A Maude Framework for Efficient Analysis of Multiformalism Models

Lorenzo Capra

Università degli Studi di Milano, Italy

16th International Workshop on Rewriting Logic and its Applications,
Torino, April 11-12, 2026

Motivations

Context: (performance) modeling of adaptive systems

- As systems grow in complexity, models based on a single formalism tend to be either overly complicated or overly simplistic..
- Multiformalism modeling is an approach for evaluating a system's quantitative properties by combining multiple, heterogeneous modeling methods.
- It enables selecting the most suitable formalism for every component; often used together with multi-solution approaches.
  - Some available toolsets: AToM, Möbius, OsMoSys, SIMTHESys
  - There is very limited support for system adaptation.

Motivations

Context: (performance) modeling of adaptive systems

- As systems grow in complexity, models based on a single formalism tend to be either overly complicated or overly simplistic..
- Multiformalism modeling is an approach for evaluating a system's quantitative properties by combining multiple, heterogeneous modeling methods.
- It enables selecting the most suitable formalism for every component; often used together with multi-solution approaches.
  - Some available toolsets: AToM, Möbius, OsMoSys, SIMTHESys
  - There is very limited support for system adaptation.

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

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

- Matching modulo-axioms, reflection, subtyping ...
- Logical framework for various formalisms
- Inbuilt *model-checking* facilities
- Recently supported in *probabilistic analysis*

Fundamental concept: utilizing Maude as a multiformalism framework

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

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

- Matching modulo-axioms, reflection, subtyping ...
- Logical framework for various formalisms
- Inbuilt *model-checking* facilities
- Recently supported in *probabilistic analysis*

Fundamental concept: utilizing Maude as a multiformalism framework

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

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

- Matching modulo-axioms, reflection, subtyping ...
- Logical framework for various formalisms
- Inbuilt *model-checking* facilities
- Recently supported in *probabilistic analysis*

Fundamental concept: utilizing Maude as a multiformalism framework

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

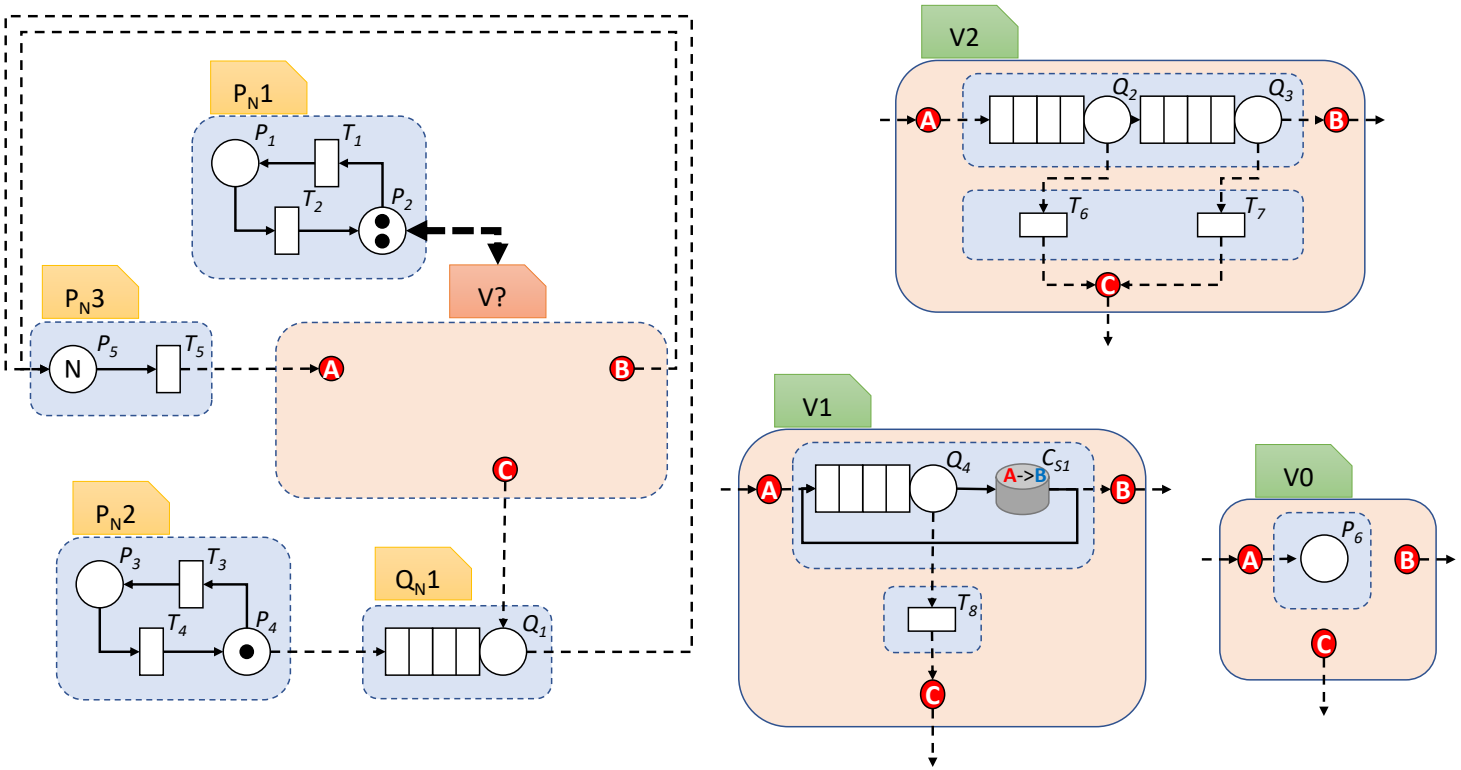
Declarative, expressive, performing, and with *rewriting logic* semantics

