Modeling and Analyzing Real-Time Systems in Rewriting Logic (I)

Peter Ölveczky

University of Oslo

April 29, 2010

Based on joint work with José Meseguer

Real-Time Maude

Object-Oriented Specification

Example: RTT Protocol

Example: EDF Scheduling

Completeness

Lecture Series

Today: Specifying and analyzing real-time systems in Real-Time Maude

Tuesday: Context and applications

Real-Time Maude

Object-Oriented Specification

Example: RTT Protocol

Example: EDF Scheduling

Completeness

Content

Real-Time Maude

Object-Oriented Specification

Example: RTT Protocol

Example: EDF Scheduling

Soundness and Completeness of Real-Time Maude Analyses

Real-Time Systems

Real-time system: duration of/between events affects system functionality

Examples:

- communication protocols
- avionics systems
- medical devices
- automotive embedded software
- scheduling algorithms
- ...

Real-Time Maude

Object-Oriented Specification

Example: RTT Protocol

Example: EDF Scheduling

Completeness

Real-Time Rewrite Theories

- Instantaneous change modeled by ordinary rewrite rules
- Time elapse modeled by tick rewrite rules
  - ensure uniform time elapse in all system components
  - state has the form {t}
  - tick rewrite rules have the form

$$\{t\} \longrightarrow \{t'\} \;\; \textit{in time } \tau \textbf{ if } \textit{cond}$$

  - duration expression τ may contain variables

Real-Time Maude

Real-Time Maude extends Full Maude to model, simulate, and model check real-time rewrite theories

- parametric time domain
- predefined time domains in modules NAT-TIME-DOMAIN, NAT-TIME-DOMAIN-WITH-INF, POSRAT-TIME-DOMAIN, POSRAT-TIME-DOMAIN-WITH-INF
- timed modules (`tmod ... endtm`) and timed object-oriented modules (`tomod ... endtom`)
- tick rules written

  `crl [l] : {`t`} => {`t'`} in time` τ `if` *condition*

Dense Time

To "cover" entire dense time domain (intervals), tick rules have the form

```
crl [l] : {t}  =>  {t'}  in time  X
          if [X <= f(t) and] condition
```

where `X` is a variable of sort *Time* that does not occur in t

- tick rule not executable
- execute with time sampling strategies:
  - advance time by default value Δ
  - advance time as much as possible ("event-driven simulation")
- analysis incomplete: all possible behaviors not covered

Example: "Retrograde" Clock

Clock that shows time:

- state: `{clock(`r`)}` or `{stopped-clock(`r`)}`
- dense time domain
- clock can stop at any time
- retrograde clock: `clock(24)` must be reset to `clock(0)`

Modeling and Analyzing Real-Time Systems in Rewriting Logic (I)

Real-Time Maude

Object-Oriented Specification

Example: RTT Protocol

Example: EDF Scheduling

Completeness

Real-Time Maude Specification

```
(tmod DENSE-CLOCK is pr POSRAT-TIME-DOMAIN .
  ops clock stopped-clock : Time -> System .
  vars R R' : Time .

  crl [tickWhenRunning] :
      {clock(R)} => {clock(R + R')} in time R'
        if  R' <= 24 - R  [nonexec] .

  rl [tickWhenStopped] :
     {stopped-clock(R)} => {stopped-clock(R)} in time R'
                           [nonexec] .

  rl [reset] :   clock(24) => clock(0) .

  rl [batteryDies] :  clock(R) => stopped-clock(R) .
endtm)
```

Formal Analysis in Real-Time Maude

Real-Time Maude's analysis commands extend those of Maude and add some new ones

- all commands take selected time sampling strategy into account when executing time-nondeterministic tick rules
- tick rule applications with time elapse 0 not executed
- Real-Time Maude extends Full Maude: commands enclosed by parentheses
- two "versions" of Maude commands:
  - with time bounds:
    - consider behaviors up to the given time bound
    - finitely many states reachable
  - without time bounds:
    - finiteness of reachable state space preserved

