SPECIFICATION AND ANALYSIS OF REAL-TIME SYSTEMS IN REAL-TIME MAUDE

Peter Ölveczky
(University of Oslo)

International School on Rewriting, Paris, July 5, 2019

CONTENT

FORMAL METHODS: KEY TRADE-OFF

expressiveness/modeling convenience $\longleftrightarrow$ analytic power

REWRITING LOGIC AND MAUDE

- modeling convenience!
- simple and intuitive
- any data type
- unbounded data structures
- simple model of concurrent objects
    - dynamic object/message creation/deletion
    - easy to define forms of communication
- hierarchical structures

- properties in general undecidable

- analysis by system execution

MAUDE AND REAL-TIME SYSTEMS

(How) can we model and analyze real-time systems in rewriting logic so that the advantages of Maude are maintained?

MODELING AND ANALYZING REAL-TIME SYSTEMS WITH REAL-TIME MAUDE

MOTIVATION

Cannot always abstract from time:

- fault-tolerant systems must discover message loss/node crash
  - impossible without time

MOTIVATION

Cannot always abstract from time:

- fault-tolerant systems must discover message loss/node crash
  - impossible without time
- time key parameter in many algorithms
  - e.g. to fine-tune performance

MOTIVATION

Cannot always abstract from time:

- fault-tolerant systems must discover message loss/node crash
  - impossible without time
- time key parameter in many algorithms
  - e.g. to fine-tune performance
- time key in most systems
  - toasters
  - cars, airplanes, ...
  - e-banking

MOTIVATION

Cannot always abstract from time:

- fault-tolerant systems must discover message loss/node crash
    - impossible without time
- time key parameter in many algorithms
    - e.g. to fine-tune performance
- time key in most systems
    - toasters
    - cars, airplanes, ...
    - e-banking
- scheduling algorithms

MOTIVATION

Cannot always abstract from time:

- fault-tolerant systems must discover message loss/node crash
  - impossible without time
- time key parameter in many algorithms
  - e.g. to fine-tune performance
- time key in most systems
  - toasters
  - cars, airplanes, ...
  - e-banking
- scheduling algorithms
- timed models enable reasoning about performance

MOTIVATION

Cannot always abstract from time:

- fault-tolerant systems must discover message loss/node crash
  - impossible without time
- time key parameter in many algorithms
  - e.g. to fine-tune performance
- time key in most systems
  - toasters
  - cars, airplanes, ...
  - e-banking
- scheduling algorithms
- timed models enable reasoning about performance
- timed properties important
  - airbag must deploy within 10ms of a crash

MAIN CHALLENGE

How to deal with dense time?

MAIN CHALLENGE

How to deal with dense time?

Why dense time ($\mathbb{R}_{\geq 0}$, $\mathbb{Q}_{\geq 0}$, ...)?

MAIN CHALLENGE

How to deal with dense time?

Why dense time ($\mathbb{R}_{\geq 0}$, $\mathbb{Q}_{\geq 0}$, ...)?

- "real" real-time system
- techniques useful for discrete-time systems
  - example: events take place approximately once per 10,000 ms

REAL-TIME MAUDE [ÖLVECZKY AND MESEGUER '04, '06, ÖLVECZKY'14]

Real-Time Maude:

- Extends Maude to model and analyze real-time systems
- Object-oriented modeling of distributed real-time systems
- Implemented in Maude as an extension of Full Maude
- `http://www.ifi.uio.no/RealTimeMaude`

REAL-TIME MAUDE MODELING

- Data types modeled by algebraic equational specification
  - parametric time domain
  - built-in time domains `NAT-TIME-DOMAIN-WITH-INF`, `POSRAT-TIME-DOMAIN-WITH-INF`, ...

REAL-TIME MAUDE MODELING

- Data types modeled by algebraic equational specification
  - parametric time domain
  - built-in time domains `NAT-TIME-DOMAIN-WITH-INF`, `POSRAT-TIME-DOMAIN-WITH-INF`, ...
- Instantaneous transitions modeled by rewrite rules

```
crl [l] : t => t' if cond
```

REAL-TIME MAUDE MODELING

- Data types modeled by algebraic equational specification
  - parametric time domain
  - built-in time domains NAT-TIME-DOMAIN-WITH-INF, POSRAT-TIME-DOMAIN-WITH-INF, ...
- Instantaneous transitions modeled by rewrite rules

crl [*l*] : *t* => *t′* if *cond*

- Time advance modeled explicitly by tick rewrite rules

crl [*l*] : {*t*} => {*t′*} in time τ if *cond*

  - global state has form {*t*}
  - ensures uniform time elapse

REAL-TIME MAUDE ANALYSIS

- Timed rewriting
  - simulate system to time T
- Timed reachability analysis
  - find states reachable in time interval
- LTL model checking
  - unbounded
  - time-bounded
    - finite reachable state space

REAL-TIME MAUDE ANALYSIS

- Timed rewriting
  - simulate system to time T
- Timed reachability analysis
  - find states reachable in time interval
- LTL model checking
  - unbounded
  - time-bounded
    - finite reachable state space
- Timed CTL model checking

REAL-TIME MAUDE ANALYSIS

- Timed rewriting
  - simulate system to time T
- Timed reachability analysis
  - find states reachable in time interval
- LTL model checking
  - unbounded
  - time-bounded
    - finite reachable state space
- Timed CTL model checking
- “Find earliest” and “find latest” ...

“Time sampling” discretization of dense time (see below)

MODELING IN REAL-TIME MAUDE

EXAMPLE: "RETROGRADE" CLOCK

- State: {clock(r)} or {stopped-clock(r)}
- Dense time domain
- Clock can stop at any time
- Retrograde clock: clock(12) must be reset to clock(0)

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' <= 12 - R  [nonexec] .

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

  rl [reset] :   clock(12) => clock(0) .