- Matching modulo-axioms, reflection, subtyping ...
- Logical framework for various formalisms
- Inbuilt *model-checking* facilities
- Recently supported in *probabilistic analysis*

Fundamental concept: utilizing Maude as a multiformalism framework

Example: Reconfigurable server

A server has two computing units with buffers that block new requests when full. A front-end routes requests; the second unit handles time-outs from the first. Normally, requests go through two phases, but overflow goes to a backup server during high demand. If one unit fails, the other handles both stages.



Encoding an heterogeneous Network in Maude

- A multiformalism model combines multiple components that are defined by a (colored) graph-based structure.
  - PNs, automata, activity diagrams, fault trees, queue networks, routing or scheduling protocols (JSQ), etc.
  - keeps a distributed state that is modeled as a commutative monoid
- The `NETWORK{S :: C-MONOID}` module specifies the abstract syntax of the model.
- Includes the sort `Network`, the subsort `Node`, and the AC `Network` juxtaposition `_,_`.
- "Multiset" of diverse nodes interacting through shared state elements.
- Particular node categories are connected to the network through subsort relationships (e.g., `Queue < Node`).

Encoding an heterogeneous Network in Maude

- A multiformalism model combines multiple components that are defined by a (colored) graph-based structure.
  - PNs, automata, activity diagrams, fault trees, queue networks, routing or scheduling protocols (JSQ), etc.
  - keeps a distributed state that is modeled as a commutative monoid
- The `NETWORK{S :: C-MONOID}` module specifies the abstract syntax of the model.
- Includes the sort `Network`, the subsort `Node`, and the AC `Network` juxtaposition `_,_`.
- "Multiset" of diverse nodes interacting through shared state elements.
- Particular node categories are connected to the network through subsort relationships (e.g., `Queue < Node`).

Encoding an heterogeneous Network in Maude

- A multiformalism model combines multiple components that are defined by a (colored) graph-based structure.
  - PNs, automata, activity diagrams, fault trees, queue networks, routing or scheduling protocols (JSQ), etc.
  - keeps a distributed state that is modeled as a commutative monoid
- The `NETWORK{S :: C-MONOID}` module specifies the abstract syntax of the model.
- Includes the sort `Network`, the subsort `Node`, and the AC `Network` juxtaposition `_,_`.
- "Multiset" of diverse nodes interacting through shared state elements.
- Particular node categories are connected to the network through subsort relationships (e.g., `Queue < Node`).

Encoding an heterogeneous Network in Maude

- A multiformalism model combines multiple components that are defined by a (colored) graph-based structure.
  - PNs, automata, activity diagrams, fault trees, queue networks, routing or scheduling protocols (JSQ), etc.
  - keeps a distributed state that is modeled as a commutative monoid
- The `NETWORK{S :: C-MONOID}` module specifies the abstract syntax of the model.
- Includes the sort `Network`, the subsort `Node`, and the AC `Network` juxtaposition `_,_`.
- "Multiset" of diverse nodes interacting through shared state elements.
- Particular node categories are connected to the network through subsort relationships (e.g., `Queue < Node`).

Encoding an heterogeneous Network in Maude

- A multiformalism model combines multiple components that are defined by a (colored) graph-based structure.
  - PNs, automata, activity diagrams, fault trees, queue networks, routing or scheduling protocols (JSQ), etc.
  - keeps a distributed state that is modeled as a commutative monoid
- The `NETWORK{S :: C-MONOID}` module specifies the abstract syntax of the model.
- Includes the sort `Network`, the subsort `Node`, and the AC `Network` juxtaposition `_,_`.
- "Multiset" of diverse nodes interacting through shared state elements.
- Particular node categories are connected to the network through 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
 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$Elt —> NetSys [ctor prec 125] . *** stateful network
 op netw : NetSys —> Network .
 op state : NetSys —> S$Elt .
 vars N N' : Network . var M : S$Elt .
 eq netw((N : M)) = N . eq state((N : M)) = M .
 op remove : Network Network —> Network . *** network manipulations
 eq remove((N, N'), N) = N' .
 eq remove(N, N') = N [owise] .
 ...
endfm

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

fmod NETWORK—M{PL :: TRIV} is *** network using the marking as distributed state
 protecting NETWORK{S—Pbag{PL}} .
endfm
```

Node examples: Stochastic PNs and Multi-class queues

`Net`: juxtaposition of "transitions" (labels + adjacency lists)

```
t("a", 1.5, 0) |-> [1 .  p(1) + 2 .  p(2), 1 .  p(3)] ;
t("b", 2.0, 1) |-> [1 .  p(3), 1 .  p(2) + 1 .  p(4)]
```

`Queue`: list of individual servers. Can have enabling conditions.

