Modeling and Analysis in Maude

Lecture 4: Analyzing models

Einar Broch Johnsen

University of Oslo, Norway
einarj@ifi.uio.no

DAT355, 14 April 2021

UNIVERSITY OF OSLO

SIRIUS Center for Scalable Data Access in the Oil and Gas Domain

sfi Centre for Research-based Innovation
The Research Council of Norway

http://www.sirius-labs.no

Plan for the lectures

- **Lecture 1:** Basic ideas and concepts of rewriting logic & Maude
- **Lecture 2:** From equations to rules
- **Lecture 3:** Modeling parallel & distributed systems
- **Lecture 4:** Analyzing models

Analyzing models

Summary of yesterday's lecture

Modeling parallel & distributed systems

- GCL: Modeling of interleaved concurrency
- Multisets as a representation of system configurations
- Predefined module CONFIGURATION
- Define constructors for the sorts Object and Msg
- Modelling patterns:
  - internal activity,
  - synchronous communication,
  - asynchronous communication

Questions for today

1. How many final states from init3 in GCL?
2. How can we model multicast and broadcast?

Multicast / broadcast

- **Multicast**: send a message to a list of receivers
- **Broadcast**: send a message to all receivers
- **Aim**: do this in one rewrite step to avoid possible interference from other rules ...

op greeks : $\rightarrow$ Configuration .
eq greeks =< "Gaia" : Person | age : 999, status : single >
< "Uranus" : Person | age : 900, status : single >
< "Kronos" : Person | age : 800, status : single >
< "Rhea" : Person | age : 16, status : single > .

Messages

Messages are defined as terms of sort `Msg`:

> **ops** marry? yes no : Oid Oid $\rightarrow$ Msg .
> **op** waitFor : Oid $\rightarrow$ Status .

Example: Send a `marry?` message:

crl [propose] :
$<$ X : Person | age : N, status : single $>$
$<$ Y : Person | age : M, status : single $>$
$\Longrightarrow$
$<$ X : Person | age : N, status : waitFor(Y) $>$
$<$ Y : Person | age : M, status : single $>$
marry?(Y, X)
if N $>$ 15 .

Multicast

Let us assume an additional attribute

op prospects : _ : List{Oid} $\rightarrow$ Attribute [**ctor**] .

such that

op greeks : $\rightarrow$ Configuration .
eq greeks =
$<$ "Gaia" : Person | age : 999, status : single, prospects : ("Uranus" "Kronos" "Rhea") $>$
$<$ "Uranus" : Person | age : 900, status : single $>$
$<$ "Kronos" : Person | age : 800, status : single $>$
$<$ "Rhea" : Person | age : 12, status : single $>$.

op multimarry : List{Oid} Oid $\rightarrow$ Msg .

rl [multicast] : $<$ X : Person | age : N, status : single, prospects : L $>$
$\Longrightarrow$ $<$ X : Person | age : N, status : waitFor(L), prospects : L $>$ multimarry(L,X) .

eq multimarry(nil, Y) =none .
ceq multimarry(X L, Y) =marry?(X,Y) multimarry(L,Y) **if** X =/=Y .
eq multimarry(X L, X) =multimarry(L,X) .

Broadcast

Idea: We want to collect the receivers from the Configuration.

```
var C : Configuration .

op collect : Configuration → List{Oid} .
eq collect(none) = nil .
eq collect ( < X : Person | AS > C) = X collect(C) .
eq collect (message C) = collect(C) .
```

Trick:

```
op { _ } : Configuration → Configuration [ ctor ] .

crl [broadcast] :
  { < X : Person | age : N, status : single, AS > C } ⟹
  { < X : Person | age : N, status : waitFor(L), AS > C multimarry(L,X) }
  if L := collect(C) .
```

Use the brackets to enclose the Configuration:

```
op greeks2 : → Configuration .
eq greeks2 = { greeks } .
```

Analysis in Maude (1)

- Maude specs. are executable (under reasonable conditions)
- Supports a range of increasingly stronger validation techniques:

1. Rewriting / prototyping / simulation:
   - Simulate one possible behavior from one initial state
2. Exhaustive search:
   - Search all possible behaviors from one initial state
   - Search for reachable "good" or "bad" states, deadlocks
   - Breadth-first search, may search forever (or exhaust memory)
3. Linear temporal logic (LTL) model checking:
   - Check whether each behavior from one initial state satisfies a temporal property
   - "Will one reach a desired state in all behaviors?"
   - "Will a request eventually be followed by a response?"
   - Set of reachable states from initial state must be finite

GCL Semantics

```
mod GCL−SEMANTICS is pr EVAL . pr RUNTIME−SYNTAX .

  var RS : RuntimeState . var x : Var .
  vars e e' : Expr . vars Sigma Sigma' : Substitution .
  vars s1 s2 : Statement . vars g g' : GuardStm .

  rl [Skip] : skip ; g ⟹ g .

  crl [Assignment] : Sigma (e ▷ x :=e' ; g) ⟹ Sigma' g
    if eval(Sigma, e) /\ Sigma' :=insert(x, eval(Sigma, e'), Sigma) .

  crl [Spawn] : Sigma (e ▷ spawn(g') ; g) ⟹ Sigma g g' if eval(Sigma, e) .

  crl [Choice1] : Sigma (s1 ◁ e ▷ s2 ; g) ⟹ RS
    if eval(Sigma, e) /\ (Sigma (e ▷ s1 ; g)) ⟹ RS .

  crl [Choice2] : Sigma (s1 ◁ e ▷ s2 ; g) ⟹ RS
    if eval(Sigma, not(e)) /\ (Sigma ((not e) ▷ s2 ; g)) ⟹ RS .
endm
```

Some initial states

ops init1 init2 init3 : $\to$ RuntimeState .

eq init1 =('x $\mapsto$1, 'y $\mapsto$2) (true $\rhd$ spawn(true $\rhd$ 'x :='x +1) ; true $\rhd$ spawn(true $\rhd$ 'y :=3)) .

eq init2 =('x $\mapsto$1) (true $\rhd$ spawn(true $\rhd$ 'x :='x +1) ; true $\rhd$ spawn(true $\rhd$ 'x :=3)) .

eq init3 =('x $\mapsto$0, 'y $\mapsto$0, 'flag $\mapsto$true)
(true $\rhd$ spawn(('x =:=0 or 'flag) $\rhd$ 'x :=100 $*$ 'x) ;
true $\rhd$ spawn(('x :='x +2 $\lhd$ 'flag $\rhd$ 'x :='x +−1 ; true $\rhd$ 'flag :=(not 'flag) ;
'x :='x +2 $\lhd$ 'flag $\rhd$ 'x :='x +−1 ; true $\rhd$ 'flag :=(not 'flag) ;
'x :='x +2 $\lhd$ 'flag $\rhd$ 'x :='x +−1 ; true $\rhd$ 'flag :=(not 'flag) ;
'x :='x +2 $\lhd$ 'flag $\rhd$ 'x :='x +−1 ; true $\rhd$ 'flag :=(not 'flag) ;
'x :='x +2 $\lhd$ 'flag $\rhd$ 'x :='x +−1 ; true $\rhd$ 'flag :=(not 'flag) ;
true $\rhd$ 'y :='y +1))) .

Question for today: How many final states do we reach from init3?

From (Fair) Rewriting to Search

What happens when we run

rew init3 .

or

frew init3 .

Is the Maude model of GCL deterministic?

Let's search for all possible final states...

search init3 $\Longrightarrow$! RS .

What happened here?

Search

- Maude's search-command visits states which are reachable from the initial state by means of rewrites.
- This is a *breadth-first* search. It will find the state we are searching for *if* this state exists, otherwise the search may not terminate...
- Different notions of reachability can be explored

search init3 $\Longrightarrow$ 1 RS .
search init3 $\Longrightarrow$! RS .
search init3 $\Longrightarrow$ + RS .
search init3 $\Longrightarrow$ $*$ RS .

search [n] init3 $\Longrightarrow$! RS .

search init3 $\Longrightarrow$! RS **such that** ...

