A Modular Model Checker for Reference Nets: MoMoC

Sven Willrodt, Daniel Moldt and Michael Simon

17.06.2020

University of Hamburg
Faculty of Mathematics, Informatics and Natural Sciences
Department of Informatics
http://www.informatik.uni-hamburg.de/TGI/

Content

Introduction

Introduction

MoMoC is a novel Model Checking tool for Reference nets, featuring a modular architecture.

Introduction

MoMoC is a novel Model Checking tool for Reference nets, featuring a modular architecture.

MoMoC pursues two goals:

Introduction

MoMoC is a novel Model Checking tool for Reference nets, featuring a modular architecture.

MoMoC pursues two goals:

- Teaching model checking
- Extensibility, to form a basis for further research on model checking of Reference nets

Reference Nets

The Java Reference Net Formalism

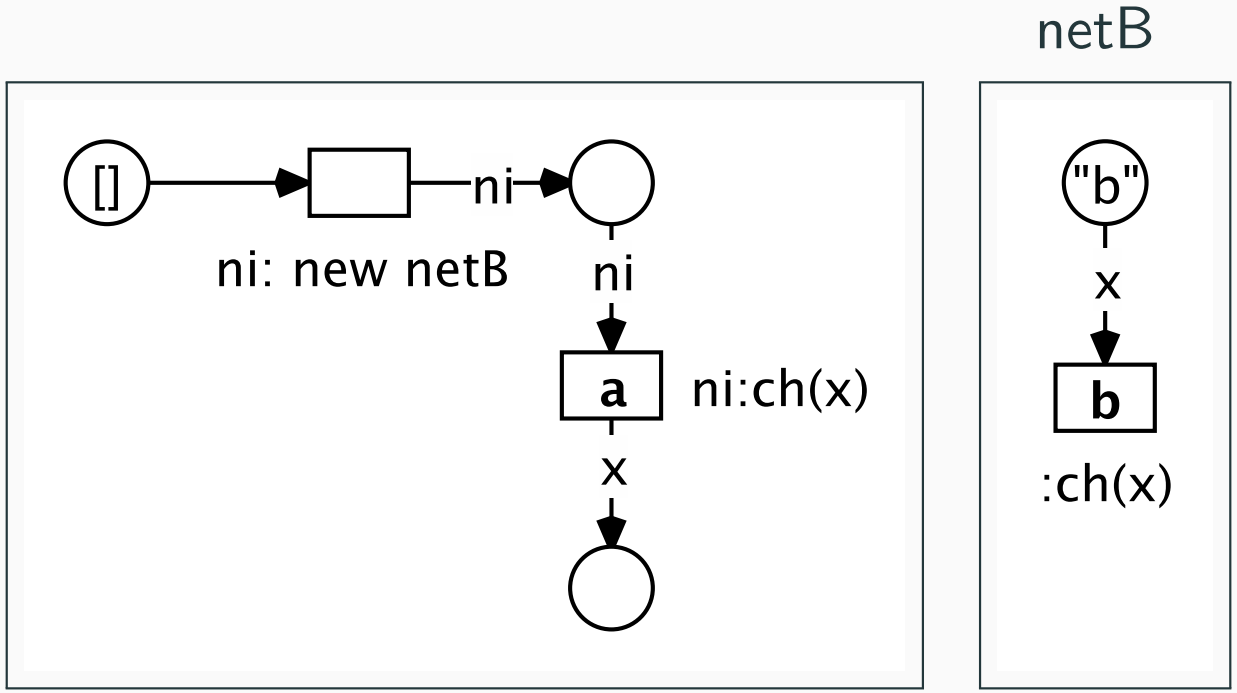
- Coloured Petri net (CPN) formalism
- Primary formalism of the Renew simulator
- Java code inscriptions
- Tokens: Java objects or *net instances*
- Interaction: *synchronous channels*

Net Instance Tokens

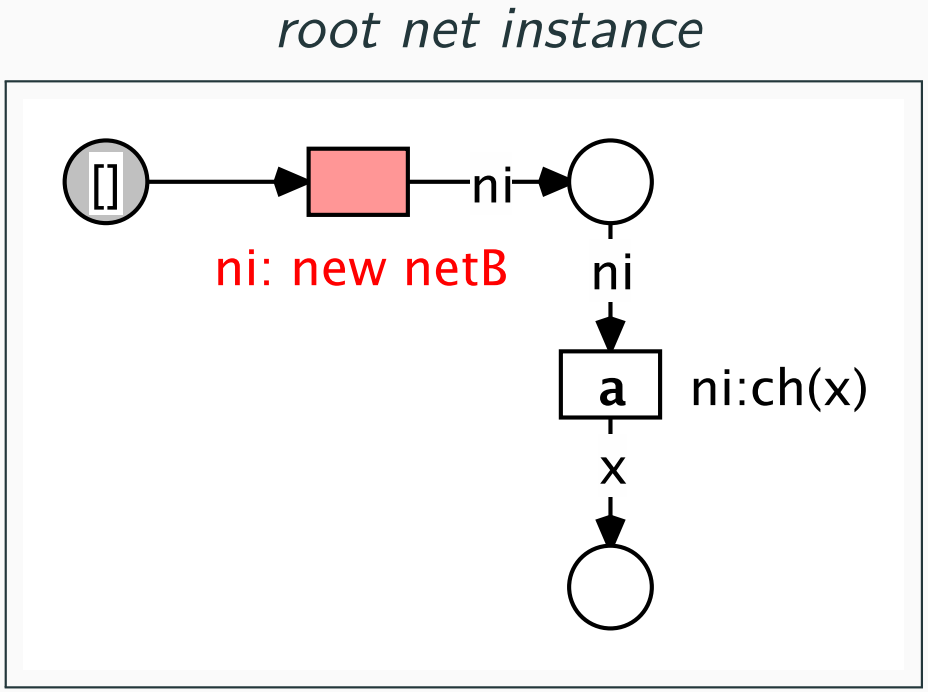
- nets-within-nets
- follows the object-oriented paradigm