```
[2 .  p(1), nilP] p(1) @ 1.0 p(2) @ 1.5 > p(3)
```

New hybrid components can be straightforwardly defined

Linking nodes to the network (example: SPNs)

- We can (want to) reuse the existing signatures (e.g. `SPN-SIG`)
- The (timed) node semantics is defined by rewrite rules
- We can connect any kind of node using a simple pattern

```
fmod SPN-NODE{PL :: TRIV} is
  extending NETWORK-M{PL} .
  pr SPN-SIG{String, PL} .
  subsort Net < Node .
endfm

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

Linking nodes to the network (example: SPNs)

- We can (want to) reuse the existing signatures (e.g. `SPN-SIG`)
- The (timed) node semantics is defined by rewrite rules
- We can connect any kind of node using a simple pattern

```
fmod SPN-NODE{PL :: TRIV} is
  extending NETWORK-M{PL} .
  pr SPN-SIG{String, PL} .
  subsort Net < Node .
endfm

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

Linking nodes to the network (example: SPNs)

- We can (want to) reuse the existing signatures (e.g. `SPN-SIG`)
- The (timed) node semantics is defined by rewrite rules
- We can connect any kind of node using a simple pattern

```
fmod SPN-NODE{PL :: TRIV} is
  extending NETWORK-M{PL} .
  pr SPN-SIG{String, PL} .
  subsort Net < Node .
endfm

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

Facing the analytical complexity

- We generalize to the multiformalism framework a modular methodology that was originally developed exclusively for SPNs.
- Compositional net operators implicitly capture model symmetries ($\equiv$ graph automorphisms) using structured node labels
  - A `Network` and its constituent parts are each regarded as (recursive) monoid structures (two options: stateful or stateless)
  - We exploit the node parameterization with respect to the type of labels
  - The pattern connecting various components can be easily adapted.
  - Technically: small hierarchy of parametrized modules, parametrized views, views from theory to theory (details in the paper)

Facing the analytical complexity

- We generalize to the multiformalism framework a modular methodology that was originally developed exclusively for SPNs.
- Compositional net operators implicitly capture model symmetries ($\equiv$ graph automorphisms) using structured node labels
  - A `Network` and its constituent parts are each regarded as (recursive) monoid structures (two options: stateful or stateless)
  - We exploit the node parameterization with respect to the type of labels
  - The pattern connecting various components can be easily adapted.
  - Technically: small hierarchy of parametrized modules, parametrized views, views from theory to theory (details in the paper)

Facing the analytical complexity

- We generalize to the multiformalism framework a modular methodology that was originally developed exclusively for SPNs.
- Compositional net operators implicitly capture model symmetries ($\equiv$ graph automorphisms) using structured node labels
  - A `Network` and its constituent parts are each regarded as (recursive) monoid structures (two options: stateful or stateless)
  - We exploit the node parameterization with respect to the type of labels
  - The pattern connecting various components can be easily adapted.
  - Technically: small hierarchy of parametrized modules, parametrized views, views from theory to theory (details in the paper)

Facing the analytical complexity

- We generalize to the multiformalism framework a modular methodology that was originally developed exclusively for SPNs.
- Compositional net operators implicitly capture model symmetries ($\equiv$ graph automorphisms) using structured node labels
  - A `Network` and its constituent parts are each regarded as (recursive) monoid structures (two options: stateful or stateless)
  - We exploit the node parameterization with respect to the type of labels
  - The pattern connecting various components can be easily adapted.
  - Technically: small hierarchy of parametrized modules, parametrized views, views from theory to theory (details in the paper)

Facing the analytical complexity

- We generalize to the multiformalism framework a modular methodology that was originally developed exclusively for SPNs.
- Compositional net operators implicitly capture model symmetries ($\equiv$ graph automorphisms) using structured node labels
  - A `Network` and its constituent parts are each regarded as (recursive) monoid structures (two options: stateful or stateless)
  - We exploit the node parameterization with respect to the type of labels
  - The pattern connecting various components can be easily adapted.
  - Technically: small hierarchy of parametrized modules, parametrized views, views from theory to theory (details in the paper)

Facing the analytical complexity

- We generalize to the multiformalism framework a modular methodology that was originally developed exclusively for SPNs.
- Compositional net operators implicitly capture model symmetries ($\equiv$ graph automorphisms) using structured node labels
  - A `Network` and its constituent parts are each regarded as (recursive) monoid structures (two options: stateful or stateless)
  - We exploit the node parameterization with respect to the type of labels
  - The pattern connecting various components can be easily adapted.
  - Technically: small hierarchy of parametrized modules, parametrized views, views from theory to theory (details in the paper)