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

NEXT EXAMPLE: SINGLE IMPRECISE CLOCK

- One clock
- Imprecise: goes faster or slower than "real time"
  - given by its rate

MODELING SINGLE IMPRECISE CLOCK

```
(tomod SINGLE-SKEWED-CLOCK is pr POSRAT-TIME-DOMAIN .

  class Clock | running : Bool, time : Time, rate : PosRat .

  var C : Oid .  vars R R' : Time .  var RATE : PosRat .

  crl [tickRunning] :
      {< C : Clock | running : true, time : R, rate : RATE >}
    =>
      {< C : Clock | time : R + (RATE * R') >} in time R'
    if R' <= (12 - R) / RATE [nonexec] .

  rl [tickStopped] :
     {< C : Clock | running : false >}  =>  {< C : Clock | >}
      in time R'  [nonexec] .
```

MODELING SINGLE IMPRECISE CLOCK (CONT.)

```
  rl [reset] :
     < C : Clock | running : true, time : 12 >
  =>
     < C : Clock | time : 0 > .

  rl [break] :
     < C : Clock | running : true >
  =>
     < C : Clock | running : false > .
endtom)
```

ANOTHER EXAMPLE: MULTIPLE IMPRECISE CLOCKS

We now model many imprecise clocks.

- Instantaneous rules as before
- Single tick rule:

```
var CLOCKS : Configuration .   var R' : Time .

crl [tick] :
    {CLOCKS}  =>  {advanceTime(CLOCKS)} in time R'
    if R' <= maxTimeAdvance(CLOCKS) .
```

MULTIPLE IMPRECISE CLOCKS (CONT.)

```
vars CLOCKS CLOCKS' : Configuration .

op advanceTime : Configuration Time -> Configuration [frozen (1)]

ceq advanceTime(CLOCKS CLOCKS', R)
  = advanceTime(CLOCKS, R)  advanceTime(CLOCKS', R)
   if CLOCKS =/= none and CLOCKS' =/= none .

eq advanceTime(< C : Clock | running : true, time : R,
                             rate : RATE >, R')
 = < C : Clock | time : R + (R' * RATE) >

eq advanceTime(< C : Clock | running : false >, R')
 = < C : Clock | > .
```

MULTIPLE IMPRECISE CLOCKS (CONT.)

```
op maxTimeAdvance : Configuration -> TimeInf [frozen (1)] .

ceq maxTimeAdvance(CLOCKS CLOCK')
  = min(maxTimeAdvance(CLOCKS), maxTimeAdvance(CLOCKS'))
   if CLOCKS =/= none and CLOCKS' =/= none .

eq maxTimeAdvance(< C : Clock | running : true, time : R,
                                rate : RATE >)
 = (12 - R) / RATE .

eq maxTimeAdvance(< C : Clock | running : false >) = INF .
```

MULTIPLE IMPRECISE CLOCKS (CONT.)

Suitable initial state is

```
{< ap : Clock | running : true, time : 0, rate : 5/4 >
 < seiko : Clock | running : true, time : 0, rate : 1 >
 < casio : Clock | running : true, time : 0, rate : 99/100 >}
```

ANOTHER EXAMPLE: POPULATIONS

Many people (cont):

- Time passes uniformly for all living persons
    - everybody birthday same time

ANOTHER EXAMPLE: POPULATIONS

Many people (cont):

- Time passes uniformly for all living persons
  - everybody birthday same time
- Engagements, weddings, etc. instantaneous actions

ANOTHER EXAMPLE: POPULATIONS

Many people (cont):

- Time passes uniformly for all living persons
    - everybody birthday same time
- Engagements, weddings, etc. instantaneous actions
- Must ensure nobody older than 1000!

POPULATIONS (CONT.)

Engagement normal instantaneous rule:

```
crl [engagement] :
    < P : Person | age : N, status : single >
    < P' : Person | age : M, status : single >
  =>
    < P : Person | status : engaged(P') >
    < P' : Person | status : engaged(P) >
  if N > 15 and M > 15 .
```

POPULATIONS: TICK RULE

Time can pass by many milliseconds/days/years up to age limit:

```
var PERSONS : Configuration .  var R : Time .

crl [tick] :
    {PERSONS}  =>  {advanceAge(PERSONS, R)}  in time R
   if R <= maxTimeAdvance(PERSONS) [nonexec] .
```

POPULATIONS: TICK RULE

Time can pass by many milliseconds/days/years up to age limit:

```
var PERSONS : Configuration .  var R : Time .

crl [tick] :
    {PERSONS}  =>  {advanceAge(PERSONS, R)}  in time R
   if R <= maxTimeAdvance(PERSONS) [nonexec] .
```

- `advanceAge` increases age of each living person by `R`
- `maxTimeAdvance` ensures noone gets older than 1000

Exercise

Define the functions `advanceAge` and `maxTimeAdvance`

OO SPECIFICATIONS

Generalizing: OO system has many instantaneous rules and usually one tick rule

```
var STATE : Configuration .  var R : Time .

crl [tick] :
    {STATE}  =>  {timeEffect(STATE, R)}  in time R
   if R <= maxTimeAdvance(STATE) .
```

- `timeEffect(STATE, R)` defines how the elapse of time `R` affects the state `STATE`
- `maxTimeAdvance(STATE)` defines how much time can advance before something must happen

OO SPECIFICATIONS II

timeEffect and maxTimeAdvance distribute over objects and messages in a configuration:

```
vars C1 C2 : Configuration .

ceq timeEffect(C1 C2, R)
  = timeEffect(C1, R)  timeEffect(C2, R)
 if C1 =/= none and C2 =/= none .

eq timeEffect(none, R) = none .

ceq maxTimeAdvance(C1 C2)
  = min(maxTimeAdvance(C1), maxTimeAdvance(C2))
 if C1 =/= none and C2 =/= none .

eq maxTimeAdvance(none) = INF .
```

These functions must be defined for single objects and messages

MODELING MESSAGE DELAYS

Message delays (communication time):

1. Message delay any value $\in [0, \infty]$
2. Message delay exactly Δ
3. Message delay at least Δ
4. Message delay between 0 and Δ
5. Message delay Δ_1 and Δ_2