net template	class
net instance	instance/object
net elements	internals of a class
uplinks of a net	methods/interface of a class
invoking an uplink	invoking a method

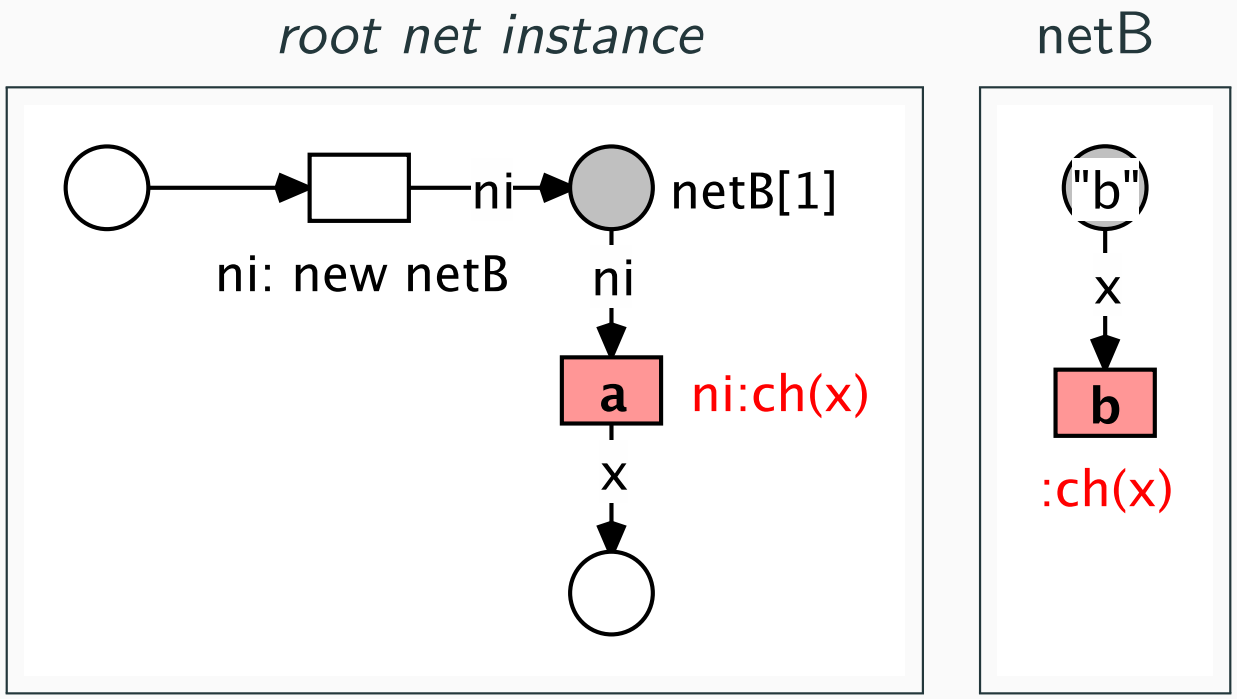
Java Reference Net Example



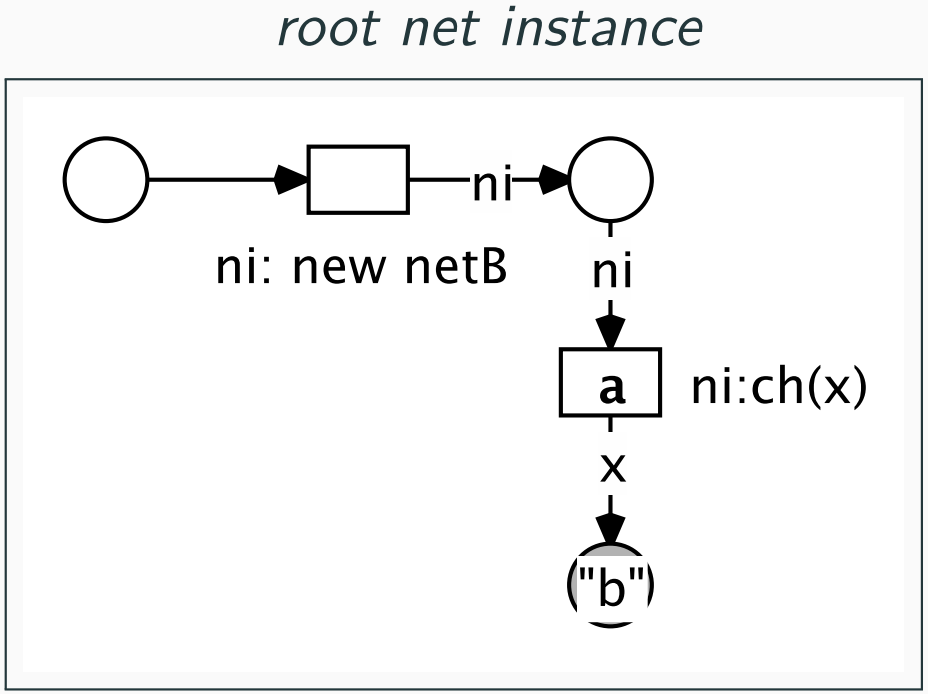
Java Reference Net Example



Java Reference Net Example



Java Reference Net Example



Features

Features

Modular Model Checker (MoMoC)

- Explicit CTL-Model Checking for Reference Nets
- Parsing
- Result visualization (exploration, colorization, layouting)
- Net Instance Quantifier
- Simpler net formalisms (P/T nets, CPNs) can be treated as flat Reference nets

Atomic Propositions

- FIREABLE(T)
- DEADLOCK

Atomic Propositions

- FIREABLE(T)
- DEADLOCK
- Marking predicates..

Net-Instance-Quantifier

Problem: During runtime, net instances are not uniquely identifiable by a name that must be entered before runtime.

Net-Instance-Quantifier

Problem: During runtime, net instances are not uniquely identifiable by a name that must be entered before runtime.

Proposed solution: Net-Instance-Quantifier

Net-Instance-Quantifier

Problem: During runtime, net instances are not uniquely identifiable by a name that must be entered before runtime.

Proposed solution: Net-Instance-Quantifier

$$!(\mathit{Net}, p) \equiv \text{Every net instance of the template } \mathit{Net} \text{ satisfies } p.$$

$$?(\mathit{Net}, p) \equiv \text{There exists a net instance of the template } \mathit{Net} \text{ that satisfies } p.$$

Net-Instance-Quantifier

Problem: During runtime, net instances are not uniquely identifiable by a name that must be entered before runtime.

Proposed solution: Net-Instance-Quantifier

$$!(\mathit{Net}, p) \equiv \text{Every net instance of the template } \mathit{Net} \text{ satisfies } p.$$

$$?(\mathit{Net}, p) \equiv \text{There exists a net instance of the template } \mathit{Net} \text{ that satisfies } p.$$

Scales independently of the size of the reachability graph, however net instances cannot be tracked over multiple states.

Parsing

Uses ANTLR as a framework for parsing.

Parsing features of MoMoC:

- Parsing of different notations
- Normalization
- Reduction
- Encoding

Result Visualization

Goal: Comprehensive results that help teaching (CTL) Model Checking

Result Visualization

Goal: Comprehensive results that help teaching (CTL) Model Checking

- States of the RG can be explored
- RG can interactively be colorized with results of subroutines

Architecture

Modules

Goal: An extensible architecture that allows quick prototyping.

Modules

Goal: An extensible architecture that allows quick prototyping. Query is handled by an interaction of three types of interchangeable modules.

- **Binding Core** - Finds bindings and calculates successive markings, thus defines the semantics

Modules

Goal: An extensible architecture that allows quick prototyping. Query is handled by an interaction of three types of interchangeable modules.