modular Network: (recursive) monoid structure

```
fth MONOID+ is *** monoid with subsort relationship
 sorts El Elt . subsort El < Elt .
 op nil : -> Elt .
 op _;_ : Elt Elt -> Elt [assoc id: nil] .
endfth

fth MO-NET is *** monoid with basic operators for manipulation of labels of network nodes
  including MONOID+ . including PSET-LAB . *** structured labeling of nodes
  op replaceWithEl : El NeLab Nat -> El .
  op addLabEl : El Lab Pset -> El .
  op placesEl : El List{String} -> Pset .
  op placesEl : El Lab -> Pset .
endfth

fmod MO-NET-OP{N :: MO-NET} is *** operations at the network level (including normalization and compositional ops)
 var J : Nat . vars K K' : NzNat . vars N N' : N$Elt . var E : N$El .
 vars L L' : Lab . var NeL : NeLab . vars W W' : String . var P : Place .
 vars S S' : Pset . var WL : List{String} . var NeWL : NeList{String} .
 op replaceWith : N$Elt NeLab Nat -> N$Elt .
 op addLab : N$Elt Lab Pset -> N$Elt .
 op places : N$Elt List{String} -> Pset .
 op places : N$Elt NeLab -> Pset .
 ...
 op abstract : N$Elt -> N$Elt . **** "name" (index) abstraction
 ...
 op repl&share : N$Elt NzNat String Pset -> N$Elt . *** replica of a node
 eq repl&share(N, K, W, S) = $repl&share(N, K, W, nil, S) .
 op $repl&share : N$Elt Nat String N$Elt Pset -> N$Elt .
 eq $repl&share(N, 0, W, N', S) = N' .
 ceq $repl&share(N,K,W,N',S) = $repl&share(N,J,W,(N' ; addLab(N,< W ; J >,S)), S) if J := sd(K,1) .
 op replica : N$Elt NzNat String -> N$Elt . *** default
 eq replica(N, K, W) = repl&share(N, K, W, emptyPset) .
endfm
```

Characterizing (sub)components as monoids

- to describe the different (sub)components as monoid-like structures, we use a standard view mechanism
- the `Network` is itself seen as a monoid through a fictional implementation of the theory's operators (late binding)
- Technically, we encapsulate the `MO-NET-OP` module (suitably instantiated) into modules with the `MOD` suffix.

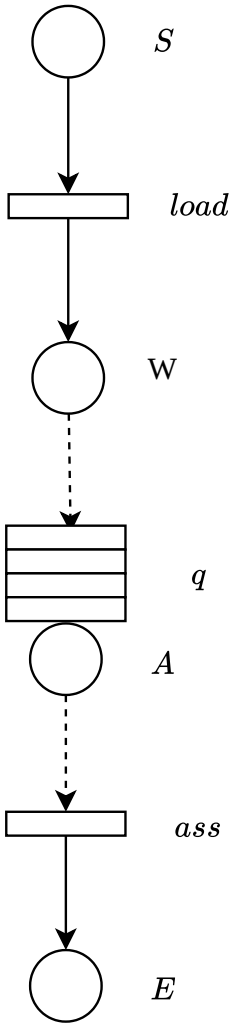
Characterizing (sub)components as monoids

- to describe the different (sub)components as monoid-like structures, we use a standard view mechanism
- the `Network` is itself seen as a monoid through a fictional implementation of the theory's operators (late binding)
- Technically, we encapsulate the `MO-NET-OP` module (suitably instantiated) into modules with the `MOD` suffix.

Characterizing (sub)components as monoids

- to describe the different (sub)components as monoid-like structures, we use a standard view mechanism
- the `Network` is itself seen as a monoid through a fictional implementation of the theory's operators (late binding)
- Technically, we encapsulate the `MO-NET-OP` module (suitably instantiated) into modules with the `MOD` suffix.

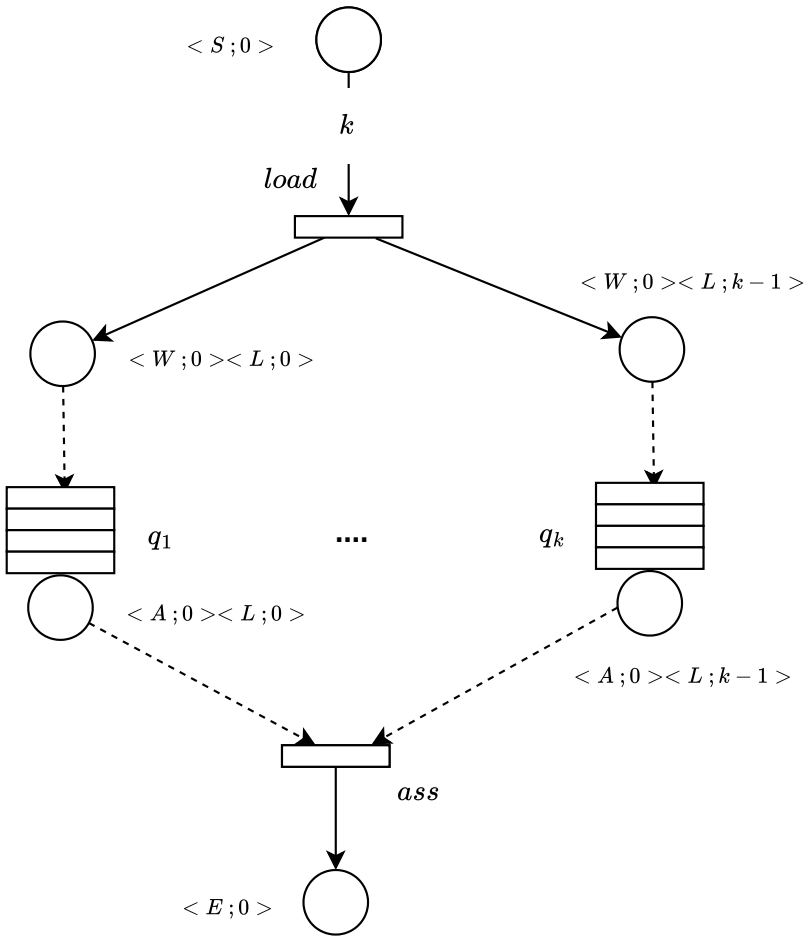
Example: Construction of a Manufacturing System