MODELING MESSAGE DELAYS: CASE 1

Case 1: delay could be anything:

- sender sends standard message
- receiver reads standard message
- `eq timeEffect(msg, R) = msg .`
- `eq maxTimeAdvance(msg) = INF .`

MODELING MESSAGE DELAYS: CASES 2 AND 3

Case 2: Message delay exactly Δ:

MODELING MESSAGE DELAYS: CASES 2 AND 3

Case 2: Message delay exactly Δ:

“Delayed” message `dly(msg, r)` where r is remaining delay

```
op dly : Msg Time -> DlyMsg [ctor right id: 0]
```

`dly(m, 0)` identical to m

MODELING MESSAGE DELAYS: CASES 2 AND 3

Case 2: Message delay exactly Δ:

"Delayed" message `dly(msg, r)` where r is remaining delay

```
op dly : Msg Time -> DlyMsg [ctor right id: 0]
```

`dly(m, 0)` identical to m

- sender sends `dly(msg, Δ)`

MODELING MESSAGE DELAYS: CASES 2 AND 3

Case 2: Message delay exactly Δ:

"Delayed" message `dly(msg, r)` where r is remaining delay

```
op dly : Msg Time -> DlyMsg [ctor right id: 0]
```

`dly(m, 0)` identical to m

- sender sends `dly(msg, Δ)`
- receiver reads *msg*

MODELING MESSAGE DELAYS: CASES 2 AND 3

Case 2: Message delay exactly Δ:

"Delayed" message `dly(msg, r)` where r is remaining delay

```
op dly : Msg Time -> DlyMsg [ctor right id: 0]
```

`dly(m, 0)` identical to m

- sender sends `dly(msg, Δ)`
- receiver reads *msg*
- `eq timeEffect(dly(M, R), R') = dly(M, R monus R')`

MODELING MESSAGE DELAYS: CASES 2 AND 3

Case 2: Message delay exactly Δ:

“Delayed” message `dly(msg,` *r*`)` where *r* is remaining delay

```
op dly : Msg Time -> DlyMsg [ctor right id: 0]
```

`dly(`*m*`, 0)` identical to *m*

- sender sends `dly(`*msg*`,` Δ`)`
- receiver reads *msg*
- `eq timeEffect(dly(M, R), R’) = dly(M, R monus R’)`
- `eq maxTimeAdvance(dly(M, R)) = R`

MODELING MESSAGE DELAYS: CASES 2 AND 3

Case 2: Message delay exactly Δ:

"Delayed" message `dly(msg, r)` where r is remaining delay

```
op dly : Msg Time -> DlyMsg [ctor right id: 0]
```

`dly(m, 0)` identical to m

- sender sends `dly(msg, Δ)`
- receiver reads *msg*
- `eq timeEffect(dly(M, R), R') = dly(M, R monus R')`
- `eq maxTimeAdvance(dly(M, R)) = R`

How do we model Case 3 (delay at least Δ)?

MODELING MESSAGE DELAYS: CASES 2 AND 3

Case 2: Message delay exactly Δ:

"Delayed" message `dly(msg, r)` where *r* is remaining delay

```
op dly : Msg Time -> DlyMsg [ctor right id: 0]
```

`dly(m, 0)` identical to *m*

- sender sends `dly(msg, Δ)`
- receiver reads *msg*
- `eq timeEffect(dly(M, R), R') = dly(M, R monus R')`
- `eq maxTimeAdvance(dly(M, R)) = R`

How do we model Case 3 (delay at least Δ)?

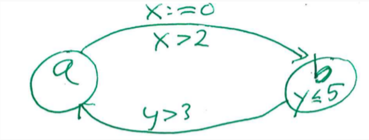
Only change: `eq maxTimeAdvance(dly(M, R)) = INF`

MORE MESSAGE DELAY

Exercise

How do we model Cases 4 and 5?

REPRESENTING TIMED AUTOMATA



State: {*location*, *xVal*, *yVal*}

```
vars X Y R : Time .

rl [tick-a] : {a, X, Y} => {a, X+R, Y+R} in time R [nonexec] .

crl [tick-b] :  {b, X, Y}  =>  {b, X+R, Y+R} in time R
                if R <= 5-Y [nonexec] .

crl [ab] :  {a, X, Y}  =>  {b, 0, Y} if X > 2 .

crl [ba] :  {b, X, Y}  =>  {a, X, Y} if Y > 3 .
```

ANALYSIS IN REAL-TIME MAUDE

REAL-TIME MAUDE ANALYSIS

- Timed rewriting
  - simulate system to time T
- Timed reachability analysis
- LTL model checking
  - unbounded
  - time-bounded
- Timed CTL model checking

INTERNAL REPRESENTATION

- Uses Maude rewriting, search and LTL model checking
- Internal representation of states:
    1. `{`t`} in time` *totalDuration*
        - time-bounded analysis
        - reachable state space infinite

INTERNAL REPRESENTATION

- Uses Maude rewriting, search and LTL model checking
- Internal representation of states:
  1. `{`*t*`} in time` *totalDuration*
     - time-bounded analysis
     - reachable state space infinite
  2. `{`*t*`}`
     - "system time" / "duration" abstracted away
     - does not add states
     - analysis without time bounds

RETROGRADE WATCH (AGAIN)

```
(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' <= 12 - R  [nonexec] .

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

  rl [reset] :   clock(12) => clock(0) .

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

MAIN CHALLENGE

How to deal with dense time?

TIME SAMPLING

- Tick rules “cover” dense time domain
  - not executable

TIME SAMPLING

- Tick rules “cover” dense time domain
  - not executable
- “On-the-fly discretization:” time sampling strategies
  - advance time by default value Δ
  - advance time as much as possible (“event-driven simulation”)

TIME SAMPLING

- Tick rules "cover" dense time domain
  - not executable
- "On-the-fly discretization:" time sampling strategies
  - advance time by default value Δ
  - advance time as much as possible ("event-driven simulation")
- Analysis incomplete: all behaviors not covered

SIMULATION

Define time sampling:

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

- analysis w.r.t. this strategy

SIMULATION

Define time sampling:

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

- analysis w.r.t. this strategy

Rewriting simulates one possible behavior:

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

SIMULATION

Define time sampling:

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

- analysis w.r.t. this strategy

Rewriting simulates one possible behavior:

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

Result ClockedSystem :
  {stopped-clock(12)} in time 100
```