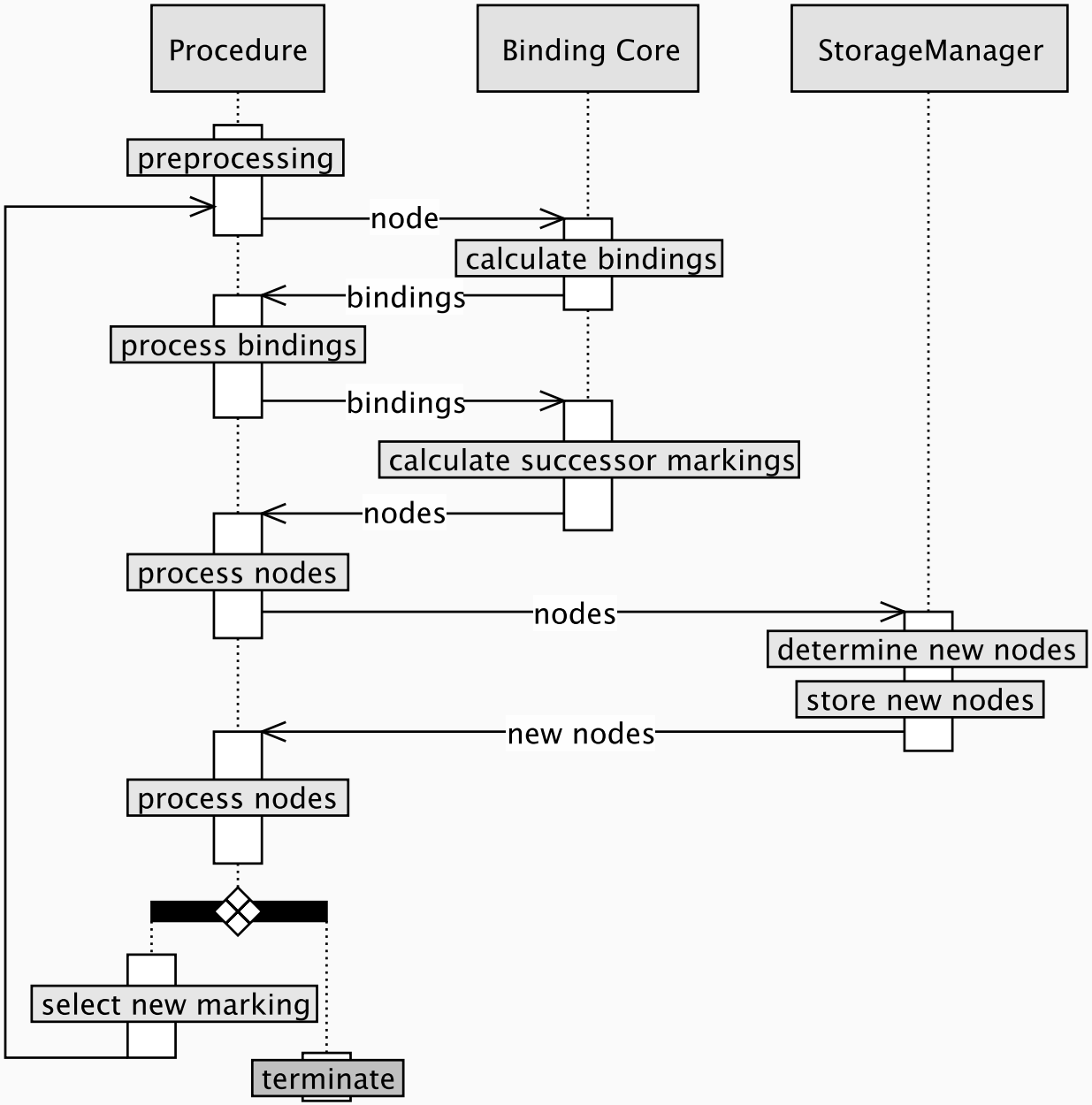
- **Binding Core** - Finds bindings and calculates successive markings, thus defines the semantics
- **Storage Manager** - Stores the reachability graph and finds cycles in the graph