Single line (base component of a Production Line)

```
mod HNET is
 inc QUEUE-NODE-SYS-MOD .
 inc SPN-NODE-SYS-MOD{String} .
 inc JSQ-NODE-SYS-MOD .
 ops lambda gamma delta omega : -> Float .
 eq lambda = 0.5 . eq gamma = 0.2 . eq delta = 0.1 . eq omega = 1.2 .
 ops S W A E : -> Place [memo] .
 eq S = place("S") . eq W = place("W") .
 eq A = place("A") . eq W = place("E") .
 ops load, ass : -> Tran [memo] .
 op q : -> Queue [memo] .
 op line : -> Network [memo] .
 eq load = t("load", lambda, 0) |-> [1 .  S, 1 .  W] .
 eq ass = t("ass", lambda, 0) |-> [1 .  A, 1 .  E] .
 eq q = newElQueue(W,  A, delta) .
 eq line = load, q, ass .
 ...
```

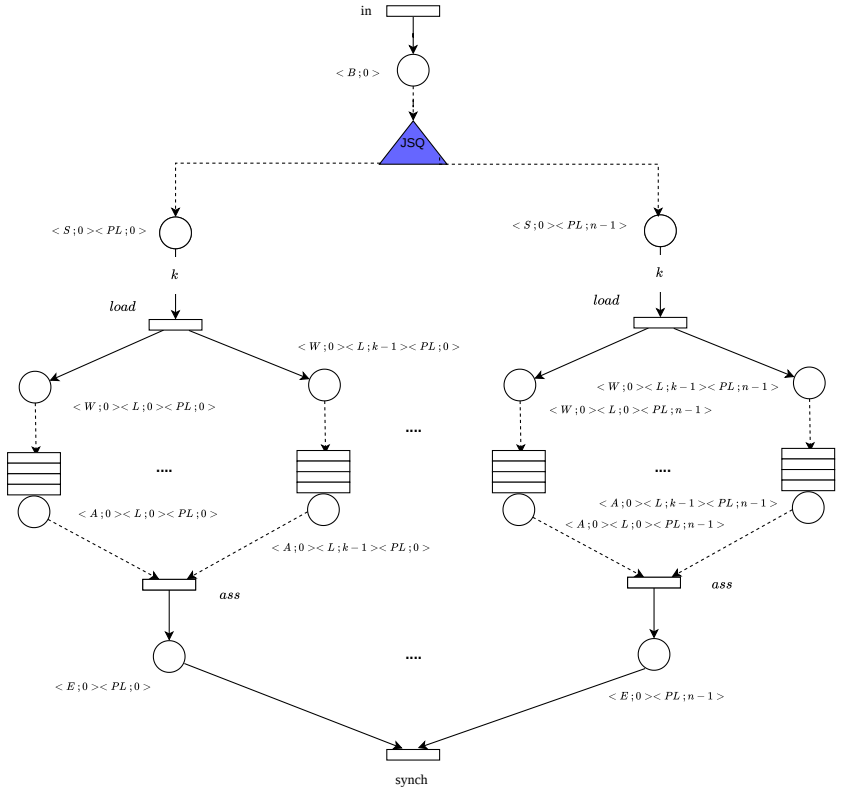
Example: Construction of a Manufacturing System



A PL made up of K similar lines

```
...
vars K N : Nat .
ops PL SL : NzNat -> Network .
*** creates K similar lines sharing S and E
eq SL(K) = repl&share(line, K, "L",  S U E) .
*** merges transitions with the same label
eq PL(K) = mergeTran(SL(K)) .
...
```