For example:

search init3 $\Longrightarrow$! Sigma RS **such that** Sigma['x] $>$ 300 $*$ Sigma['y] .

Maude can show you the sequence of transitions to reach a particular solution:

show path 2 .

Did Rhea get married?

var S : Status . **var** AS : AttributeSet . **var** C : Configuration .

crl [birthday] : $<$ X : Person | age : N , status : S $> \Longrightarrow$
$<$ X : Person | age : (N +1), status : S $>$ **if** N $<$ 999 .

rl [engagement] : $<$ X : Person | age : N, status : single $>$
$<$ X' : Person | age : N', status : single $>$
$\Longrightarrow$ $<$ X : Person | age : N, status : engaged(X') $>$
$<$ X' : Person | age : N', status : engaged(X) $>$.

crl [wedding] : $<$ X : Person | status : engaged(X'), age : N $>$
$<$ X' : Person | status : engaged(X), age : N' $>$
$\Longrightarrow$ $<$ X : Person | status : married(X'), age : N $>$
$<$ X' : Person | status : married(X), age : N' $>$ **if** N $>$ 15 and N' $>$ 15 .

eq greeks $=<$ "Gaia" : Person | age : 999, status : single $>$
$<$ "Uranus" : Person | age : 900, status : single $>$
$<$ "Kronos" : Person | age : 800, status : single $>$
$<$ "Rhea" : Person | age : 12, status : single $>$.

Questions

- How large is the state space of this model?
- Examples of properties that we can search for

$\textbf{search}$ greeks $\Longrightarrow$ 1 C .

$\textbf{search}$ [1] greeks $\Longrightarrow$! C .

$\textbf{search}$ greeks $\Longrightarrow$! C .

$\textbf{search}$ greeks $\Longrightarrow * <$ "Rhea" : Person | age : N, status : married(X) $>$ C .

$\textbf{search}$ greeks $\Longrightarrow$! $<$ "Rhea" : Person | age : N, status : married(X) $>$ C .

$\textbf{search}$ [1] greeks $\Longrightarrow * <$ "Rhea" : Person | age : N, status : married(X) $>$ C .

We can also look for errors:

- Can marriage be non-mutual?
- Can Rhea not get married?

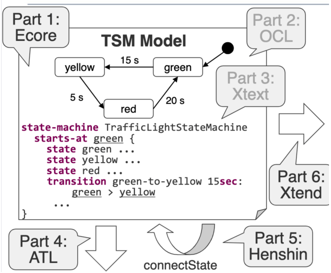
Model checking Maude models

What is model-checking?

Model checking Maude models

A Model of Traffic Lights as Timed State Machines

- Traffic lights have three states: Red, Yellow, Green
- Red light remains for 20 seconds
- Green light remains for 15 seconds
- Yellow light remains for 5 seconds



Let's make a model of the traffic lights using Maude's object model.

Traffic light model (1)

Idea: We use a tick-rule to evolve time, and add a timer to delay transitions between colors in the object state

mod TRAFFICLIGHT **is**
  pr CONFIGURATION . **pr** STRING . **pr** NAT .

  sort Color .
  ops Red Yellow Green : → Color [**ctor**] .
  op state : _ : Color → Attribute [**ctor**] .
  op timer : _ : Nat → Attribute [**ctor**] .
  op TrafficLight : → Cid [**ctor**] .

  subsort String < Oid .
endm

An object has the following form:

  < "tl" : TrafficLight | state : Red, timer : 2 >

Now we need rules: green2yellow, yellow2red, red2green and tick.

Traffic light model (2)

var X : Oid . **var** N : Nat . **var** S : Color . **var** C : Configuration .

rl [green2yellow] : $<$ X : TrafficLight | state : Green, timer : 0 $>$
$\Longrightarrow$ $<$ X : TrafficLight | state : Yellow, timer : 5 $>$.

rl [yellow2red] : $<$ X : TrafficLight | state : Yellow, timer : 0 $>$
$\Longrightarrow$ $<$ X : TrafficLight | state : Red, timer : 20 $>$.

rl [red2green] : $<$ X : TrafficLight | state : Red, timer : 0 $>$
$\Longrightarrow$ $<$ X : TrafficLight | state : Green, timer : 15 $>$.

crl [tick] : $<$ X : TrafficLight | state : S, timer : N $>$
$\Longrightarrow$ $<$ X : TrafficLight | state : S, timer : sd(N,1) $>$ **if** N $>$ 0 .