TIME-BOUNDED SEARCH

- Can `{clock(8)}` be reached in time $\in [23, 25]$?

```
Maude> (tsearch {clock(0)}  =>*  {clock(8)}
            in time-interval between >= 23 and <= 25 .)
```

TIME-BOUNDED SEARCH

- Can `{clock(8)}` be reached in time $\in [23, 25]$?

```
Maude> (tsearch {clock(0)}  =>*  {clock(8)}
            in time-interval between >= 23 and <= 25 .)

No solution
```

TIME-BOUNDED SEARCH

- Can `{clock(8)}` be reached in time $\in [23, 25]$?

```
Maude> (tsearch {clock(0)}  =>*  {clock(8)}
         in time-interval between >= 23 and <= 25 .)

No solution
```

- ... in time ≥ 31?

```
Maude> (tsearch {clock(0)}  =>*  {clock(8)}
         in time >= 31 .)
```

TIME-BOUNDED SEARCH

- Can `{clock(8)}` be reached in time $\in [23, 25]$?

```
Maude> (tsearch {clock(0)}  =>*  {clock(8)}
          in time-interval between >= 23 and <= 25 .)

No solution
```

- ... in time ≥ 31?

```
Maude> (tsearch {clock(0)}  =>*  {clock(8)}
          in time >= 31 .)

Solution 1
TIME_ELAPSED:Time --> 32
```

SEARCH

- Can `{clock(13)}` be reached?

```
(utsearch [1] {clock(0)} =>* {clock(13)} .)
```

SEARCH

- Can `{clock(13)}` be reached?

  `(utsearch [1] {clock(0)} =>* {clock(13)} .)`

- State `{clock(13)}` not found:

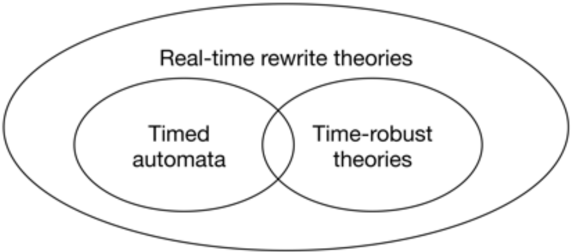
  `(utsearch [1] {clock(0)} =>* {clock(1/2)} .)`

SOUND AND COMPLETE UNTIMED ANALYSIS

- Time sampling discretization of dense time
- All behaviors not covered
  - analysis not sound/complete
  - states reached and LTL counterexamples are correct

SOUND AND COMPLETE UNTIMED ANALYSIS

- Time sampling discretization of dense time
- All behaviors not covered
  - analysis not sound/complete
  - states reached and LTL counterexamples are correct
- Maximal time sampling analysis sound and complete for time-robust models [Ölveczky-Meseguer'06]
  - events at given times
  - atomic propositions not modified by ticks
- Sound/complete analysis for systems beyond timed automata



TIMED TEMPORAL LOGIC

- So far: untimed properties/temporal logic
  - "the airbag must eventually deploy after crash detected"
  - "BO eventually closes G"

TIMED TEMPORAL LOGIC

- So far: untimed properties/temporal logic
  - “the airbag must eventually deploy after crash detected”
  - “BO eventually closes G”
- Timed temporal logics
  - “the airbag must deploy within 10ms after crash”
  - “BO closes G within one year of inauguration”

REAL-TIME MAUDE'S TCTL MODEL CHECKER

- Explicit-state timed CTL model checker for Real-Time Maude
- TCTL: temporal operators with time intervals: $\exists\, \phi\, \mathcal{U}_{[r_1,r_2]}\, \phi'$
  - $\forall\Box\,(\mathit{crash} \implies \forall\Diamond_{\leq 10ms}\ \mathit{airbagDeployed})$
  - $\forall\Box\,((\mathit{inauguration}(BO) \wedge \mathit{open}(G)) \implies \forall\Diamond_{\leq one\ year}\ \mathit{closed}(G))$

REAL-TIME MAUDE'S TCTL MODEL CHECKER

- Explicit-state timed CTL model checker for Real-Time Maude
- TCTL: temporal operators with time intervals: $\exists\, \phi\, \mathcal{U}_{[r_1, r_2]}\, \phi'$
  - $\forall\Box\,(\mathit{crash} \Longrightarrow \forall\Diamond_{\leq 10ms}\, \mathit{airbagDeployed})$
  - $\forall\Box\,((\mathit{inauguration}(BO) \wedge \mathit{open}(G)) \Longrightarrow \forall\Diamond_{\leq \mathit{one\ year}}\, \mathit{closed}(G))$
- `(mc-tctl {clock(6)}|= EF[<= than 8] clock-is(12) .)`

D. Lepri, E. Ábrahám, P.C. Ölveczky: Sound and complete timed CTL model checking of timed Kripke structures and real-time rewrite theories. Science of Computer Programming 99 (2015)

INTENDED SEMANTICS

What is the intended semantics of a Real-Time Maude model?

$\{clock(R)\} \to \{clock(R + R')\}$ **in time** R' **if** $R' \leq 12 - R$

INTENDED SEMANTICS

What is the intended semantics of a Real-Time Maude model?

$\{clock(R)\} \to \{clock(R + R')\}$ **in time** R' **if** $R' \leq 12 - R$

- Should $\forall \Diamond_{[1,2]}$ `True` hold from $\{clock(0)\}$?

INTENDED SEMANTICS

What is the intended semantics of a Real-Time Maude model?