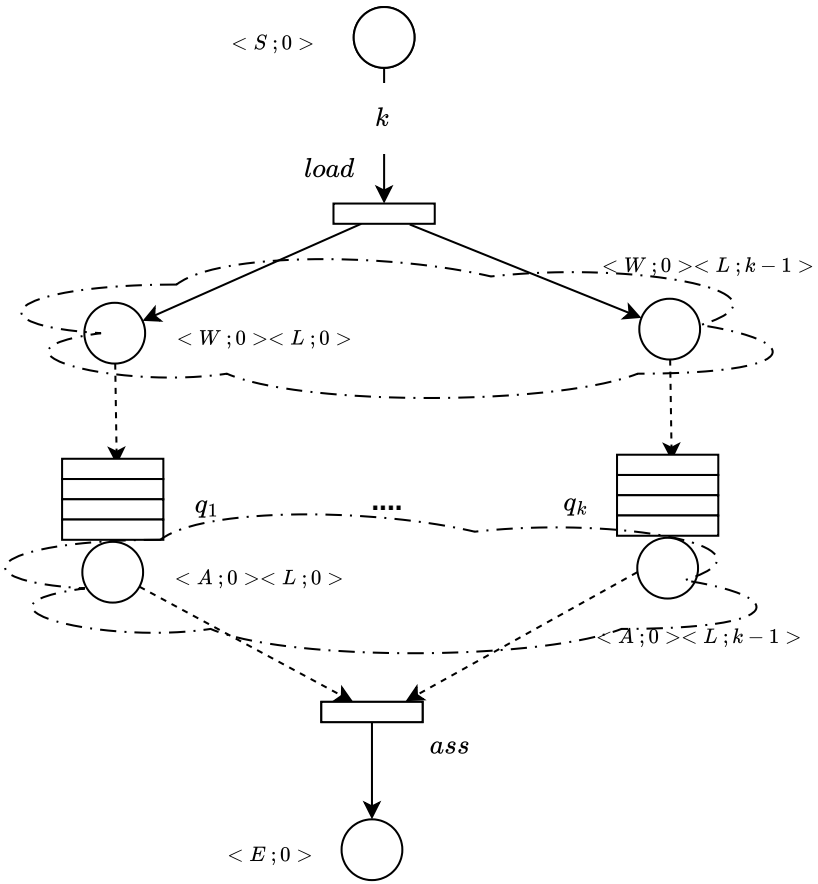
Example: Construction of a Manufacturing System



A (sub)system consisting of N PLs interconnected using a JSQ (Join the Shortest Queue) routing policy

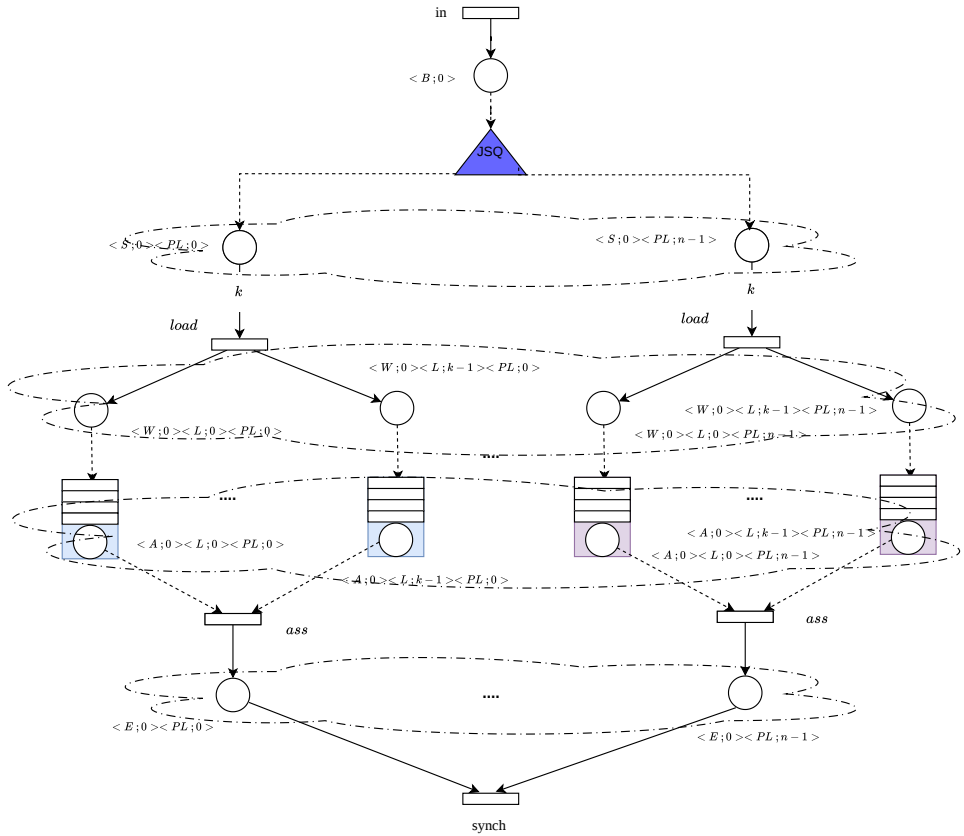
```
...
op NPL : NzNat NzNat -> Network .
eq NPL(N, K) = synch(replica(PL(K), N, "PL"), "E") .
ops k n : -> NzNat [memo] . *** model's parameters
eq k = 3 . eq n = 5 .
eq B = place("B") .
op network : -> Network [memo] .
eq network = newTout(B, "in", omega), connect&link(newJsq(B), NPL(n, k)) .
op SYS : -> NetSys .
eq SYS = network : nilP . *** empty initial state
endm
```

