$.
- $\mathcal{K}, q \models_F f\forall\mathcal{U}g$ iff for every fair path π starting at q, we have $\pi \models f\mathcal{U}g$.
- $\mathcal{K}, q \models_F f\exists\mathcal{U}g$ iff for some fair path π starting at q, we have $\pi \models f\mathcal{U}g$.

We will reduce the model-checking of fair CTL to the model-checking of CTL. To do so we introduce the following definiton. Let $\mathcal{K} = (\mathcal{Q}, R, \mu, \mathcal{F})$ be a fair Kripke structure. We denote by $\mathcal{K}^f = (\mathcal{Q}, R, \mu')$ the Kripke structure obtained from $\mathcal{K}$ be labeling every state q with the new atomic proposition *fair*, if $\mathcal{K}, q \models_F \exists\Box true$. Then, we have the following:

Lemma 2.1 $\mathcal{K}, q \models_F \exists f_1\mathcal{U}f_2$ iff $\mathcal{K}^f, q \models \exists(f_1\mathcal{U}(f_2 \wedge fair))$. $\Box$

Thus, by this lemma, if we can effectively construct the structure $\mathcal{K}^f$, then we can use the enumerative algrithm to model-check fair CTL. Therefore, we show how to construct $\mathcal{K}^f$. It suffices to show that there is an algorithm that labels a state q of $\mathcal{K}$ by *fair* iff $\mathcal{K}, q \models_F \exists\Box true$. A subset $\mathcal{Q}'$ of $\mathcal{Q}$ is called a *strongly connected component*, SCC for short, of $\mathcal{K}$, if for all $q, q'\mathcal{Q}'$ there is a finite sequence $q_0, \cdots, q_n$ with $n \geq 1$ such that $q_0 = q$, $q_n = q'$, and $(q_i, q_{i+1}) \in R$, for $i < n$. An SCC is called fair, if it conatins a state in F, for every $F \in \mathcal{F}$. We say that an SCC $\mathcal{Q}'$ is reachable from q, if there is a state $q' \in \mathcal{Q}'$ and a finite sequence $q_0, \cdots, q_n$ with $n \geq 1$ such that $q_0 = q$, $q_n = q'$, and $(q_i, q_{i+1}) \in R$, for $i < n$. Then, we have the following lemma:

Lemma 2.2 $\mathcal{K}, q \models_F \exists\Box true$ iff there is a fair SCC that is reachable from q. $\Box$

The following algorithm labels a state q with *fair* iff $\mathcal{K}, q \models_F \exists\Box true$:

Determine the set $\{C_1, \cdots, C_n\}$ of all fair maximal SCC's;
FOR $q \in \mathcal{Q}$ DO
 IF $\exists i \leq n \cdot q \in C_i$ THEN $lab(q) := lab(q) \cup \{fair\}$
 ENDIF
ENDFOR
FOR $j = 1$ TO $|\mathcal{Q}|$ DO
  FOR $q \in \mathcal{Q}$ DO
    IF $\exists q' \in \mathcal{Q} \cdot (q, q') \in R \wedge fair \in lab(q')$
    THEN $lab(q) := lab(q) \cup \{fair\}$
    ENDIF
 ENDFOR
ENDFOR

Figure 2.2: The labeling algorithm for $\models_F \exists \Box true$.

Bibliography

[AD94] R. Alur and D. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126, 1994.

[CES86] E.M. Clarke, E.A. Emerson, and E. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 1986.