Formal Analysis in Real-Time Maude: Simulation

Simulate system up to a certain duration:

`(tfrew` $[n]$ t `in time <=` τ `.)`

`(tfrew` $[n]$ t `with no time limit .)`

Modeling and Analyzing Real-Time Systems in Rewriting Logic (I)

Real-Time Maude

Object-Oriented Specification

Example: RTT Protocol

Example: EDF Scheduling

Completeness

Formal Analysis in Real-Time Maude: Reachability

```
(tsearch [n] t =>* pattern [such that cond]
    in time <= time bound .)

(tsearch [n] t =>* pattern [such that cond]
    in time >= time bound .)

(tsearch [n] t =>* pattern [such that cond]
    in time-interval between >= lb and <= ub .)

(utsearch [n] t =>* pattern [such that cond] .)
```

and with the other arrows and other comparison operators

Formal Analysis in Real-Time Maude: LTL Model Checking

`(mc` *t* `|=t` *formula* `in time <=` *time bound* `.)`

`(mc` *t* `|=u` *formula* `.)`

- Module `TIMED-MODEL-CHECKER` must be included
- Time-bounded model checking allows clocked propositions
- Metric LTL model checking (work in progress)
  - bounded response ($\Box(p \rightarrow \Diamond_{\leq t} q)$) and
  - minimum separation ($\Box(p \rightarrow (p\ W\ \Box_{\geq t} \neg p))$)

  for flat object-oriented systems

Clock Example: Simulation

Define time sampling:

```
Maude> (set tick def 1 .)
```

Rewriting simulates one possible behavior:

```
Maude> (trew {clock(0)} in time <= 100 .)

Result ClockedSystem :
  {stopped-clock(24)} in time 100
```

Clock Example: Search

- Can `{clock(25)}` be reached?

```
(utsearch [1] {clock(0)} =>* {clock(25)} .)
```

- State `{clock(1/2)}` not found due to time sampling:

```
(utsearch [1] {clock(0)} =>* {clock(1/2)} .)
```

Clock Example: Defining Propositions

- `clock-dead`: holds for stopped clocks

```
op clock-dead : -> Prop [ctor] .
vars R R' : Time .
eq {stopped-clock(R)} |= clock-dead = true .
```

- `clock-is(`*r*`)`: working clock shows *r*

```
op clock-is : Time -> Prop [ctor] .
eq {clock(R)} |= clock-is(R') = (R == R') .
```

- `clockEqualsTime`: working clock shows elapsed time

```
op clockEqualsTime : -> Prop [ctor] .
eq {clock(R)} in time R' |= clockEqualsTime
      = (R == R') .
```

Modeling and Analyzing Real-Time Systems in Rewriting Logic (I)

Real-Time Maude

Object-Oriented Specification

Example: RTT Protocol

Example: EDF Scheduling

Completeness

Clock Example: LTL Model Checking

- Clock never shows 25:

```
(mc {clock(0)} |=u [] ~ clock-is(25) .)
```

- Clock shows elapsed time until time 24, or until it stops:

```
(mc {clock(0)} |=t clockEqualsTime U
                   (clock-is(24) \/ clock-dead)
          in time <= 100 .)
```

Modeling and Analyzing Real-Time Systems in Rewriting Logic (I)

Real-Time Maude

Object-Oriented Specification

Example: RTT Protocol

Example: EDF Scheduling

Completeness

Implementation

- Real-Time Maude implemented in Maude as an extension of Full Maude
- (timed module M, Real-Time Maude command C, time sampling strategy tss, time bound b)

  transformed into a pair

  (Maude module $\widehat{M}_b^{tss}$, Maude command $\widehat{C}$)

  for efficient execution in Maude
- For "timed" commands, the Maude representation makes explicit a "global clock": {t} `in time` r
- For "untimed" Real-Time Maude commands, the translation abstracts from global clock: {t}