Graph automorphisms encoded in structured labels



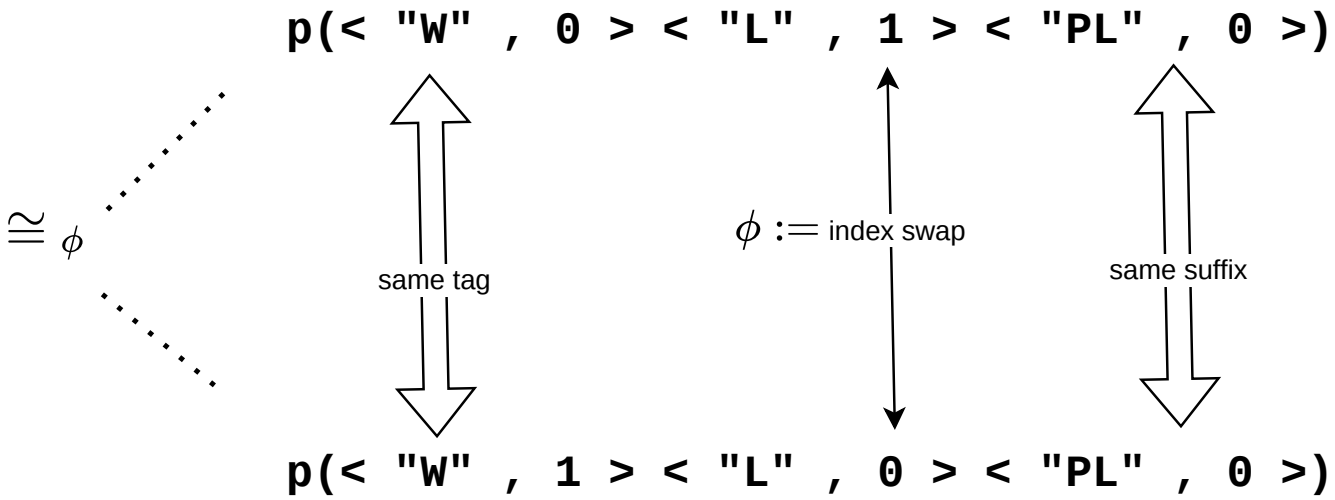
The structured labels implicitly identify automorphic sets of nodes

Graph automorphisms encoded in structured labels



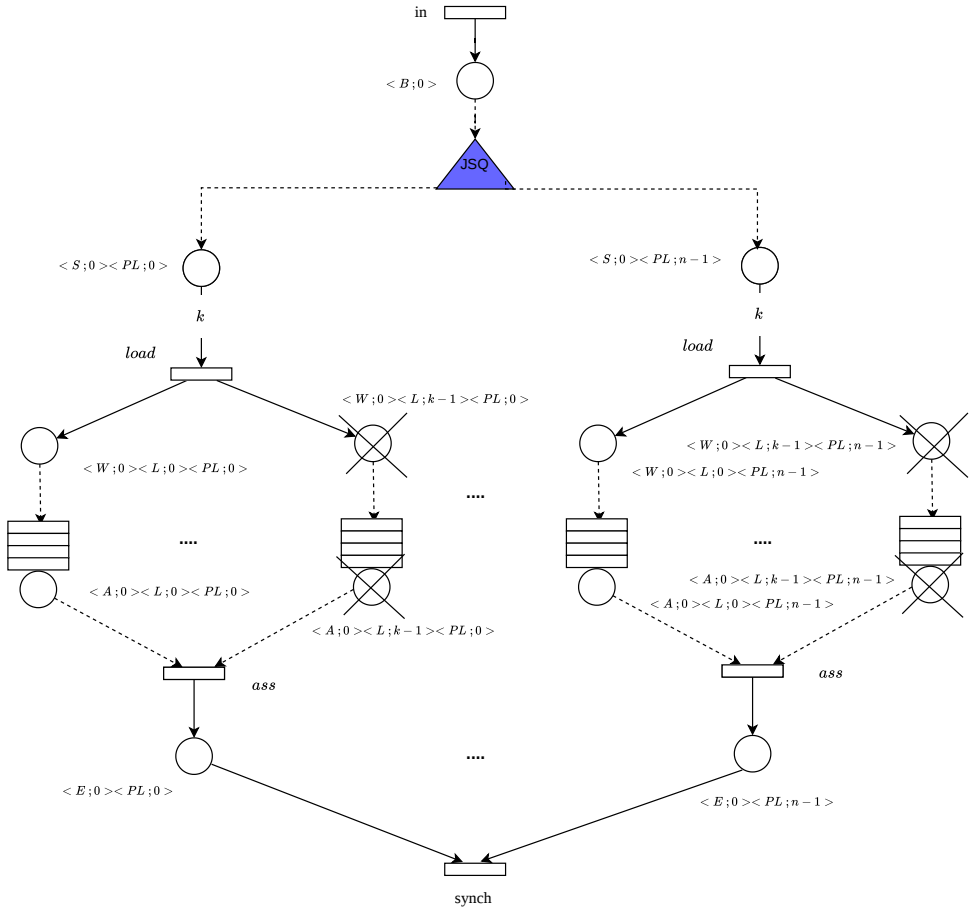
Symmetries can have a nested structure reflecting the modular layout of the model

Graph automorphisms encoded in structured labels



- Normalization (name abstraction + state minimization) : based on simple index exchange (arbitrary sequence, no backtracking).
- Far more efficient than classical graph canonization.
- Effective: points out "nested" symmetries.