Modules

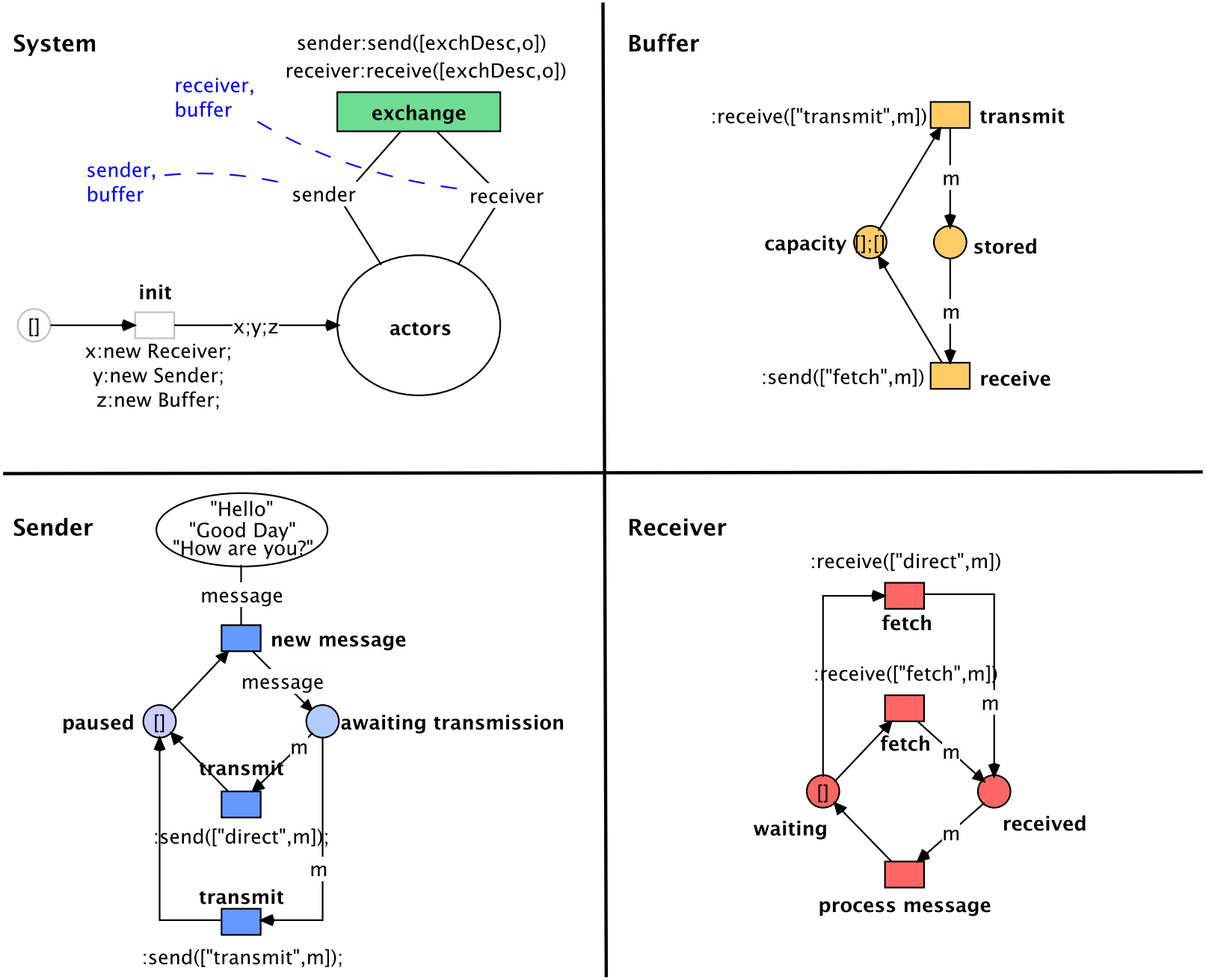
Goal: An extensible architecture that allows quick prototyping. Query is handled by an interaction of three types of interchangeable modules.

