Contents

5 Real-Time Modules .................................................. 1
  5.1 Clock Variables ................................................. 2
  5.2 Real-time Invariant Verification ............................ 3
    5.2.1 Partition Refinement .................................... 5
    5.2.2 Symbolic Analysis of Propositional Real-time Modules . 6
    5.2.3 Difference-bound Matrices ............................. 8
Chapter 5

Real-Time Modules

In reactive modules, the progress of time is abstracted into a sequence of rounds. The abstraction of time in this fashion is convenient in many circumstances. First, every round may represent a clock cycle, as in our model of synchronous circuits. Second, a special tick event may represent a clock cycle, with an arbitrary number of rounds between ticks, as in our model of timed asynchronous circuits. Third, quantitative time may be irrelevant altogether to the problem at hand, as in our model of mutual-exclusion protocols, whose correctness ought to be independent of the relative speeds of the processes. Sometimes, however, it is necessary to take a more accurate, real-numbered view of time. For this purpose, we introduce an extension of reactive modules called real-time modules.

While the behavior of a reactive module is a sequence of update rounds, the behavior of a real-time module is a sequence of update and delay rounds. During an update round, the values of variables are updated; during a delay round, the values of ordinary variables remain unchanged, while the values of special variables called clocks measure the amount of time that elapses. We assume the synchrony hypothesis, that no time elapses during update rounds. The synchrony hypothesis is a modeling assumption; it simply asserts that all time delays must be modeled explicitly by delay rounds. For example, if an assignment to a variable takes 1 time unit, it must be modeled as a delay round of duration 1 followed by an update round that changes the value of the variable. For mathematical purposes, it will be convenient to separate updates from delays in this fashion.
5.1 Clock Variables

Real-time modules have two kinds of variables: discrete variables and clock variables (or clocks, for short). Discrete variables are updated by guarded assignments, as before. Clock variables range over the nonnegative real numbers, and may change in two different ways. First, like a discrete variable, a clock may be updated by a guarded assignment. Second, while time elapses, the value of a clock increases, measuring the progress of time. We declare clock variables using the type $\mathbb{C}$; all other declared variables are discrete by default. A reactive module, then, is simply the special case of a real-time module without clocks.

Consider two sets $X$ and $Y$ of typed variables with $Y \subseteq X$. A guarded delay $\gamma$ from $X$ to $Y$ is a boolean expression $p_\gamma$ over $X$, written

$$\gamma \rightarrow \text{wait}.$$  

Informally, the guarded delay $\gamma$ can be executed if the boolean expression $p_\gamma$ evaluates to true. The execution of $\gamma$ leaves the value of each discrete variable in $Y$ unchanged, and advances the value of each clock variable in $Y$ by some uniform real value $\delta$ such that the truth value of $p_\gamma$ remains true throughout the advancement. Given a valuation $s$ for $X$, and a real number $\epsilon$, by $s + \epsilon$ we denote the valuation for $X$ that maps each discrete variable $x$ in $X$ to $x[s]$, and each clock variable $y$ in $X$ to $y[s] + \epsilon$. The guarded delay $\gamma$ defines a ternary relation $[\gamma] \subseteq \Sigma_X \times \mathbb{R}_{\geq 0} \times \Sigma_Y$: $(s, \delta, t) \in [\gamma]$ iff (1) $t = Y[s] + \delta$ and (2) for every nonnegative real $\epsilon \leq \delta$, the valuation $s + \epsilon$ satisfies $\gamma$.

A guarded real-time command $\Gamma$ from $X$ to $Y$ is a finite set $\{\gamma_i \mid i \in I\}$ of guarded assignments and guarded delays from $X$ to $Y$ such that the disjunction ($\forall i \in I, p_{\gamma_i}$) of the guards is valid. The guarded real-time command $\Gamma$ defines a ternary relation $[\Gamma] \subseteq \Sigma_X \times \mathbb{R}_{\geq 0} \times \Sigma_Y$: $(s, \delta, t) \in [\Gamma]$ iff for some $i \in I$, either (1) $\gamma_i$ is a guarded assignment, $\delta = 0$, and $[\gamma_i](s) = t$, or (2) $\gamma_i$ is a guarded delay and $(s, \delta, t) \in [\gamma_i]$.