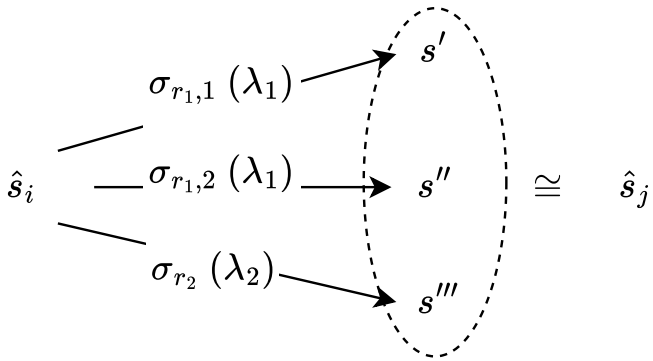
Network rewrites must retain symmetries



Network rewritings are performed with operators that preserve the symmetry encoded in the node labeling

```
*** removes all nodes with labels characterized by a given suffix
op remove : Network Elab List{String} -> Network .
op cond : Network -> [Bool] .
...
var NET : Network .
crl NET  => normalize(remove(NET, < "L" ; k - 1 >, "PL")) if cond(NET) .
```

Lumped CTMC derivation



same normal form

Issue

the state transitions triggered by rule r_1 are "merged", lacking an explicit indication of rule-match occurrences.

The exact state transition rate in the lumped CTMC is: $q_{i,j} := 2\lambda_1 + \lambda_2$

Issue

if $\sigma_{r_1,1} \equiv \sigma_{r_1,2}$ (substitutions equivalent modulo a permutation of variables)

the state transition rate should be: $q_{i,j} := \lambda_1 + \lambda_2$

We introduced a method that pre-processes Maude system modules to generate an enriched state representation, which includes exact information about matches, consequently, state transition rates. [L. Capra, SimulTech 2025, Bilbao]

Experimental evidence: Transition System build time

```
Maude> search in PLsys(N,2) =>! F:NetSys .
```

N	# states (quotient)	sec	# states (conventional)	sec	*thr*
2	451	0	1915	0	5.68
4	2123	3	124,148	13	13.79
6	4855	10	4,587,426	970	15.06
8	8643	29	-	†	17.89
10	13487	66	-	†	19.90

Table: Conventional vs quotient TS – † time out

Conclusion

- we introduce a Maude-based ecosystem in which heterogeneous models interact via a distributed state, specified parametrically
- approach is by design modular and adaptable: network components are specified as recursive monoid-based structures, which may be either stateful or stateless
- detection of symmetries (graph automorphisms) using compositional operators and structured node labels $\Rightarrow$ TS quotient (lumped CTMC)
- future work
  - integration into (graphical) multiformalism tools (**DrawNet**, **SIMTHESys**)
  - use of abstractions preserving the symmetry structure (lumpability) with suitable stochastic approximations

Conclusion

- we introduce a Maude-based ecosystem in which heterogeneous models interact via a distributed state, specified parametrically
- approach is by design modular and adaptable: network components are specified as recursive monoid-based structures, which may be either stateful or stateless
- detection of symmetries (graph automorphisms) using compositional operators and structured node labels $\Rightarrow$ TS quotient (lumped CTMC)
- future work
  - integration into (graphical) multiformalism tools (**DrawNet**, **SIMTHESys**)
  - use of abstractions preserving the symmetry structure (lumpability) with suitable stochastic approximations

Conclusion

- we introduce a Maude-based ecosystem in which heterogeneous models interact via a distributed state, specified parametrically
- approach is by design modular and adaptable: network components are specified as recursive monoid-based structures, which may be either stateful or stateless
- detection of symmetries (graph automorphisms) using compositional operators and structured node labels $\Rightarrow$ TS quotient (lumped CTMC)
- future work
  - integration into (graphical) multiformalism tools (**DrawNet**, **SIMTHESys**)
  - use of abstractions preserving the symmetry structure (lumpability) with suitable stochastic approximations

Conclusion

- we introduce a Maude-based ecosystem in which heterogeneous models interact via a distributed state, specified parametrically
- approach is by design modular and adaptable: network components are specified as recursive monoid-based structures, which may be either stateful or stateless
- detection of symmetries (graph automorphisms) using compositional operators and structured node labels $\Rightarrow$ TS quotient (lumped CTMC)
- future work
  - integration into (graphical) multiformalism tools (**DrawNet**, **SIMTHESys**)
  - use of abstractions preserving the symmetry structure (lumpability) with suitable stochastic approximations

Conclusion

- we introduce a Maude-based ecosystem in which heterogeneous models interact via a distributed state, specified parametrically
- approach is by design modular and adaptable: network components are specified as recursive monoid-based structures, which may be either stateful or stateless
- detection of symmetries (graph automorphisms) using compositional operators and structured node labels $\Rightarrow$ TS quotient (lumped CTMC)
- future work
  - integration into (graphical) multiformalism tools (**DrawNet**, **SIMTHESys**)
  - use of abstractions preserving the symmetry structure (lumpability) with suitable stochastic approximations

Conclusion

- we introduce a Maude-based ecosystem in which heterogeneous models interact via a distributed state, specified parametrically
- approach is by design modular and adaptable: network components are specified as recursive monoid-based structures, which may be either stateful or stateless
- detection of symmetries (graph automorphisms) using compositional operators and structured node labels $\Rightarrow$ TS quotient (lumped CTMC)
- future work
  - integration into (graphical) multiformalism tools (**DrawNet**, **SIMTHESys**)
  - use of abstractions preserving the symmetry structure (lumpability) with suitable stochastic approximations

Thanks for your attention!

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