Reconfigurable stochastic multi-formalism models: an approach based on Maude

Lorenzo Capra[1] Marco Gribaudo[2]

Università degli Studi di Milano, Italy lorenzo.capra@unimi.it

Politecnico di Milano, marco.gribaudo@polimi.it

The 2nd International Workshop on Reconfigurable Transition Systems: Semantics, Logics and Applications, November 11, 2025, Toledo

Motivations

Context: performance modeling of complex adaptive systems

- As systems become more complex, single-formalism models often become too complex or too simplistic.
- Multiformalism modeling is a methodology for assessing system (quantitative) properties by integrating diverse modeling techniques.
- It allows choosing the optimal formalism for each component, often combined with multi-solution approaches.
  - Some available toolsets: AToM, Möbius, OsMoSys, **SIMTHESys** (facilitates the definition of new formalisms and multiformalism solvers)
  - Practical usability issues, no dynamic features

Motivations

Context: performance modeling of complex adaptive systems

- As systems become more complex, single-formalism models often become too complex or too simplistic.
- Multiformalism modeling is a methodology for assessing system (quantitative) properties by integrating diverse modeling techniques.
- It allows choosing the optimal formalism for each component, often combined with multi-solution approaches.
  - Some available toolsets: AToM, Möbius, OsMoSys, **SIMTHESys** (facilitates the definition of new formalisms and multiformalism solvers)
  - Practical usability issues, no dynamic features

Maude (https://maude.cs.illinois.edu/)

Declarative, expressive, performing, and with *rewriting logic* semantics

- Pattern-matching (rewriting) modulo-axioms, reflection, subtyping ...
- Logical framework for various formalisms (PN, CCS, CSP, BPM, ..).
- Inbuilt *model-checking* facilities
- Recently supported in *performance analysis*

Fundamental concept: utilizing `Maude` as a multiformalism Framework

Maude (`https://maude.cs.illinois.edu/`)

Declarative, expressive, performing, and with *rewriting logic* semantics

- Pattern-matching (rewriting) modulo-axioms, reflection, subtyping ...
- Logical framework for various formalisms (PN, CCS, CSP, BPM, ..).
- Inbuilt *model-checking* facilities
- Recently supported in *performance analysis*

Fundamental concept: utilizing `Maude` as a multiformalism Framework

Maude (https://maude.cs.illinois.edu/)

Declarative, expressive, performing, and with *rewriting logic* semantics

- Pattern-matching (rewriting) modulo-axioms, reflection, subtyping ...
- Logical framework for various formalisms (PN, CCS, CSP, BPM, ..).
- Inbuilt *model-checking* facilities
- Recently supported in *performance analysis*

Fundamental concept: utilizing `Maude` as a multiformalism Framework

Maude (`https://maude.cs.illinois.edu/`)

Declarative, expressive, performing, and with *rewriting logic* semantics

- Pattern-matching (rewriting) modulo-axioms, reflection, subtyping ...
- Logical framework for various formalisms (PN, CCS, CSP, BPM, ..).
- Inbuilt *model-checking* facilities
- Recently supported in *performance analysis*

Fundamental concept: utilizing `Maude` as a multiformalism Framework

Maude

Intuitive rewriting semantics.

Statements: *equations* (simplifications) + *rewrite rules* (local concurrent state transitions).

- *Functional* module := *equational theory* $(\Sigma, E \cup A)$[1]. Model: *initial* algebra $T_{\Sigma/E\cup A} \cong$ algebra of *canonical* terms (assuming the principles of Church-Rosser and termination).
- *System* module := *rewrite theory* $(\Sigma, E \cup A, R)$. Model: *labeled transition system* (TS) associated with each term:
  - states: *canonical* terms;
  - state transitions: *equivalence classes* of rewrites

[1] Σ: signature (sorts, subsorts, ops), E: axioms, A: operator equational attributes

Maude

Intuitive rewriting semantics.

Statements: *equations* (simplifications) + *rewrite rules* (local concurrent state transitions).