$\{clock(R)\} \rightarrow \{clock(R + R')\}$ **in time** R' **if** $R' \leq 12 - R$

- Should $\forall \Diamond_{[1,2]}$ `True` hold from $\{clock(0)\}$?
- Pointwise semantics
  - only visited states into account
  - $\forall \Diamond_{[1,2]}$ `True` does not hold from $\{clock(0)\}$

INTENDED SEMANTICS

What is the intended semantics of a Real-Time Maude model?

$\{clock(R)\} \rightarrow \{clock(R + R')\}$ **in time** R' **if** $R' \leq 12 - R$

- Should $\forall \Diamond_{[1,2]}$ `True` hold from $\{clock(0)\}$?
- Pointwise semantics
  - only visited states into account
  - $\forall \Diamond_{[1,2]}$ `True` does not hold from $\{clock(0)\}$
- Continuous semantics
  - tick rule interpreted as representing continuous process
  - $\forall \Diamond_{[1,2]}$ `True` holds from $\{clock(0)\}$

SOUNDNESS AND COMPLETENESS

Soundness and completeness for maximal time sampling analyses of untimed TL do not carry over to timed CTL

- maximal time sampling analysis does not satisfy $\exists \Diamond_{[1,2]} \texttt{True}$
- ... or $\forall \Diamond_{[1,2]} \texttt{True}$

SOUND AND COMPLETE TCTL MODEL CHECKING

- Continuous and pointwise interpretation

SOUND AND COMPLETE TCTL MODEL CHECKING

- Continuous and pointwise interpretation
- Time-sampling-based sound and complete TCTL model checking for time-robust Real-Time Maude models
  - advance time by

$$\frac{gcd(numbers\ in\ formulas, max-tick\ durations)}{2}$$

TIMED CTL MODEL CHECKING (II)

- Not reducible to Maude model checking
- No counterexamples/witnesses

TIMED CTL MODEL CHECKING (II)

- Not reducible to Maude model checking
- No counterexamples/witnesses
- Crossing-the-Bridge benchmark comparison:

Initial state	TSMV	Real-Time Maude (pointwise)	Real-Time Maude (continuous)	RED 7.0
init(1)	0.074	0.149	1.266	0.429
init(10)	0.148	0.168	0.999	0.408
init(100)	1.443	0.168	1.012	0.404
init(1000)	57.426	0.327	1.014	0.426
init+(2)	0.191	0.746	6.864	1.044
init+(4)	0.280	1.772	17.752	2.153
init+(8)	0.759	5.227	57.580	16.912
init+(12)	1.080	11.198	129.957	79.319
init+(16)	1.515	19.620	233.414	241.098

Execution times for the bridge crossing problem (in seconds).

IN CONTEXT (I)

- Timed automata
  - restricted formalism . . .
  - . . . many properties decidable
  - state-of-the-art tools: UPPAAL, RED
- Time(d) Petri nets
  - limited tool support
- Timed process algebras

IN CONTEXT (I)

- Timed automata
  - restricted formalism ...
  - ... many properties decidable
  - state-of-the-art tools: UPPAAL, RED
- Time(d) Petri nets
  - limited tool support
- Timed process algebras
- IF, TE-LOTOS, etc:
  - separate formalisms for data types, dynamic behavior, and time
  - unclear or non-existing semantics
  - based on fixed communication primitives
- MOBY/RT
  - designs specified as PLC-automata
  - translated into timed automata for model checking
- BIP (Behavior, Interaction, Priority)
  - "Behavior is described as a Petri net extended with data and functions described in C"

IN CONTEXT (II)

Real-Time Maude:

- simple and intuitive
- expressive
- any data type
- unbounded data structures
- dynamic object/message creation/deletion
- hierarchical structures
- easy to define communication forms

IN CONTEXT (II)

Real-Time Maude:

- simple and intuitive
- expressive
- any data type
- unbounded data structures
- dynamic object/message creation/deletion
- hierarchical structures
- easy to define communication forms

- properties in general undecidable
- discrete abstraction may not exist in general

APPLICATIONS

THE MAIN QUESTION

Complex data types; unbounded data structures; flexible communication models; hierarchical objects; dynamic object creation/deletion; ...

THE MAIN QUESTION

Complex data types; unbounded data structures; flexible communication models; hierarchical objects; dynamic object creation/deletion; . . .

Are there systems where Real-Time Maude's expressiveness needed

and

Real-Time Maude analysis yields interesting results?

CLASSES OF APPLICATIONS

- “Concrete” systems/protocols
- Semantic framework for real-time systems
- Formal analysis tool for other languages
- ...

AER/NCA [WITH C. TALCOTT AND OTHERS]

AER/NCA:

- Multicast for active networks
  - 50 pages of use cases
  - involves link capacity and propagation delay, packet sizes, etc.

AER/NCA [WITH C. TALCOTT AND OTHERS]

AER/NCA :

- Multicast for active networks
  - 50 pages of use cases
  - involves link capacity and propagation delay, packet sizes, etc.
- Real-Time Maude analysis found all known design errors

AER/NCA [WITH C. TALCOTT AND OTHERS]

AER/NCA :

- Multicast for active networks
  - 50 pages of use cases
  - involves link capacity and propagation delay, packet sizes, etc.
- Real-Time Maude analysis found all known design errors
- . . . and additional unknown serious design errors

AER/NCA [WITH C. TALCOTT AND OTHERS]

AER/NCA :

- Multicast for active networks
  - 50 pages of use cases
  - involves link capacity and propagation delay, packet sizes, etc.
- Real-Time Maude analysis found all known design errors
- . . . and additional unknown serious design errors

Key Real-Time Maude features:

- detailed parametric model of communication
- laaaaarge functions
- multiple class inheritance to combine subprotocols

CASH SCHEDULING ALGORITHM [ÖLVECZKY-CACCAMO'06]

CASH: State-of-the-art scheduling algorithm

- A job can use more or less time than allocated
- Unused execution times put in a queue for reuse

CASH SCHEDULING ALGORITHM [ÖLVECZKY-CACCAMO'06]