Running Real-Time Maude

- Maude file `real-time-maude.maude` available from `http://www.ifi.uio.no/RealTimeMaude`
- Execution in the usual way:

```
linux> maude real-time-maude
...
Maude> (tmod ... endtm)

Maude> (set tick ... .)

Maude> (tsearch ... .)

Maude> (help .)

Maude> (help tsearch .)
```

Object-Oriented Specification Techniques

- Repetition:
  - classes: class C | $a_1 : s_1$, ..., $a_n : s_n$.
  - objects: < o : C | $a_1 : v_1$, ..., $a_n : v_n$ >
  - global state {*config*}, where *config* is a multiset of objects and messages
- Useful specification techniques for OO specifications:

```
var C : Configuration .   var T : Time .

crl [tick] : {C} => {timeEffect(C, T)} in time T
             if T <= mte(C) [nonexec] .

op timeEffect : Configuration Time -> Configuration
                                      [frozen (1)] .
vars NECF1 NECF2 : NEConfiguration .
eq timeEffect(none, T) = none .
eq timeEffect(NECF1 NECF2, T) =
     timeEffect(NECF1, T)  timeEffect(NECF2, T) .
```

Object-Oriented Specification Techniques (cont.)

```
op mte : Configuration -> TimeInf [frozen (1)] .
eq mte(none) = INF .
eq mte(NECF1  NECF2) = min(mte(NECF1), mte(NECF2)) .
```

- These functions must be defined for messages and single objects

Object-Oriented Specification Techniques (cont.)

- Local clocks are attributes of sort `Time`
- Timers are attributes of sort `TimeInf`
  - denotes time until timer expires
  - value = `INF` means timer is turned off
  - when timer is 0, time advance should be blocked
- `timeEffect` increases clocks and decreases timers:

```
vars T1 T2 T3 : Time .   var TI : TimeInf .

eq timeEffect(< O : C | clock : T, timer : TI >, T2) =
     < O : C | clock : T + T2, timer : TI monus T2 > .
```

- Time advance must stop when timer expires:

```
eq mte(< O : C | timer : TI >) = TI .
```

Object-Oriented Specification Techniques (cont.)

Message m with delay [at least] t represented by $\texttt{dly}(m, t)$

```
sort DlyMsg .
subsorts Msg < DlyMsg < NEConfiguration .
op dly : Msg Time -> DlyMsg [ctor right id: 0] .
```

- "ripe" message has form m
- ```
  var M : Msg .
  eq timeEffect(dly(M, T), T2) = dly(M, T monus T2) .
  ```
- if ripe messages must be read urgently:

  ```
  eq mte(dly(M, T)) = T .
  ```
- if messages can be read any time they are ripe:

  ```
  eq mte(DM:DlyMsg) = INF .
  ```
  ```

Example: Simple RTT Protocol

- Find the round trip time to a given neighbor
- Mimimum messaging delay is `MIN-TRANS-DLY`
- If received RTT value smaller than `MAX-RTT`, then record it
- Restart procedure every `MAX-RTT` time units, until OK RTT value found

Example: Simple RTT Protocol

```
(tomod RTT is protecting NAT-TIME-DOMAIN-WITH-INF .
  op MAX-RTT : -> Time .          eq MAX-RTT = 6 .
  op MIN-TRANS-DLY : -> Time .    eq MIN-TRANS-DLY = 1 .

  class Node | clock : Time, rtt : TimeInf,
               nbr : Oid, timer : TimeInf .

  msgs rttReq rttResp : Oid Oid Time -> Msg .
  msg  findRtt : Oid -> Msg .                    --- start a run

  sort DlyMsg .       subsorts Msg < DlyMsg < NEConfiguration .
  op dly : Msg Time -> DlyMsg [ctor right id: 0] .

  vars O O' : Oid .    vars R R' : Time .    var TI : TimeInf .
  vars NEC NEC' : NEConfiguration .    var C : Configuration .
  var DM : DlyMsg .    var M : Msg .
```

Example: Simple RTT Protocol

```
--- start a session, and set timer:
rl [startSession] :
   findRtt(O)
   < O : Node | clock : R, nbr : O' >
  =>
   < O : Node | timer : MAX-RTT >
   dly(rttReq(O', O, R), MIN-TRANS-DLY) .