- *Functional* module := *equational theory* $(\Sigma, E \cup A)^1$. Model: *initial* algebra $T_{\Sigma/E\cup A} \cong$ algebra of *canonical* terms (assuming the principles of Church-Rosser and termination).
- *System* module := *rewrite theory* $(\Sigma, E \cup A, R)$. Model: *labeled transition system* (TS) associated with each term:
    - states: *canonical* terms;
    - state transitions: *equivalence classes* of rewrites

[1] Σ: signature (sorts, subsorts, ops), E: axioms, A: operator equational attributes

Maude

Intuitive rewriting semantics.

Statements: *equations* (simplifications) + *rewrite rules* (local concurrent state transitions).

- *Functional* module := *equational theory* $(\Sigma, E \cup A)$[1]. Model: *initial* algebra $T_{\Sigma/E\cup A} \cong$ algebra of *canonical* terms (assuming the principles of Church-Rosser and termination).
- *System* module := *rewrite theory* $(\Sigma, E \cup A, R)$. Model: *labeled transition system* (TS) associated with each term:
  - states: *canonical* terms;
  - state transitions: *equivalence classes* of rewrites

[1] Σ: signature (sorts, subsorts, ops), E: axioms, A: operator equational attributes

Maude

Intuitive rewriting semantics.

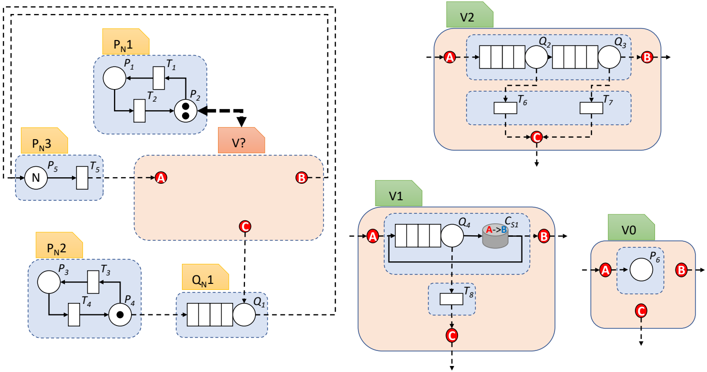
Statements: *equations* (simplifications) + *rewrite rules* (local concurrent state transitions).

- *Functional* module := *equational theory* $(\Sigma, E \cup A)^1$. Model: *initial* algebra $T_{\Sigma/E\cup A} \cong$ algebra of *canonical* terms (assuming the principles of Church-Rosser and termination).
- *System* module := *rewrite theory* $(\Sigma, E \cup A, R)$. Model: *labeled transition system* (TS) associated with each term:
  - states: *canonical* terms;
  - state transitions: *equivalence classes* of rewrites

[1] Σ: signature (sorts, subsorts, ops), E: axioms, A: operator equational attributes

Case-study: Reconfigurable server

A server has two computing units with buffers that stop new requests when full. A front-end routes requests; the second unit manages time-outs from the first. Ideally, requests pass through two phases, but overflow goes to a backup server during high demand. Maintenance may make all nodes unavailable, but if one fails, the other manages both stages.



Encoding an heterogeneous Network in Maude

- A multiformalism model integrates components like Petri nets, multi-class queue networks, and JSQ to maintain a distributed state, formalized by commutative monoids
- The `NETWORK{L :: C-MONOID}` module provides an ADT for the model.
- The model's abstract signature includes the sort `Network`, subsort `Node`, and AC `Network` juxtaposition `_,_`.
- The model functions as a multiset of diverse nodes, interacting via shared state elements.
- Specific types of nodes link to the network via subsort relationships (e.g., `Queue < Node`).

Encoding an heterogeneous Network in Maude

- A multiformalism model integrates components like Petri nets, multi-class queue networks, and JSQ to maintain a distributed state, formalized by commutative monoids
- The `NETWORK{L :: C-MONOID}` module provides an ADT for the model.
- The model's abstract signature includes the sort `Network`, subsort `Node`, and AC `Network` juxtaposition `_,_`.
- The model functions as a multiset of diverse nodes, interacting via shared state elements.
- Specific types of nodes link to the network via subsort relationships (e.g., `Queue < Node`).