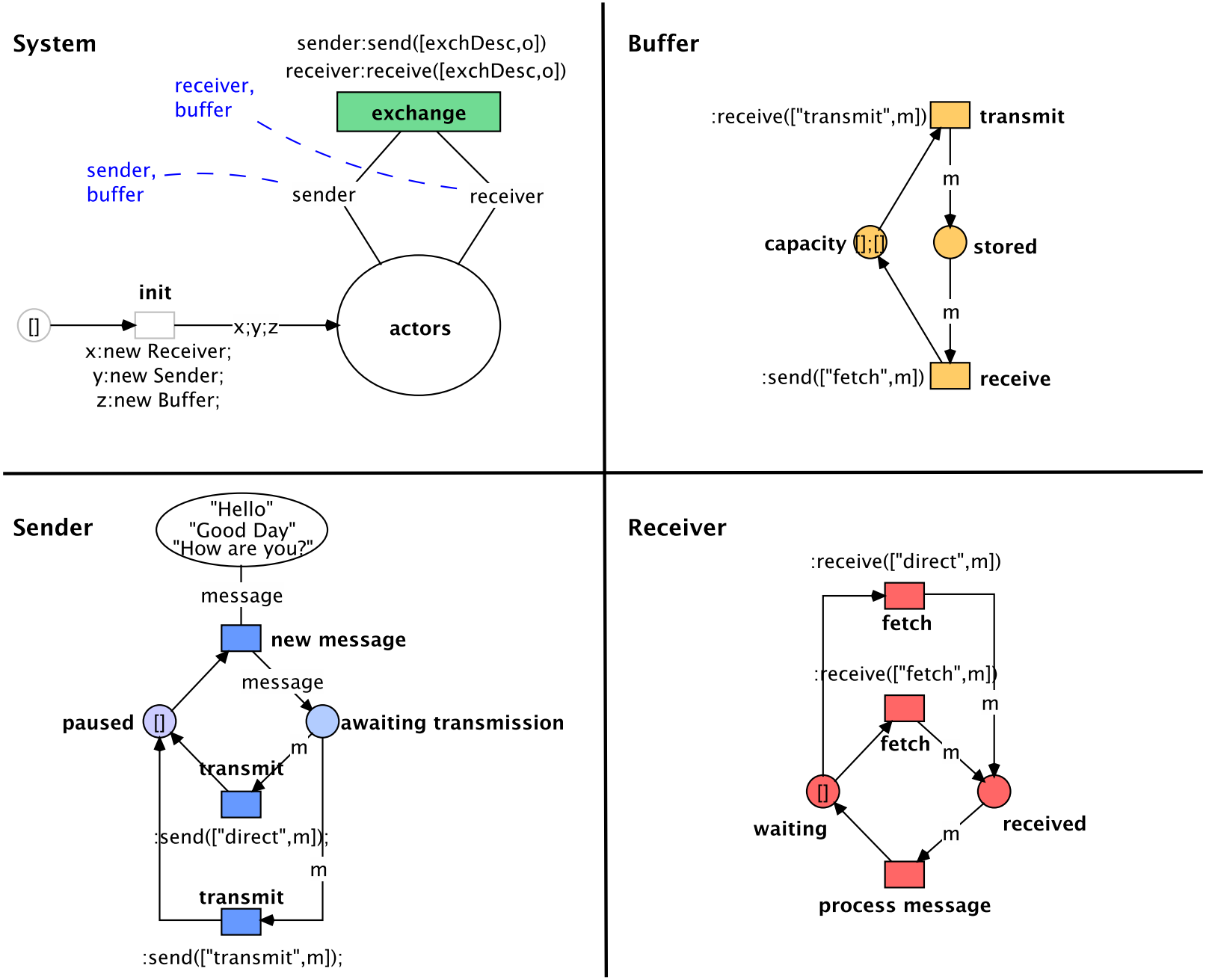
- **Binding Core** - Finds bindings and calculates successive markings, thus defines the semantics
- **Storage Manager** - Stores the reachability graph and finds cycles in the graph
- **Procedure** - Contains logic and steps to process a query