CASH: State-of-the-art scheduling algorithm

- A job can use more or less time than allocated
- Unused execution times put in a queue for reuse
- Simulation: # elements in queue unbounded

CASH SCHEDULING ALGORITHM [ÖLVECZKY-CACCAMO'06]

CASH: State-of-the-art scheduling algorithm

- A job can use more or less time than allocated
- Unused execution times put in a queue for reuse
- Simulation: # elements in queue unbounded
- Search found missed hard deadline

CASH SCHEDULING ALGORITHM [ÖLVECZKY-CACCAMO'06]

CASH: State-of-the-art scheduling algorithm

- A job can use more or less time than allocated
- Unused execution times put in a queue for reuse
- Simulation: # elements in queue unbounded
- Search found missed hard deadline
- Extensive "Monte-Carlo simulation" did not find flaw

CASH SCHEDULING ALGORITHM [ÖLVECZKY-CACCAMO'06]

CASH: State-of-the-art scheduling algorithm

- A job can use more or less time than allocated
- Unused execution times put in a queue for reuse
- Simulation: # elements in queue unbounded
- Search found missed hard deadline
- Extensive "Monte-Carlo simulation" did not find flaw

CASH SCHEDULING ALGORITHM [ÖLVECZKY-CACCAMO'06]

CASH: State-of-the-art scheduling algorithm

- A job can use more or less time than allocated
- Unused execution times put in a queue for reuse
- Simulation: # elements in queue unbounded
- Search found missed hard deadline
- Extensive "Monte-Carlo simulation" did not find flaw

Key Real-Time Maude feature: unbounded data structures

OGDC WIRELESS SENSOR NETWORK ALGORITHM [WITH S. THORVALDSEN]

OGDC: density control algorithm for wireless sensor networks

- Simulated by developers using ns-2 with wireless extension

OGDC WIRELESS SENSOR NETWORK ALGORITHM [WITH S. THORVALDSEN]

OGDC: density control algorithm for wireless sensor networks

- Simulated by developers using ns-2 with wireless extension
- New form of communication: radio transmission
  - easy to specify in Real-Time Maude
- Real-Time Maude simulations found unknown major flaw

OGDC WIRELESS SENSOR NETWORK ALGORITHM [WITH S. THORVALDSEN]

OGDC: density control algorithm for wireless sensor networks

- Simulated by developers using ns-2 with wireless extension
- New form of communication: radio transmission
  - easy to specify in Real-Time Maude
- Real-Time Maude simulations found unknown major flaw
- Performance estimation as good as WSN simulation tool

OGDC Wireless Sensor Network Algorithm [with S. Thorvaldsen]

OGDC: density control algorithm for wireless sensor networks

- Simulated by developers using ns-2 with wireless extension
- New form of communication: radio transmission
  - easy to specify in Real-Time Maude
- Real-Time Maude simulations found unknown major flaw
- Performance estimation as good as WSN simulation tool

Key Real-Time Maude features:

- easy to define “new” model of communication
- complex data types and functions (areas, angles, distances)
- simulation

MEGASTORE AND MEGASTORE-CGC [GROV-ÖLVECZKY'14]

Megastore: Google's distributed data store

MEGASTORE AND MEGASTORE-CGC [GROV-ÖLVECZKY'14]

Megastore: Google's distributed data store

- Developed Real-Time Maude specification

MEGASTORE AND MEGASTORE-CGC [GROV-ÖLVECZKY'14]

Megastore : Google's distributed data store

- Developed Real-Time Maude specification
- Megastore:
  - consistency for transactions accessing one entity group
- Megastore-CGC:
  - consistency for transactions accessing multiple entity groups

MEGASTORE AND MEGASTORE-CGC [GROV-ÖLVECZKY'14]

Megastore: Google's distributed data store

- Developed Real-Time Maude specification
- Megastore:
  - consistency for transactions accessing one entity group
- Megastore-CGC:
  - consistency for transactions accessing multiple entity groups

Key Real-Time Maude features:

- simple and intuitive language
- automatic "testing" highly appreciated
- analysis of performance and correctness

HUMAN MULTITASKING [BROCCIA-MILAZZO-ÖLVECZKY'18]

Modeling and analysis framework for human multitasking

- human short-term memory
- attention
- tasks
- ...

HUMAN MULTITASKING [BROCCIA-MILAZZO-ÖLVECZKY'18]

Modeling and analysis framework for human multitasking

- human short-term memory
- attention
- tasks
- ...

- Will GPS distract driver for more than n seconds?
- Will other tasks make driver/pilot forget important things?

SOME OTHER "CONCRETE" APPLICATIONS

- Found several bugs in embedded car software used by major car makers (Japan)
  - bugs not found by model-checking tools employed in industry

SOME OTHER "CONCRETE" APPLICATIONS

- Found several bugs in embedded car software used by major car makers (Japan)
  - bugs not found by model-checking tools employed in industry
- ERMTS/ETCS railway signaling and control system
- Leader election for mobile ad hoc networks
- EIGRP Cisco routing protocol (Riesco, Verdejo)
- Parts of NORM multicast protocol developed by IETF

FORMAL SEMANTICS AND ANALYSIS FOR MDE LANGUAGE

- Modeling languages for embedded systems
  - intuitive domain-specific modeling
  - often lack formal semantics and analysis

FORMAL SEMANTICS AND ANALYSIS FOR MDE LANGUAGE

- Modeling languages for embedded systems
  - intuitive domain-specific modeling
  - often lack formal semantics and analysis
- Real-Time Maude semantic framework and formal analysis tool for such languages

FORMAL SEMANTICS AND ANALYSIS FOR MDE LANGUAGE

- Modeling languages for embedded systems
  - intuitive domain-specific modeling
  - often lack formal semantics and analysis
- Real-Time Maude semantic framework and formal analysis tool for such languages
  - modeling languages used in industry
    - Ptolemy II DE models
    - AADL avionics modeling standard (subset)
    - DOCOMO's $\mathcal{L}$ language

FORMAL SEMANTICS AND ANALYSIS FOR MDE LANGUAGE

- Modeling languages for embedded systems
  - intuitive domain-specific modeling
  - often lack formal semantics and analysis
- Real-Time Maude semantic framework and formal analysis tool for such languages
  - modeling languages used in industry
    - Ptolemy II DE models
    - AADL avionics modeling standard (subset)
    - DOCOMO's $\mathcal{L}$ language
  - timed model transformations
    - Real-Time MOMENT-2
    - e-Motions
  - Orc, Timed Rebeca, . . .