Encoding an heterogeneous Network in Maude

- A multiformalism model integrates components like Petri nets, multi-class queue networks, and JSQ to maintain a distributed state, formalized by commutative monoids
- The `NETWORK{L :: C-MONOID}` module provides an ADT for the model.
- The model's abstract signature includes the sort `Network`, subsort `Node`, and AC `Network` juxtaposition `_,_`.
- The model functions as a multiset of diverse nodes, interacting via shared state elements.
- Specific types of nodes link to the network via subsort relationships (e.g., `Queue < Node`).

Encoding an heterogeneous Network in Maude

- A multiformalism model integrates components like Petri nets, multi-class queue networks, and JSQ to maintain a distributed state, formalized by commutative monoids
- The `NETWORK{L :: C-MONOID}` module provides an ADT for the model.
- The model's abstract signature includes the sort `Network`, subsort `Node`, and AC `Network` juxtaposition `_,_`.
- The model functions as a multiset of diverse nodes, interacting via shared state elements.
- Specific types of nodes link to the network via subsort relationships (e.g., `Queue < Node`).

Encoding an heterogeneous Network in Maude

- A multiformalism model integrates components like Petri nets, multi-class queue networks, and JSQ to maintain a distributed state, formalized by commutative monoids
- The `NETWORK{L :: C-MONOID}` module provides an ADT for the model.
- The model's abstract signature includes the sort `Network`, subsort `Node`, and AC `Network` juxtaposition `_,_`.
- The model functions as a multiset of diverse nodes, interacting via shared state elements.
- Specific types of nodes link to the network via subsort relationships (e.g., `Queue < Node`).

Heterogeneous Network ADT

https://github.com/lgcapra/rewpt/tree/main/multiformalism

```
fth C-MONOID is *** distributed state concept
   sort Elt .
   op 0 : -> Elt .
   op _+_ : Elt Elt -> Elt [assoc comm id: 0] .
endfth

fmod NETWORK{S :: C-MONOID} is *** network parametric in the state type
 protecting EXT-BOOL .
 sorts Node Network NetSys .
 subsort Node < Network .
 op emptyNetW : -> Network [ctor] .
 op _,_ : Network Network -> Network [ctor assoc comm prec 123 id: emptyNetW] . *** network
 op _:_ : Network S$Elt -> NetSys [ctor prec 125] . *** network and associated state
 op netw : NetSys -> Network .
 op state : NetSys -> S$Elt .
 vars N N' : Network . var M : S$Elt .
 eq netw((N : M)) = N .
 eq state((N : M)) = M .
 op remove : Network Network -> Network . *** some network operations
 eq remove((N, N'), N) = N' .
 eq remove(N, N') = N [owise] .
 op in : Network Network -> Bool .
 eq in((N, N'), N) = true .
 eq in(N, N') = false [owise] .
endfm

view S-Pbag{PL :: TRIV} from C-MONOID to PBAG{PL} is *** maps the state concept to a multiset of places (or marking)
 sort Elt to Pbag .
 op 0 to nilP .
endv
```

Linking nodes to the network (example: Stochastic PNs)

We reuse the SPN signature (SPN-SIG)

We define the node's semantics through a rewrite rule

Standard approach; we can connect any kind of node

```
fmod SPN-NODE{PL :: TRIV} is
  extending NETWORK{S-Pbag{PL}} . *** network using a marking as distributed state
  protecting SPN-SIG{String, PL} .
  subsort Net < Node . *** link
endfm

mod SPN-NODE-SYS{PL :: TRIV} is
  including SPN-NODE{PL} .
  var N : Network . vars B B' : Pbag . var K : NzNat . var rate : Float . var T : Tran .
  crl [spn-t] : (N , T) : B => (N , T) : B' if enabled(T, B) /\ B' := firing(T, B) /\
       rate := firingRate(T, B) . *** SPN dynamics
endm
```