Module Interaction



Demo


System
sender:send([exchDesc,o])
receiver:receive([exchDesc,o])
receiver,
buffer
exchange
sender,
buffer
sender
receiver
init
[]
x;y;z
actors
x:new Receiver;
y:new Sender;
z:new Buffer;
Buffer
:receive(["transmit",m])
transmit
m
capacity [];[]
stored
m
:send(["fetch",m])
receive
Sender
"Hello"
"Good Day"
"How are you?"
message
new message
message
paused
[]
awaiting transmission
m
transmit
:send(["direct",m]);
m
transmit
:send(["transmit",m]);
Receiver
:receive(["direct",m])
fetch
:receive(["fetch",m])
m
fetch
m
[]
received
waiting
m
process message



$$EG(!(Buffer, m(stored) = 0) \wedge AF?(Receiver, m(received) = 1))$$

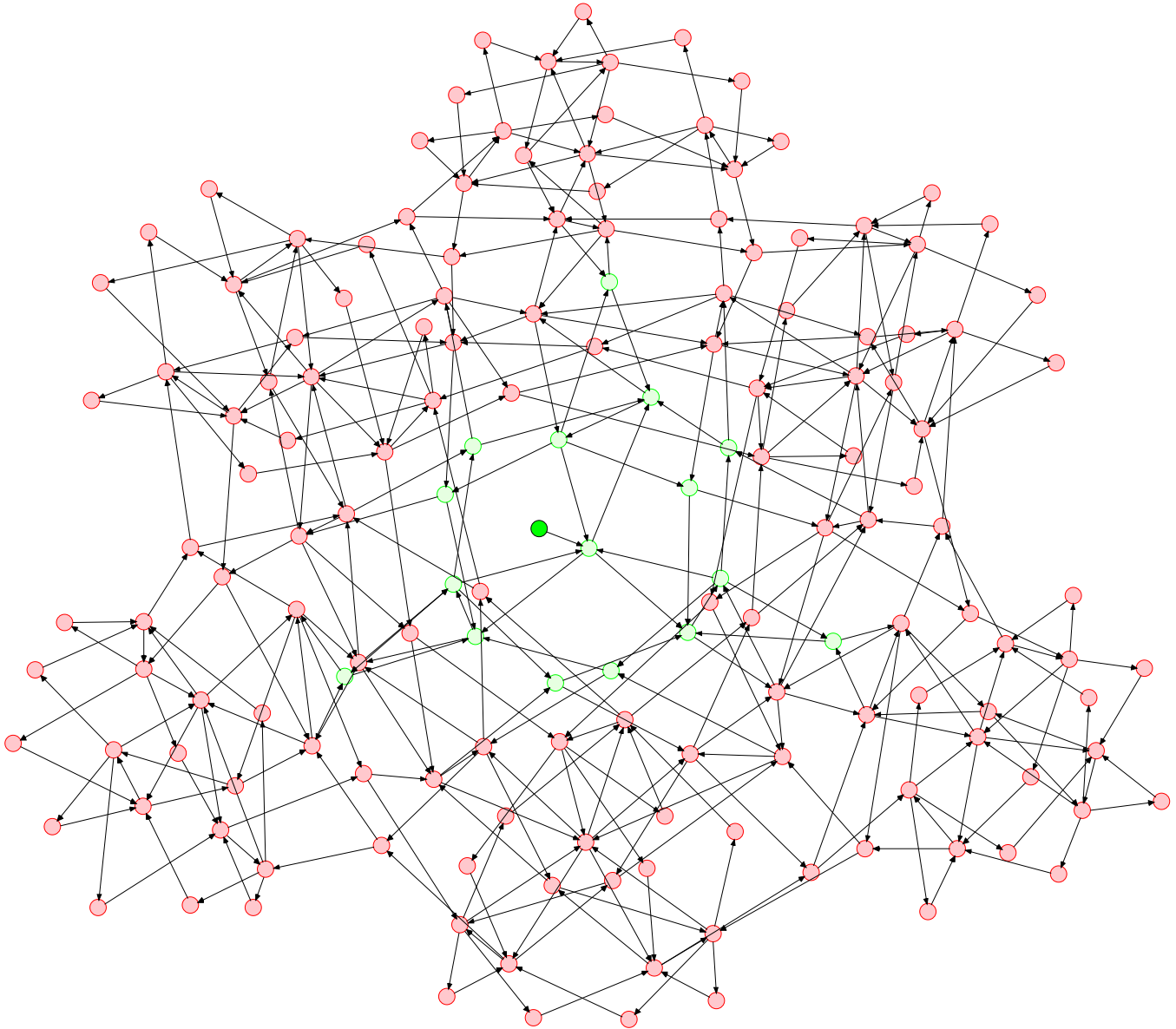
Result

EG[AND[![Buffer, m[stored]=0],NOT[EG[NOT[?[Receiver, m[received]=1]]]]]]