Each real-time module defines a transition graph. The states of a real-time module are the valuations for the module variables. Since the values of clock variables are real numbers, the state space of a real-time module is usually infinite. A real-time module has two kinds of transitions. Update transitions correspond to guarded assignments, which update the values of discrete and clock variables, and time transitions correspond to guarded delays, which advance the values of clock variables and leave the values of discrete variables unchanged.
**Transition relation of a real-time module**

Let $P$ be a real-time module, let $s$ and $t$ be two states of $P$, and let $t' = t[X_{P} := X_{P}']$. The state pair $(s, t)$ is a *transition* of $P$ if there is a nonnegative real $\delta$ such that for every atom $U$ of $P$,

$$(\text{read}X_{U}[s] \uplus \text{await}X_{U}'[t'], \delta, \text{ctr}X_{U}'[t']) \in [\text{update}_U],$$

and if $\delta > 0$, then for every external discrete variable $x$ of $P$, $x[t] = x[s]$, and for every external clock variable $y$ of $P$, $y[t] = y[s] + \delta$. If $\delta = 0$, then $(s, t)$ is an *update transition*; if $\delta > 0$, then $(s, t)$ is a *time transition* of duration $\delta$.

**Remark 5.1** [Time transitions] If $(s, t)$ is a time transition of duration $\delta$, then $t = s + \delta$. Moreover, for each nonnegative real $\epsilon < \delta$, $(s, s + \epsilon)$ is also a time transition. It follows that the transition graph of a real-time module is usually infinitely branching.

Operationally, in each update round the atoms of a module are sorted topologically with respect to the precedes relation. In each subround, an atom either updates its controlled variables, or proposes a time delay (the proposed time delay is not known to the other atoms). After all subrounds, if some atom has updated its controlled variables, then the module moves instantaneously to the next update round. If, on the other hand, all atoms have proposed time delays, then the module waits for the amount of time equal to the minimum of the proposed time delays, before moving to the next update round.

**Example 5.1** [Real-time counter] The following module increments a counter $x$ every 3 to 5 time units:

```plaintext
module RealTimeCount
  interface x : N
  private y : C
  atom controls x, y reads x, y
  init
    || true -> x' := 0; y' := 0
  update
    || y ≥ 3 -> x' := x + 1; y' := 0
    || y ≤ 5 -> wait
```

### 5.2 Real-time Invariant Verification

The invariant verification problem is decidable for an interesting class of real-time modules, because we can construct finite stable partitions of the infinite state spaces.
module RealTimeTrain

interface pc : {far, near, gate}; arrive : E

private x : C

atom controls pc, x, arrive reads pc, x, arrive

init
  true → pc' := far

update
  pc = far → pc' := near; arrive!; x' := 0
  pc = far → wait
  pc = near ∧ x ≥ 3 → pc' := gate; x' := 0
  pc = near ∧ x ≤ 5 → wait
  pc = gate ∧ x ≥ 1 → pc' := far
  pc = gate ∧ x ≤ 2 → wait

module RealTimeGate

external arrive : E

interface pc : {open, closing, closed}

private y : C

atom controls pc, y reads pc, y, arrive awaits arrive

init
  true → pc' := open

update
  pc = open ∧ arrive? → pc' := closing; y' := 0
  pc = open ∧ ¬arrive? → wait
  pc = closing ∧ y ≥ 1 → pc' := closed; y' := 0
  pc = closing ∧ y ≤ 2 → wait
  pc = closed ∧ y ≥ 7 → pc' := open
  pc = closed ∧ y ≤ 7 → wait

Figure 5.1: Real-time railroad gate control
Propositional real-time modules
The clock constraints are the boolean expressions generated by the grammar
\[ p ::= x \leq c \mid c \leq x \mid x + c \leq y + d \mid p \land p, \]
where \( x \) and \( y \) are clock variables, and \( c \) and \( d \) are integer constants. A clock formula is a boolean combination of propositional formulas and clock constraints. A propositional real-time module is a real-time module \( P \) such that (1) all discrete variables of \( P \) are propositions, and (2) every expression that appears in the guards of initial and update commands of \( P \) is a clock formula, and (3) every expression that appears in the assignments of initial and update commands of \( P \) is a propositional formula or an integer constant.

Example 5.2 [Real-time train] Figure 5.1 shows an example of two propositional real-time modules that model a train and a gate controller. Consider the train system

\[ \text{module RailRoadXing is RealTimeTrain}[pc := p_{cT}] \parallel \text{RealTimeGate}[pc := p_{cG}]. \]

We want to show that the clock formula

\[ p_{cT} = \text{gate} \rightarrow p_{cG} = \text{closed} \]

is an invariant of RailRoadXing.