Traffic light model (2)

Some initial states:

$$
\begin{array}{l}
\textbf{ops}\ \text{initial1 initial2} : \to \text{Configuration} .\\
\textbf{eq}\ \text{initial1} = < \text{"tl"} : \text{TrafficLight} \mid \text{state} : \text{Red}, \text{timer} : 2 > .\\
\textbf{eq}\ \text{initial2} = < \text{"tl1"} : \text{TrafficLight} \mid \text{state} : \text{Red}, \text{timer} : 2 >\\
\qquad\qquad\quad < \text{"tl2"} : \text{TrafficLight} \mid \text{state} : \text{Red}, \text{timer} : 2 > .
\end{array}
$$

How does this model behave? Is the model terminating?

Let's do some rews, frews and searches...

Traffic light model (2)

Is is always the case that when the light is Red, it will eventually become Yellow?

This is a *temporal* property of the model, perfect for LTL model-checking...

How can we specify this property in LTL?

Assume we have a state property *red* which is true when the traffic light is red, and *yellow* which is true when the traffic light is yellow.

Then the LTL property becomes:

$$\Box\ (red \rightarrow \Diamond\ yellow)$$

Maude's transition system can be explored to model-check LTL properties.

Traffic light model (2)

Load model−checker.maude. Module SATISFACTION defines a sort Prop and

```
op _|=_ : State Prop → Bool [frozen] .
```

We use this to define:

```
mod TRAFFICLIGHT−PREDS is pr TRAFFICLIGHT . pr SATISFACTION .
  subsort Configuration < State .
  ops red green yellow : Oid → Prop .

  vars X Y : Oid . var S : Color . vars N M : Nat . var C : Configuration . var P : Prop .

  eq < X : TrafficLight | state : Red, timer : N > C |=red(X) =true .
  eq < X : TrafficLight | state : Yellow, timer : N > C |=yellow(X) =true .
  eq < X : TrafficLight | state : Green, timer : N > C |=green(X) =true .

  op same−color : Oid Oid → Prop .
  eq < X : TrafficLight | state : S, timer : N >
     < Y : TrafficLight | state : S, timer : M > C |=same−color(X,Y) =true .

  eq (C |=P) =false [owise] .
endm
```

Traffic light model (3)

We can now combine TRAFFICLIGHT-PREDS with Maude's MODEL-CHECKER module:

```
mod TRAFFICLIGHT−CHECK is
    pr TRAFFICLIGHT−PREDS . pr MODEL−CHECKER . pr LTL−SIMPLIFIER .
endm
```

Let us try some LTL formulas:

```
reduce modelCheck(initial1, red("tl")) .

reduce modelCheck(initial1, green("tl")) .

reduce modelCheck(initial1, <> ~ yellow("tl")) .

reduce in TRAFFICLIGHT−CHECK :
 modelCheck(initial1, [] (red("tl") -> <> yellow("tl")) ) .

reduce in TRAFFICLIGHT−CHECK :
modelCheck(initial1, [] (red("tl") -> (red("tl") U green("tl")) )) .
```

Analysis techniques in Maude

We have seen the following analysis techniques for Maude models:

- **rew**: explore a single transition path
- **frew**: explore a transition path with a notion of fairness
  - Generally, **frew** is rule fair but not position fair
  - Mostly position fair using objects and messages)
- **search** in a number of variations: =>*, =>!, such that, ...
  - Search checks reachability of a configuration matching a pattern
  - You can check invariants by searching for a counter-example...
- **LTL model-checking** over rewrite theories
  - Interprets the rewrite system as a Kripke sturcture
  - State predicates apply to Kripke frames
  - Rewrite rules transition between the frames