Case Study: includes SPN and MQN

```
mod MQN—SPN is
  including SPN—NODE—SYS{Nat} . including QUEUE—NODE—SYS{Nat} .
  var K : NzNat . *** model parameter
  vars N N' N'' : Network . var S : Pbag .
  ops eq1 eq2 eq3 : —> Server .
  eq eq1 = p(7) @ 1.0 . eq eq2 = p(1) @ 1.5 . eq eq3 = p(6) @ 2.5 .
  ops q1 q2 q3 q23 : —> Queue [memo] .
  eq q1 = [1 . p(5), nilP] eq1 > p(0) . *** elementary queues with enabling conditions
  eq q2 = [2 . p(2), nilP] eq2 > p(6) .
  eq q3 = [2 . p(2), nilP] eq3 > p(0) .
  eq q23 = [1 . p(2), nilP] ql(q2) ql(q3) > out(q3) . *** multi—class queue
  ops t0 t1 t2 t3 t4 t5 t6 : —> Tran .
  eq t0 = t("start", 1.0, 1 ) |—> [1 . p(0), 1 . p(1), nilP] . *** SPN transitions
  eq t1 = t("switch1", 0.5, 1 ) |—> [1 . p(3), 1 . p(2), nilP] .
  eq t2 = t("switch2", 0.05, 1 ) |—> [1 . p(2), 1 . p(3), nilP] .
  eq t3 = t("on", 2.0, 1 ) |—> [1 . p(4), 1 . p(5), nilP] .
  eq t4 = t("off", 1.0, 1 ) |—> [1 . p(5), 1 . p(4), nilP] .
  eq t5 = t("rem1", 1.0, 1 ) |—> [1 . p(1), 1 . p(7), nilP] .
  eq t6 = t("rem6", 1.5, 1 ) |—> [1 . p(6), 1 . p(7), nilP] .
  op network : —> Network . op netsys : NzNat —> NetSys .
  op V : NzNat —> [Network] [memo] . *** variable component (depends on the marking of place p2)
  eq V(2) = q2 , q3 , t5 , t6 . *** "out" place of q2 is "in" place for q3: sequential composition
  eq V(1) = q23 , t6 . *** hybrid component: multi—class queue + SPN transition
  eq V(0) = p(eq2) @ 0.0 p(eq3) @ 0.0 > out(q3) . *** "dead" queue
  eq network = t0 , t1 , t2 , t3 , t4 , t5 , t6 , q1 , V(2) .
  eq netsys(K) = network : K . p(0) + 2 . p(2) + 1 . p(5) .
  *** structural rewriting: the variable component is replaced under certain conditions
  crl [V2>V1] : N : S => N' , V(1) : S if S[p(2)] = 1 /\ N'' , N' := N /\ N'' = V(2) .
  crl [V2>V0] : N : S => N' , V(0) : S if S[p(2)] = 0 /\ N'' , N' := N /\ N'' = V(2) .
  crl [V1>V2] : N : S => N' , V(2) : S if S[p(2)] = 2 /\ N'' , N' := N /\ N'' = V(1) .
  crl [V1>V0] : N : S => N' , V(0) : S if S[p(2)] = 0 /\ N'' , N' := N /\ N'' = V(1) .
  crl [V0>V2] : N : S => N' , V(2) : S if S[p(2)] = 2 /\ N'' , N' := N /\ N'' = V(0) .
endm
```

The congruence issue

The pattern-matching(-mod A)-based rewriting engine of `Maude` presupposes ground **coherence**. ($\widehat{t}$: canonical form of t)

$$\text{if } t \rightarrow t' \text{ with } R \text{ modulo } A \text{ then } \widehat{t} \rightarrow t'', \text{ such that } \widehat{t'} = \widehat{t''}$$

Meticulous attention is necessary. For instance, this rudimentary formulation of the rule $[V2 > V1]$ does not work.

```
crl [V2>V1] : N , V(2) : S => N , V(1) : S if S[p(2)] = 1 .
```