5.2.1 Partition Refinement

Theorem 5.1 [Propositional real-time modules] [Alur and Dill] Let \( P \) be a propositional real-time module, and let \( \preceq^l \) be a finite partition of \( G_P \) such that every equivalence class of \( \preceq^l \) can be defined by an observation predicate for \( P \) that is a clock formula. Then the coarsest stable refinement \( \min(\preceq^l) \) has finitely many equivalence classes.

Proof. Let \( c \) be the largest constant occurring in \( P \) and the formulas that define the equivalence classes of \( \preceq^l \). Define the region equivalence \( s \preceq t \) iff (1) for all discrete variables \( x \) of \( P \), \( x[s] = x[t] \); (2) for all clock variables \( y \) of \( P \), either \( [y[s]] = [y[t]] \) and \( [y[s]] = [y[t]] \), or \( [y[s]] > c \) and \( [y[t]] > c \); and
(3) for all clock variables $y$ and $z$ of $P$, $\langle x[s] \rangle < \langle y[s] \rangle$ if and only if $\langle x[t] \rangle < \langle y[t] \rangle$ (where $\langle x \rangle = x - \lfloor x \rfloor$). Then $\cong$ is a stable refinement of $\preceq_l$. □

It follows that symbolic backward reachability checking and symbolic partition refinement terminates on the transition graphs of propositional real-time modules. The number of equivalence classes is $O(2^{k + l} \cdot l! \cdot (c_{max} + 1))$, which is exponential in the size of the module $P$.

**Corollary 5.1** [Real-time invariant verification] The propositional real-time invariant verification problem can be solved in exponential time.

**Exercise 5.1** {T3} [PSPACE invariant verification] Prove that the propositional real-time invariant verification problem can be solved polynomial space.

**Exercise 5.2** {P2} [Region graph] The quotient graph $G_p/\pi$, where $P$ is a propositional real-time module and $\pi$ is the equivalence relation from the proof of Theorem 5.1, is called the **region graph** of $P$. Draw the region graph of the real-time module RailRoadXing. □

**Exercise 5.3** {P3} [Coarsest stable refinement] Consider again the real-time module RailRoadXing and the initial partition $\hat{\pi}$, with two equivalence classes, that separates the states in which the train is inside a non-closed gate from all other states. (1) Step through partition refinement to find the coarsest stable refinement of $\hat{\pi}$. (2) Step through the Lee-Yannakakis algorithm. □

### 5.2.2 Symbolic Analysis of Propositional Real-time Modules

The transition predicates of real-time modules can be constructed using quantifiers over the reals. For example, the transition predicate of the module Real-TimeTrain is

\[
\begin{align*}
\forall (pc = far \land pc' = near \land x' = 0 \land arrive' \neq arrive) \\
\forall (\exists \delta \geq 0, pc' = pc \land x' = x + \delta \land arrive' = arrive \land (\forall 0 \leq \epsilon \leq \delta, pc = far)) \\
\forall (pc = near \land x \geq 3 \land pc' = gate \land x' = 0 \land arrive' = arrive) \\
\forall (\exists \delta \geq 0, pc' = pc \land x' = x + \delta \land arrive' = arrive \land (\forall 0 \leq \epsilon \leq \delta, pc = near \land x + \epsilon \leq 5)) \\
\forall (pc = gate \land x \geq 2 \land pc' = far \land x' = x \land arrive' = arrive) \\
\forall (\exists \delta \geq 0, pc' = pc \land x' = x + \delta \land arrive' = arrive \land (\forall 0 \leq \epsilon \leq \delta, pc = gate \land x + \epsilon \leq 2))
\end{align*}
\]

This transition predicate is not a clock formula, because it contains existential quantifiers, and subformulas of the form $x' = x + \delta$, which are not clock
constraints. By eliminating the existential quantifiers, we obtain the equivalent formula

\[
\begin{align*}
\forall (pc = \text{far} \land pc' = \text{near} \land x' = 0 \land \text{arrive}' \neq \text{arrive}) \\
\forall (pc = \text{far} \land pc' = \text{far} \land x' \geq x \land \text{arrive}' = \text{arrive}) \\
\forall (pc = \text{near} \land x \geq 3 \land pc' = \text{gate} \land x' = 0 \land \text{arrive}' = \text{arrive}) \\
\forall (pc = \text{near} \land pc' = \text{far} \land x' = x \land \text{arrive}' = \text{arrive}) \\
\forall (pc = \text{gate} \land x \geq 2 \land pc' = \text{far} \land x' = x \land \text{arrive}' = \text{arrive}) \\
\forall (pc = \text{gate} \land pc' = \text{gate} \land x \leq x' \leq 5 \land \text{arrive}' = \text{arrive}) \\
\forall (pc = \text{gate} \land pc' = \text{gate} \land x \leq x' \leq 2 \land \text{arrive}' = \text{arrive}).
\end{align*}
\]