- AND[![Buffer, m[stored]=0],NOT[EG[NOT[?[Receiver, m[received]=1]]]]]
  - ![Buffer, m[stored]=0]
  - NOT[EG[NOT[?[Receiver, m[received]=1]]]]
    - EG[NOT[?[Receiver, m[received]=1]]]
      - NOT[?[Receiver, m[received]=1]]
        - ?[Receiver, m[received]=1]

Specification	EG(!(Buffer,m(stored)=0) ∧ AF?(Receiver,m(received)=1))
Result:	TRUE
Root Net:	System[10786]
Normalized Spec.:	EG[AND[![Buffer, m[stored]=0],NOT[EG[NOT[?[Receiver, m[received]=1]]]]]]

Show Reachability Graph



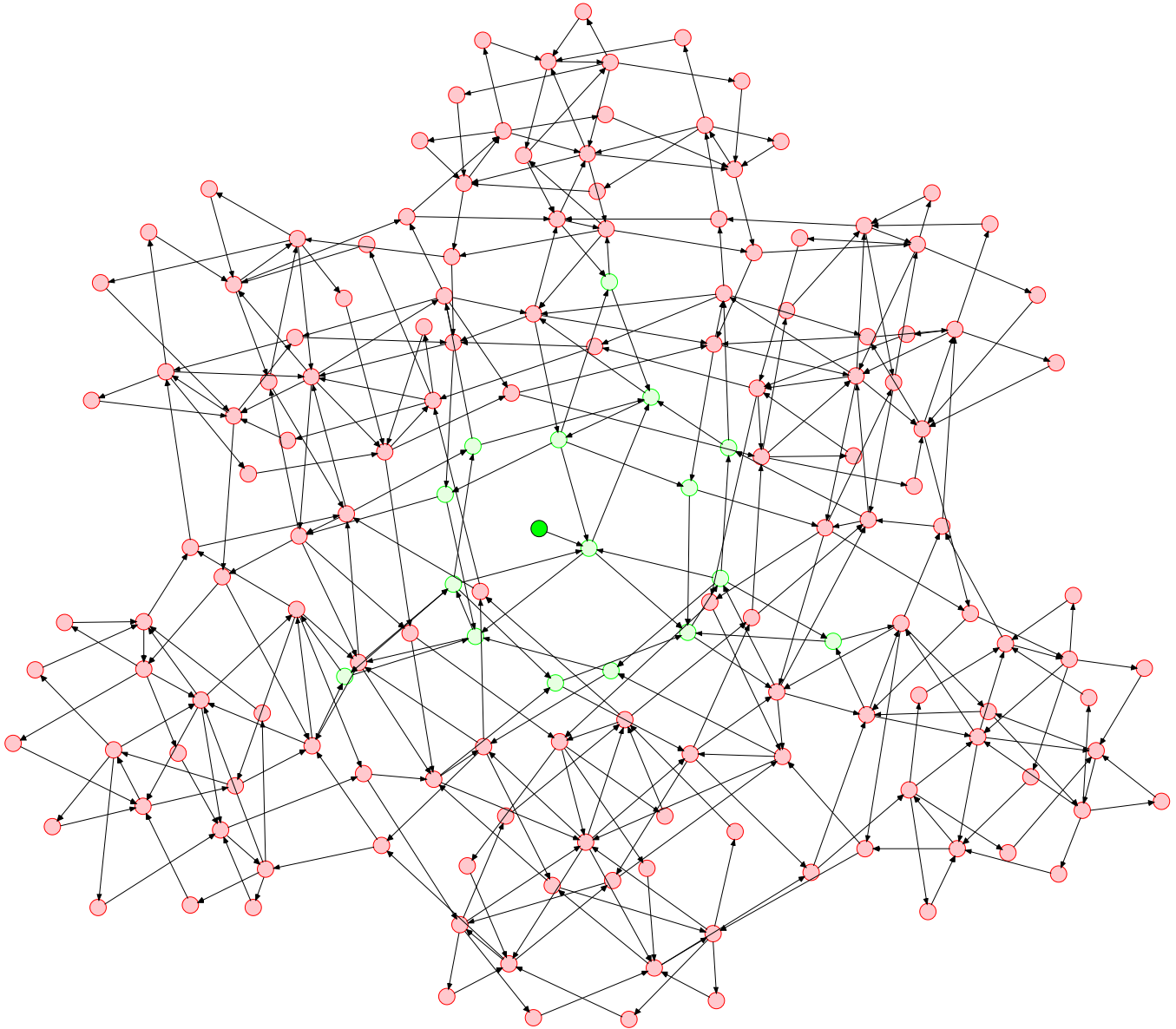
Result

EG[AND[![Buffer, m[stored]=0],NOT[EG[NOT[?[Receiver, m[received]=1]]]]]]

- AND[![Buffer, m[stored]=0],NOT[EG[NOT[?[Receiver, m[received]=1]]]]]
  - ![Buffer, m[stored]=0]
  - NOT[EG[NOT[?[Receiver, m[received]=1]]]]
    - EG[NOT[?[Receiver, m[received]=1]]]
      - NOT[?[Receiver, m[received]=1]]
        - ?[Receiver, m[received]=1]