PTOLEMY II DE MODELS [JOINT WORK WITH KYUNGMIN BAE ET AL.]

Ptolemy II

- graphical modeling and simulation tool from UC Berkeley
- hierarchical composition of actors

PTOLEMY II DE MODELS [JOINT WORK WITH KYUNGMIN BAE ET AL.]

Ptolemy II

- graphical modeling and simulation tool from UC Berkeley
- hierarchical composition of actors
- different models of computation
- Discrete Event (DE) models:
  - timed
  - fixed-point semantics of synchronous languages

PTOLEMY II DE MODELS [JOINT WORK WITH KYUNGMIN BAE ET AL.]

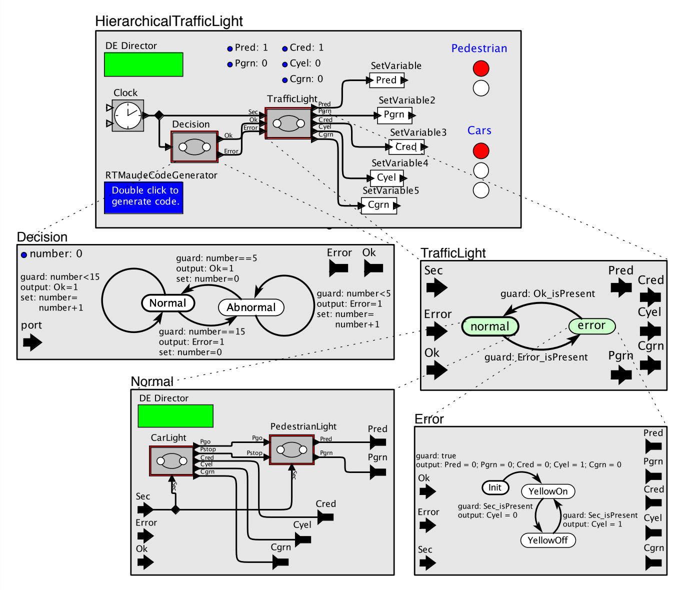
Ptolemy II

- graphical modeling and simulation tool from UC Berkeley
- hierarchical composition of actors
- different models of computation
- Discrete Event (DE) models:
  - timed
  - fixed-point semantics of synchronous languages

Key Maude features:

- hierarchical configurations
- expressiveness
- unbounded data structures
- parametric atomic propositions

PTOLEMY II: FAULT-TOLERANT TRAFFIC LIGHTS



FORMAL ANALYSIS OF PTOLEMY DE MODELS

Predefined parametric propositions:

$$\textit{actorId}\ \texttt{|}\ \textit{var}_1 = \textit{value}_1, \ldots, \textit{var}_n = \textit{value}_n$$

$$\textit{actorId}\ \texttt{@}\ \textit{location}$$

$$\textit{actorId}\ \texttt{|}\ \texttt{port}\ p\ \texttt{is}\ \textit{value}$$

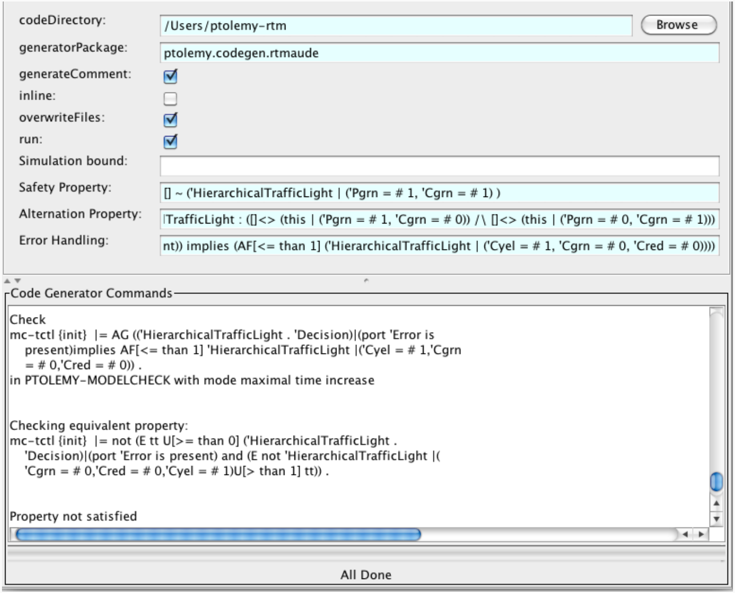
$$\textit{actorId}\ \texttt{|}\ \texttt{port}\ p\ \texttt{is}\ \textit{status}$$

A TIMED CTL PROPERTY

Car light will show only yellow within time 1 of a failure:

```
AG (('HierarchicalTrafficLight . 'Decision |
        port 'Error is present)
   => AF[<= 1] ('HierarchicalTrafficLight |
                   'Cyel = 1, 'Cgrn = 0, 'Cred = 0))
```

ANALYZING PTOLEMY II MODELS WITHIN PTOLEMY



CONCLUDING REMARKS

- Real-Time Maude formalism expressive and intuitive

CONCLUDING REMARKS

- Real-Time Maude formalism expressive and intuitive
- Sound/complete timed CTL model checking abstraction/discretization for time-robust theories
    - sound/complete analysis for new classes of systems

CONCLUDING REMARKS

- Real-Time Maude formalism expressive and intuitive
- Sound/complete timed CTL model checking
  abstraction/discretization for time-robust theories
  - sound/complete analysis for new classes of systems
- Used on state-of-the-art systems in different domains
  - value added to domain-specific analysis