While this is a clock formula, this need not be the case. To see this, consider the propositional real-time module with two clocks, \(x\) and \(y\), and the single guarded delay \(true \rightarrow \text{wait}\). The transition predicate of this module is

\[
(\exists \delta \geq 0. x' = x + \delta \land y' = y + \delta)
\]

or equivalently, after quantifier elimination,

\[
x' \geq x \land x' - x = y' - y.
\]

The synchronization of the two clocks introduces a constraint between clock differences.

**Exercise 5.4** (T3) [Real-time transition predicates] (1) Given a real-time module \(P\), define the transition predicate of \(P\). (2) A clock difference formula is like a clock formula, only that atomic subformulas may have the form \(x - y = z - u\), for clock variables \(x\), \(y\), \(z\), and \(u\). Prove that for every propositional real-time module, the transition predicate is equivalent to a clock difference formula (use existential-quantifier elimination for \(\delta\)). (3) How expensive is it to construct the clock difference formula that is equivalent to the transition predicate of \(P\)?

Consider a propositional real-time module \(P\). Suppose you are given a clock formula \(p\) (over the unprimed variables \(X\), which contain both propositions and clocks) which defines a region \(\sigma\), and a clock difference formula \(q\) (over the unprimed and primed variables \(X \cup X'\)) which is equivalent to the transition predicate of \(P\). Then the region \(\text{pre}(\sigma)\) is defined by the formula \(r = (\exists X'. p[X := X'] \land q)\) (and the region \(\text{post}(\sigma)\) is defined by the formula \(r' = (\exists X. p \land q)[X' := X]\)). By Theorem 5.1, the formula \(r\) is again a clock formula. (What about \(r'\)?) This formula can be found by existential-quantifier elimination.

**Exercise 5.5** (T3) [Reals with addition] The first-order theory of the reals with addition contains all atomic formulas that are either boolean variables or have the form \(t \sim c\), where \(t\) is a sum of clock variables, \(\sim\) is either \(\leq\) or \(\geq\), and \(c\) is an integer constant. In particular, all clock difference formulas are quantifier-free formulas of the first-order theory of the reals with addition. (1) Show that this theory permits quantifier elimination; that is, for every formula there is
an equivalent quantifier-free one. What is the complexity of your quantifier-el
mination procedure? (2) Show that this theory has a decidable satisfiability
problem. What is the complexity class of the satisfiability problem?

Exercise 5.6 {P3} [Real-time reachability analysis] Give a symbolic forward
reachability algorithm for propositional real-time modules. Represent all regions
that are computed by your algorithms using clock formulas. Step your forward
algorithm through a proof that the train \textit{RealTimeTrain} is never in the gate
when the gate \textit{RealTimeGate} is closed. Here “stepping through a proof” means
to list the clock formulas for $\sigma^t$, $\text{post}(\sigma^t)$, $\text{post}^2(\sigma^t)$, etc., for the real-time
module \textit{RailRoadXing}.

Exercise 5.7 {T3} [Forward reachability analysis] Symbolic backward reacha-
bility analysis is guaranteed to terminate for propositional real-time modules,
because every region constructed by the algorithm is a block of the region equiva-
ience (which has only finitely many blocks). The same cannot be said for for-
ward analysis. (1) Give a simple propositional real-time module for which sym-
bolic forward reachability analysis does not terminate. (2) Modify the forward
algorithm so that it is guaranteed to terminate on all propositional real-time
modules.

Exercise 5.8 {P3} [Real-time mutual exclusion] If clocks are available, mutual
exclusion can be guaranteed in a quite simple way. Formalize the following
protocol as propositional real-time modules, and step a forward-reachability
algorithm through a proof that the protocol ensures mutual exclusion. In your
protocol, assume that each assignment requires 2 time units. The two competing
processes share a variable $k$ whose value is initially 0, and always either 0, 1,
or 2. When the first (second) process wants to enter its critical section, it waits
until $k = 0$, then sets $k$ to 1 (resp. 2), then waits for 3 time units, then enters
its critical section if $k$ is still 1 (resp. 2); if the value of $k$ is no longer 1, the
process repeats the sequence starting from waiting until $k = 0$. Upon leaving
its critical section, the process resets $k$ to 0. (Since $k$ is a write-shared variable,
you must model it as a separate module.)

