Model Checking Real Time Systems (Lecture 7)

Analysis of Software Artifacts

Agenda

- overview of model checking real time systems
- introduce real time CTL or RTCTL
- introduce quantitative analysis

Why verify real-time systems?

- several applications require predictable response times
- to function correctly
- some applications are
  - controllers for aircraft
  - industrial machinery
  - robots
- errors can have catastrophic effects

Rate monotonic scheduling (RMS)

- powerful tool for analyzing real-time systems
- simple to use and provides useful information
- limitations on types of processes
  - periodicity
  - synchronization

Some papers on RMS

- M.G. Harbour, M.H. Klein, and J.P. Lehoczky, Timing analysis for fixed-priority of hard real-time systems, *IEEE Transactions on Software Engineering*, 20(1), 1994.
- J.P.Lehoczky, L. Sha, J.K. Strosnider, and H. Tokuda, Fixed priority scheduling theory for hard real-time systems, In *Foundations of Real-Time Computing-Scheduling and Resource Management*, Kluwer Academic Publishers, 1991.

Real time model checking

- no restrictions on system being specified
- use real time version of CTL
- much harder than RMS
- for certain types of systems, does not scale

Dense versus discrete time

- *dense* time uses real numbers to represent time
- *discrete* time uses integers to represent time
- analysis for dense time is very hard
- we will only cover discrete time

Real time CTL (RTCTL)

- regular CTL has *no notion of time*
- $\mathbf{EF}(f)$ says that sometime in the future f
- will become true, but does not say when
- only way to talk about time is using the $\mathbf{X}$ operator

Example

- If in a state a transaction T starts (denoted by $T.started$), then
- it always finishes in the next 3 cycles
- finishing a transaction is denoted by $T.finished$

RTCTL path operators

- consider a path π

$$s_0, s_1, \cdots, s_i, \cdots$$

- $f\mathbf{U}_{[a,b]}g$ is true on a path π if and only if
  - for some i, $a \leq i \leq b$, $s_i \models g$
  - and for all $j < i$, $s_j \models f$
- g becomes true somewhere in the time interval $[a, b]$
- f is true until g becomes true

Points to notice

- each transition takes *one unit of time*
- how can one model transitions that take more than one unit of time?
  - model them as several unit-time transitions

$\mathbf{G}_{[a,b]}$ path operator

- consider a path π

$$s_0, s_1, \cdots, s_i, \cdots$$

- $\mathbf{G}_{[a,b]} f$ is true on a path π if and only if
  - for all i such that $a \leq i \leq b$, $s_i \models g$
- f is true in the time interval $[a, b]$

Example revisited

- a transaction started always finishes with next three time cycles

$$\mathbf{AG}(T.started \rightarrow \mathbf{AF}_{[0,3]}(T.\mathit{finished}))$$

- represent $\mathbf{F}_{[a,b]}$ in terms $\mathbf{U}_{[a,b]}$

Specification patterns revisited

- $\mathbf{EF}_{[0,a]}(Started \wedge \neg Ready)$
- $\mathbf{AG}(Req \rightarrow \mathbf{AF}_{[0,a]}(Ack))$
- $\mathbf{AG}(\mathbf{AF}_{[0,a]}(DeviceEnable))$
- $\mathbf{AG}(\mathbf{EF}_{[0,a]}(Restart))$

Quantitative timing analysis

- provide information on how much a system deviates from its expected performance
- extremely useful in fine-tuning the system
- identify bottlenecks in your system, i.e., slow operations

Minimum Delay Analysis

- **inputs:** two sets of states, *start* and *final*
- **returns**
  - shortest path between a state in *start*
  - to a state in *final*
  - return ∞ if no such path exists
- `MIN(T.started,T.finished)`

Maximum Delay Analysis

- **inputs:** two sets of states, *start* and *final*
- **returns**
  - longest path between a state in *start*
  - to a state in *final*
  - return ∞ if there is an infinite
  - path from a state in *start* that never
  - reaches *finish*
- `MAX(T.started,T.finished)`

Condition counting

- condition counting measures how many times a given
- condition is true on a path
- earlier measures strictly based on path length

Minimum condition counting

- **inputs**
  - set of starting states: *start*
  - set of final states: *final*
  - a condition: *cond*
- **output:** minimum times condition *cond* is true along a path from *start* to *final*
- `MIN(T.started,T.finished,T.idle)`

Maximum condition counting

- **inputs**
  - set of starting states: *start*
  - set of final states: *final*
  - a condition: *cond*
- **output:** maximum times condition *cond* is true along a path from *start* to *final*
- `MAX(T.started,T.finished,T.idle)`

Round robin scheduling

- assume that there are n processes $P_0, \cdots, P_{n-1}$
- each process has the following four states
  - `idle` (process is not doing anything)
  - `ready` (process is ready to run)
  - `running` (process is running)
- scheduler picks a process in state `ready` to run
- we will now describe the round-robin policy

Round robin scheduling

- keep a variable *last*
- initial value of *last* is 0
- some processes are ready to run
- scheduler scans the processes in the following order

$$last, (last + 1) \bmod n, \cdots, (last + n - 1) \bmod n$$

Round robin scheduling

- pick the first process that is scanned
- and is in the state `ready` and schedule it
- let the process that is picked be P_i
- set variable *last* to i

Is Round Robin Scheduling Fair?

- is it possible that a process is in the
- state `ready` and never gets to run?
- i claim this is not possible. *Why?*