CONCLUDING REMARKS

- Real-Time Maude formalism expressive and intuitive
- Sound/complete timed CTL model checking
  abstraction/discretization for time-robust theories
  - sound/complete analysis for new classes of systems
- Used on state-of-the-art systems in different domains
  - value added to domain-specific analysis
- Useful both as simulation tool and model checker

CONCLUDING REMARKS

- Real-Time Maude formalism expressive and intuitive
- Sound/complete timed CTL model checking
  abstraction/discretization for time-robust theories
  - sound/complete analysis for new classes of systems
- Used on state-of-the-art systems in different domains
  - value added to domain-specific analysis
- Useful both as simulation tool and model checker
- Semantics and analysis tool for modeling languages
  - model checker for free for those languages

IMMEDIATE RESEARCH CHALLENGES

COMBINING REAL-TIME AND PROBABILITIES

- Large distributed systems often real-time and probabilistic

COMBINING REAL-TIME AND PROBABILITIES

- Large distributed systems often real-time and probabilistic
- PVeStA and MultiVeStA: statistical model checking
    - estimate value of expression with statistical guarantees
    - scalable formal method
    - apply to fully probabilistic rewrite theories

COMBINING REAL-TIME AND PROBABILITIES (CONT.)

Exists:

1. Formal model for probabilistic real-time rewrite theories

   [Bentea-Ölveczky'11]

COMBINING REAL-TIME AND PROBABILITIES (CONT.)

Exists:

1. Formal model for probabilistic real-time rewrite theories [Bentea-Ölveczky'11]
2. Obtain fully probabilistic "real-time" OO models [Meseguer et al'12]:
   - message delay sampled probabilistically from dense interval
     - zero probability that two message-triggered actions enabled same time
   - estimate performance, etc, of cloud computing systems and DoS defense mechanisms

COMBINING REAL-TIME AND PROBABILITIES (CONT.)

Exists:

1. Formal model for probabilistic real-time rewrite theories [Bentea-Ölveczky'11]
2. Obtain fully probabilistic "real-time" OO models [Meseguer et al'12]:
   - message delay sampled probabilistically from dense interval
     - zero probability that two message-triggered actions enabled same time
   - estimate performance, etc, of cloud computing systems and DoS defense mechanisms

Need: theoretical model and language/tool support

- larger class of systems (message-triggered; flat OO; ad-hoc timing)
- automatic transformation to PVeStA

SYMBOLIC METHODS

- Dense time $\longrightarrow$ symbolic methods important

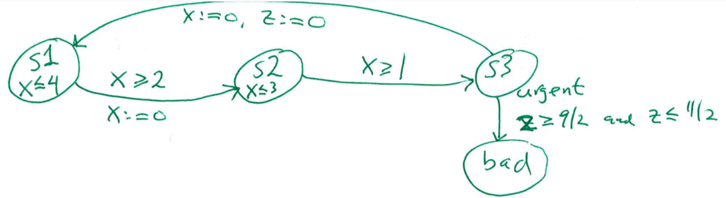
SYMBOLIC METHODS

- Dense time $\longrightarrow$ symbolic methods important
- Soundness/completeness for non-time-robust theories
  - events not at “fixed” times

SYMBOLIC METHODS

- Dense time $\longrightarrow$ symbolic methods important
- Soundness/completeness for non-time-robust theories
  - events not at "fixed" times
- "Real-Time Maude modulo SMT"
  - full reachability analysis for timed automata (?)

REWRITING MODULO SMT: COMPLETE REACHABILITY FOR TIMED AUTOMATA (?)




```
vars x y z : Real .

crl < s1 ; x ; z > => < s2 ; 0/1 ; z >  if x >= 2/1 = true .
crl < s1 ; x ; z > => < s1 ; x + y ; z + y >  if x + y <= 4/1 =
crl < s2 ; x ; z > => < s3 ; x ; z >  if x >= 1/1 = true .
crl < s2 ; x ; z > => < s2 ; x + y ; z + y > if x + y <= 3/1 = t
crl < s3 ; x ; z > => < bad ; x ; z >  if z >= 9/2 and z <= 11/2
rl < s3 ; x ; z > => < s1 ; 0/1 ; 0/1 > .
```

REWRITING MODULO SMT: COMPLETE REACHABILITY FOR TIMED AUTOMATA (?) (CONT.)

Region reachable from some initial *x*-value?

```
Maude> smt-search [1]  < s1 ; x ; 0/1 > =>*  < bad ; y ; z >
         such that z > y and z > 2/1 = true .
```

```
Solution 1
rewrites: 26 in 21ms cpu (21ms real) (1233 rewrites/second)
state: < bad ; 0/1 + #2-y:Real ; 0/1 + #1-y:Real + #2-y:Real >
empty substitution
where z > y and z > 2/1 and x + #1-y:Real <= 4/1 and x + #1-y:Real >= 2/1 and 0
    #2-y:Real >= 1/1 and (0/1 + #1-y:Real + #2-y:Real >= 9/2 and 0/1 + #1-y:Rea
    0/1 + #2-y:Real and z === 0/1 + #1-y:Real + #2-y:Real)
```

REFERENCE PAPERS ON REAL-TIME MAUDE

- Peter C. Ölveczky: Real-Time Maude and its Applications. In Proc. WRLA 2014, volume 8663 of Lecture Notes in Computer Science, Springer, 2014.
- Peter C. Ölveczky and José Meseguer: Semantics and Pragmatics of Real-Time Maude. In volume 20(1/2) of Higher-Order and Symbolic Computation, Springer 2007.

EXERCISE

In rate-monotonic scheduling, you have given a set of periodic tasks, each with an execution time and period. When its period ends, it starts a new period. Within each period, it must execute for a total of its execution time. We assume that we have only one processor. The task with the shortest period has the highest priority to execute, and can preempt an executing task with a lpwer priority. Assume that you can have tasks with the same period/priority. Specify the rate-monotnic scheduling algorithm in Real-Time Maude, and try model checking of a few examples to check whetehr your task set is schedulable.