We have defined a broad (structural) sufficient condition

Experimental evidence: Transition System build time

We get the generator matrix of the CTMC associated with the TS through (semi-)automated preprocessing of `Maude` modules

Solution 2: facilitates the description of node dynamics

(the data in the paper refer to a previous implementation)

N	# states	Solution 1 (sec)	Solution 2 (sec)	Thr (jobs/sec)
10	5.148	2	2	2,3
20	31.878	20	17	4,1
30	98.208	43	40	6,1
40	222.138	110	101	10,7
50	421.668	290	224	14,4
60	714.798	513	488	19,8
70	1.119.528	998	870	24,8
80	1.653.858	1.480	1.362	29,7
90	2.335.778	2.629	2.324	35,5
100	3.183.318	3.290	3.105	39,2

Conclusion and ongoing work

- objective: utilize `Maude` as an ecosystem where diverse formal models interact through a specific protocol (a distributed shared state), operating under semantics grounded in rewriting.
- parametric notion of state
- nodes might also have a hidden or private state
- inherently modular and based on a small extensible hierarchy of modules
- issues: coherence must be ensured, analysis complexity
- ongoing work: integration into graphical and multi-solution tools (**DrawNet**, **SIMTHESys**); automatic detection of symmetries to get a quotient TS and a lumped CTMC (using compositional operators)

Conclusion and ongoing work

- objective: utilize `Maude` as an ecosystem where diverse formal models interact through a specific protocol (a distributed shared state), operating under semantics grounded in rewriting.
- parametric notion of state
- nodes might also have a hidden or private state
- inherently modular and based on a small extensible hierarchy of modules
- issues: coherence must be ensured, analysis complexity
- ongoing work: integration into graphical and multi-solution tools (**DrawNet**, **SIMTHESys**); automatic detection of symmetries to get a quotient TS and a lumped CTMC (using compositional operators)

Conclusion and ongoing work

- objective: utilize `Maude` as an ecosystem where diverse formal models interact through a specific protocol (a distributed shared state), operating under semantics grounded in rewriting.
- parametric notion of state
- nodes might also have a hidden or private state
- inherently modular and based on a small extensible hierarchy of modules
- issues: coherence must be ensured, analysis complexity
- ongoing work: integration into graphical and multi-solution tools (**DrawNet**, **SIMTHESys**); automatic detection of symmetries to get a quotient TS and a lumped CTMC (using compositional operators)

Conclusion and ongoing work

- objective: utilize `Maude` as an ecosystem where diverse formal models interact through a specific protocol (a distributed shared state), operating under semantics grounded in rewriting.
- parametric notion of state
- nodes might also have a hidden or private state
- inherently modular and based on a small extensible hierarchy of modules
- issues: coherence must be ensured, analysis complexity
- ongoing work: integration into graphical and multi-solution tools (**DrawNet**, **SIMTHESys**); automatic detection of symmetries to get a quotient TS and a lumped CTMC (using compositional operators)

Conclusion and ongoing work

- objective: utilize `Maude` as an ecosystem where diverse formal models interact through a specific protocol (a distributed shared state), operating under semantics grounded in rewriting.
- parametric notion of state
- nodes might also have a hidden or private state
- inherently modular and based on a small extensible hierarchy of modules
- issues: coherence must be ensured, analysis complexity
- ongoing work: integration into graphical and multi-solution tools (**DrawNet**, **SIMTHESys**); automatic detection of symmetries to get a quotient TS and a lumped CTMC (using compositional operators)

Conclusion and ongoing work

- objective: utilize `Maude` as an ecosystem where diverse formal models interact through a specific protocol (a distributed shared state), operating under semantics grounded in rewriting.
- parametric notion of state
- nodes might also have a hidden or private state
- inherently modular and based on a small extensible hierarchy of modules
- issues: coherence must be ensured, analysis complexity
- ongoing work: integration into graphical and multi-solution tools (**DrawNet**, **SIMTHESys**); automatic detection of symmetries to get a quotient TS and a lumped CTMC (using compositional operators)