5.2.3 Difference-bound Matrices

For representing the regions of a propositional real-time module which can be
defined by clock formulas, a more efficient symbolic representation is based on
difference-bound matrices. For a region $R$, let $\text{timepre}(R)$ the set of all states
from which a state in $R$ can be reached by a time step; that is, $\text{timepre}(R) =
\{s \mid \exists t \in R. \exists \delta \geq 0. t = s + \delta\}$. For a clock formula $p$, let $\text{timepre}(p)$ be the
formula $(\exists \delta \geq 0. p + \delta)$, where $p + \delta$ is obtained from $p$ by replacing each clock
variable $x$ with $x + \delta$. Define $\text{timepost}$ similarly.
Exercise 5.9 \{T3\} [Clock formulas] (1) Prove that the clock constraints are closed under timepre, timepost, and existential-quantifier elimination, and have a decidable satisfiability problem. What is the cost of each operation? (2) Repeat the exercise for clock formulas.

Suppose you are given a clock formula \( p \) (over the unprimed variables \( X \), which contain both propositions and clocks) which defines a region \( \sigma \), a clock formula formula \( q \) (over the unprimed and primed variables \( X \cup X' \)) which defines the discrete transitions of \( P \), and a clock formula formula \( r \) (over the unprimed variables \( X \)) which defines the time invariant of \( P \) (i.e., the disjunctions of all guards of guarded delays). Then the region pre(\( \sigma \)) is defined by the clock formula \( r = (\exists X'. p[X := X'] \land q) \lor (\text{timepre}(p) \land r) \), and the region post(\( \sigma \)) is defined by the clock formula \( r = (\exists X. p \land q)[X' := X] \lor (\text{timepost}(p) \land r) \). This, together with the previous exercise, gives us symbolic backward and forward reachability algorithms which do not rely on arbitrary formulas of the first-order theory of the reals with addition, but only on clock formulas. And for clock formulas, there is an efficient symbolic representation.

Exercise 5.10 \{T3\} [Difference-bound matrices (DBMs)] Represent clock constraints with \( n \) clocks by integer square matrices with \( n + 1 \) rows and \( n + 1 \) columns. For two clocks \( x \) and \( y \), the \((x, y)\) entry contains a tight upper bound on the clock difference \( x - y \) (or \( \infty \), if there is no such upper bound). The \((x, n+1)\) entry contains a tight upper bound on the value of \( x \), and the \((n+1, x)\) entry contains a tight upper bound on \(-x\). Every matrix entry also contains a “bit” indicating if the bound is strict (\(<\)) or weak (\(\leq\)). (1) Is this representation canonical? If not, how would you make it canonical? (2) Give algorithms for computing conjunction, equivalence checking, satisfiability checking, renaming, existential-quantifier elimination (for clock variables), timepre and timepost on the matrix representation of clock constraints. What is the cost of each operation?

Exercise 5.11 \{T3\} [Combining boolean state and DBMs] Devise a “semi-symbolic” representation for clock formulas. Clock formulas result from clock constraint by adding both disjunction and propositions. Every clock formula can be thought of as a set of boolean states (valuations for the propositions), and for each boolean state, a set (disjunction) of clock constraints represented by DBMs. This representation is called semi-enumerative, because boolean state is represented enumeratively and only clock state is represented symbolically (by DBMs). Give algorithms for the boolean operations, satisfiability checking, implication and equivalence checking, renaming, existential-quantifier elimination (for both propositions and clocks), timepre and timepost on your representation of clock formulas. You may use the algorithms from the previous exercise as black-box subroutines. What is the cost of each operation?
**Exercise 5.12** [T4] [Combining BDDs and DBMs] Devise a “fully-symbolic” representation for clock formulas. Represent the boolean part of a clock formula as a BDD such that each leave does not point to 0 or 1, but to a set (disjunction) of BDMs. As in the previous exercise, give algorithms for the boolean operations, satisfiability checking, implication and equivalence checking, renaming, existential-quantifier elimination (for both propositions and clocks), Pre and Post on your representation of clock formulas. What is the cost of each operation?