--- respond to request:
rl [rttResponse] :
   rttReq(O, O', R)
   < O : Node | >
  =>
   < O : Node | >
   dly(rttResp(O', O, R), MIN-TRANS-DLY) .
```

Modeling and Analyzing Real-Time Systems in Rewriting Logic (I)

Real-Time Maude

Object-Oriented Specification

Example: RTT Protocol

Example: EDF Scheduling

Completeness

Example: Simple RTT Protocol (cont.)

```
--- receiv resp; if within time MAX-RTT: record rtt value
--- and turn off timer; otherwise: ignore response
rl [treatRttResp] :
   rttResp(O, O', R)
   < O : Node | clock : R' >
  =>
   if (R' monus R) < MAX-RTT then    --- ok value
      < O : Node | rtt : (R' monus R), timer : INF >
   else < O : Node | > fi .

--- start new round and reset timer when timer expires:
rl [tryAgain] :
   < O : Node | timer : 0, clock : R, nbr : O' >
  =>
   < O : Node | timer : MAX-RTT >
   dly(rttReq(O', O, R), MIN-TRANS-DLY) .

--- time should not advance beyond expiration of a timer:
crl [tick] :
    {C} => {timeEffect(C, R)} in time R
   if R <= mte(C) [nonexec] .
```


Modeling and Analyzing Real-Time Systems in Rewriting Logic (I)

Real-Time Maude

Object-Oriented Specification

Example: RTT Protocol

Example: EDF Scheduling

Completeness

Example: Simple RTT Protocol (cont.)

```
  op timeEffect : Configuration Time -> Configuration
                                          [frozen (1)] .
  eq timeEffect(none, R) = none .
  eq timeEffect(NEC NEC', R) =
       timeEffect(NEC, R) timeEffect(NEC', R) .
  eq timeEffect(< O : Node | clock : R, timer : TI >, R') =
          < O : Node | clock : R + R', timer : TI monus R' > .
  eq timeEffect(dly(M, R'), R) = dly(M, R' monus R) .

  op mte : Configuration -> TimeInf [frozen (1)] .
  eq mte(none) = INF .
  eq mte(NEC NEC') = min(mte(NEC), mte(NEC')) .
  eq mte(< O : Node | timer : TI >) = TI .
  eq mte(DM) = INF .
endtom)
```

Example: EDF Scheduling

- Set of periodic tasks $t_1, \ldots, t_n$, where t_i has
  - period p_i
  - execution time e_i in each period
- Preemption
- Task with earliest deadline has highest priority
- Schedulability problem: can each task t_i execute for time e_i in each round?
- Note: schedulability can be decided analytically for simple scheduling algorithms like basic EDF (schedulable if $\sum_i \frac{e_i}{p_i} \leq 1$)

Example: EDF Scheduling (cont.)

```
class Task | period : NzTime,
             exTime : NzTime,
             state : State,
             exTimeRemaining : Time,
             deadline : Time .

sort State .
ops executing waiting finished : -> State [ctor] .

op processor : -> Object [ctor] .
```

Example: EDF Scheduling (cont.)

When a task finishes its previous period (`deadline : 0`), it starts a new period

1. if `processor` available: grab processor and start executing
2. if other task executing and has later deadline: preempt executing task and start executing
3. if other task, with an earlier deadline, is executing: wait

Example: EDF Scheduling (cont.)

Start of new period, and processor available:

```
rl [startNewRound] :
   processor
   < O : Task | period : P, exTime : EXT, state : finished,
                exTimeRemaining : 0, deadline : 0 >
  =>
   < O : Task | state : executing,
                exTimeRemaining : EXT, deadline : P > .
```

Example: EDF Scheduling (cont.)

Start of new period, and either preempt executing task or go to waiting:

```
rl [startNewRound2] :
    < O : Task | period : P, exTime : EXT, state : finished,
                 exTimeRemaining : 0, deadline : 0 >
    < O' : Task | state : executing, deadline : T,
                  exTimeRemaining : NZT >
  =>
    if P < T    --- preempt
    then
      < O : Task | state : executing,
                   exTimeRemaining : EXT, deadline : P >
      < O' : Task | state : waiting >
    else
      < O : Task | state : waiting,
                   exTimeRemaining : EXT, deadline : P >
      < O' : Task | >
    fi .
```

Example: EDF Scheduling (cont.)

When a task finishes executing (`exTimeRemaining : 0`):

1. if other task(s) waiting: give processor to waiting task with earliest deadline
2. otherwise: release processor

Example: EDF Scheduling (cont.)

End of execution in period and someone waiting:

```
crl [endExAndSomeoneWaiting] :
    {< O : Task | state : executing, exTimeRemaining : 0 >
     < O' : Task | state : waiting, deadline : D >
     REST}
   =>
    {< O : Task | state : finished >
     < O' : Task | state : executing >
     REST}
    if D <= earliestDeadlineWaiting(REST) .
```

Example: EDF Scheduling (cont.)

End of execution in period and no one waiting:

```
crl [endExAndNoOneWaiting] :
    {< O : Task | state : executing, exTimeRemaining : 0 >
     REST}
   =>
    {< O : Task | state : finished >
     processor
     REST}
   if noOneWaiting(REST) .
```

Example: EDF Scheduling (cont.)

Timed behavior:

- "standard" OO tick rule
- `mte` and `timeEffect` distribute over the objects in the configuration and must be defined for single objects:

```
eq timeEffect(< O : Task | state : S, exTimeRemaining : T1,
                           deadline : T2 >, T3)
   =
     < O : Task | exTimeRemaining : (if S == executing then
                                       T1 monus T3 else T1 fi,
                  deadline : T2 monus T3 > .

eq mte(< O : Task | state : S, exTimeRemaining : T1,
                    deadline : T2 >)
   = if S == executing then min(T1, T2) else T2 fi .

eq timeEffect(processor, T) = processor .
eq mte(processor) = INF .
```

EDF Scheduling: Schedulability Analysis

Deadline missed when a task has more execution time left than time until deadline:

```
(utsearch
  {processor
   < t1 : Task | exTime : 2, period : 5,
                 state : finished,
                 exTimeRemaining : 0, deadline : 0 >
   < t2 : Task | exTime : 4, period : 7,
                 state : finished,
                 exTimeRemaining : 0, deadline : 0 >}
  =>*
   {C:Configuration
    < O : Task | deadline : T1, exTimeRemaining : T2 >}
  such that T1 < T2 .)
```

- can also analyze multiprocessor systems

Modeling and Analyzing Real-Time Systems in Rewriting Logic (I)

Real-Time Maude

Object-Oriented Specification

Example: RTT Protocol

Example: EDF Scheduling

Completeness

Completeness

- Real-Time Maude analyses incomplete for dense time
  - formalism too general for "region graphs"
- Given conditions for soundness and completeness of maximal time sampling analyses
  - easy to check for OO systems
  - holds for interesting systems
  - completeness for dense time systems beyond timed automata

Main References

1. P. C. Ölveczky and J. Meseguer. *Semantics and Pragmatics of Real-Time Maude*. Higher-Order and Symbolic Computation, volume 20(1-2), 2007. (Main reference to Real-Time Maude)
2. P. C. Ölveczky and J. Meseguer. *Specification of real-time and hybrid systems in rewriting logic*. Theoretical Computer Science, vol 285(2), 2002. (Theoretical foundations)
3. P. C. Ölveczky. *Real-Time Maude 2.3 Manual*. http://www.ifi.uio.no/RealTimeMaude, 2007. (User manual)