Specification	EG(!(Buffer,m(stored)=0) ∧ AF?(Receiver,m(received)=1))
Result:	TRUE
Root Net:	System[10786]
Normalized Spec.:	EG[AND[![Buffer, m[stored]=0],NOT[EG[NOT[?[Receiver, m[received]=1]]]]]]

Show Reachability Graph



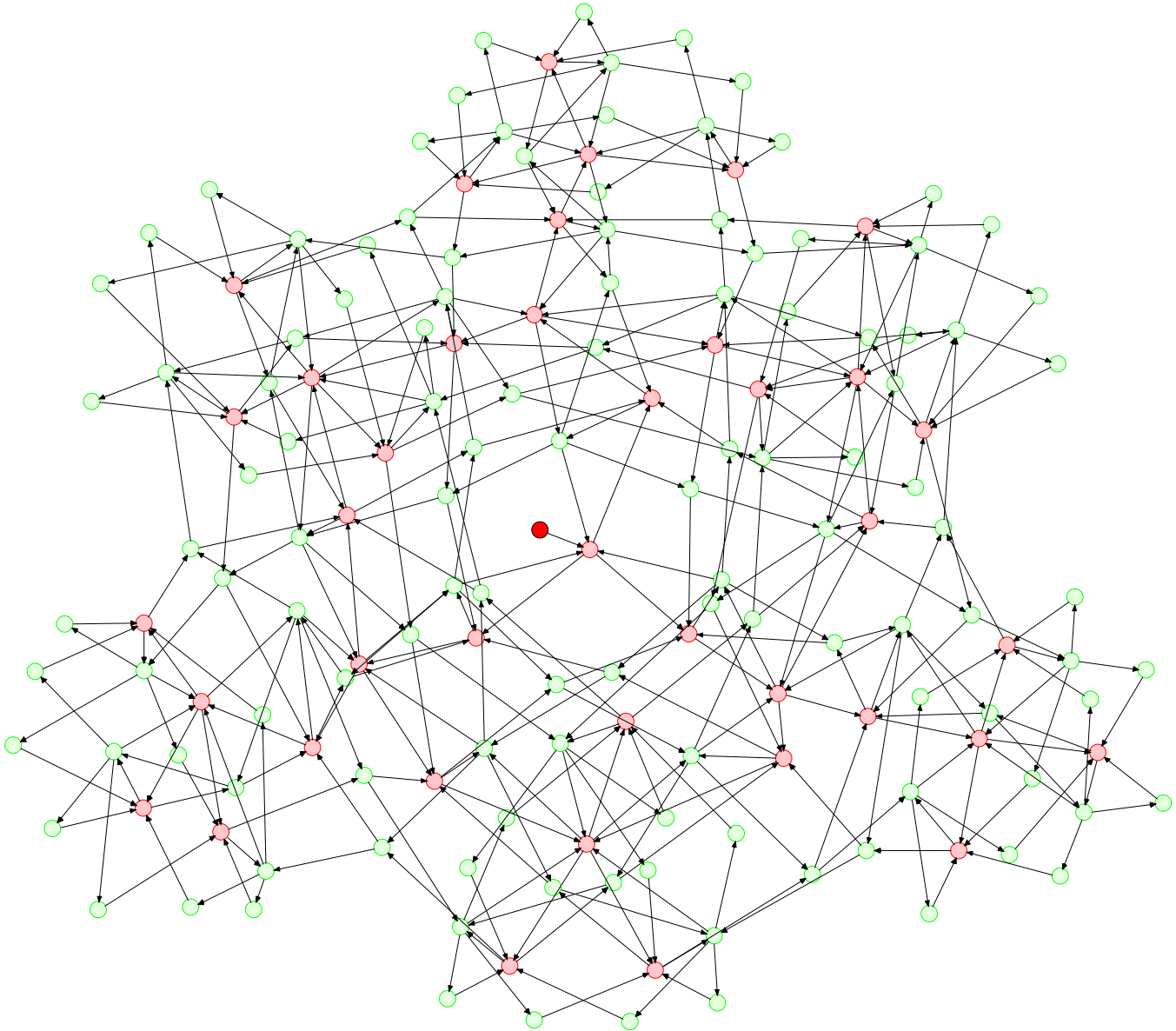
Result

EG[AND[![Buffer, m[stored]=0],NOT[EG[NOT[?[Receiver, m[received]=1]]]]]]

- AND[![Buffer, m[stored]=0],NOT[EG[NOT[?[Receiver, m[received]=1]]]]]
  - ![Buffer, m[stored]=0]
  - NOT[EG[NOT[?[Receiver, m[received]=1]]]]
    - EG[NOT[?[Receiver, m[received]=1]]]
      - NOT[?[Receiver, m[received]=1]]
        - ?[Receiver, m[received]=1]

Specification	EG(!(Buffer,m(stored)=0) ∧ AF?(Receiver,m(received)=1))
Result:	TRUE
Root Net:	System[10786]
Normalized Spec.:	EG[AND[![Buffer, m[stored]=0],NOT[EG[NOT[?[Receiver, m[received]=1]]]]]]

Show Reachability Graph



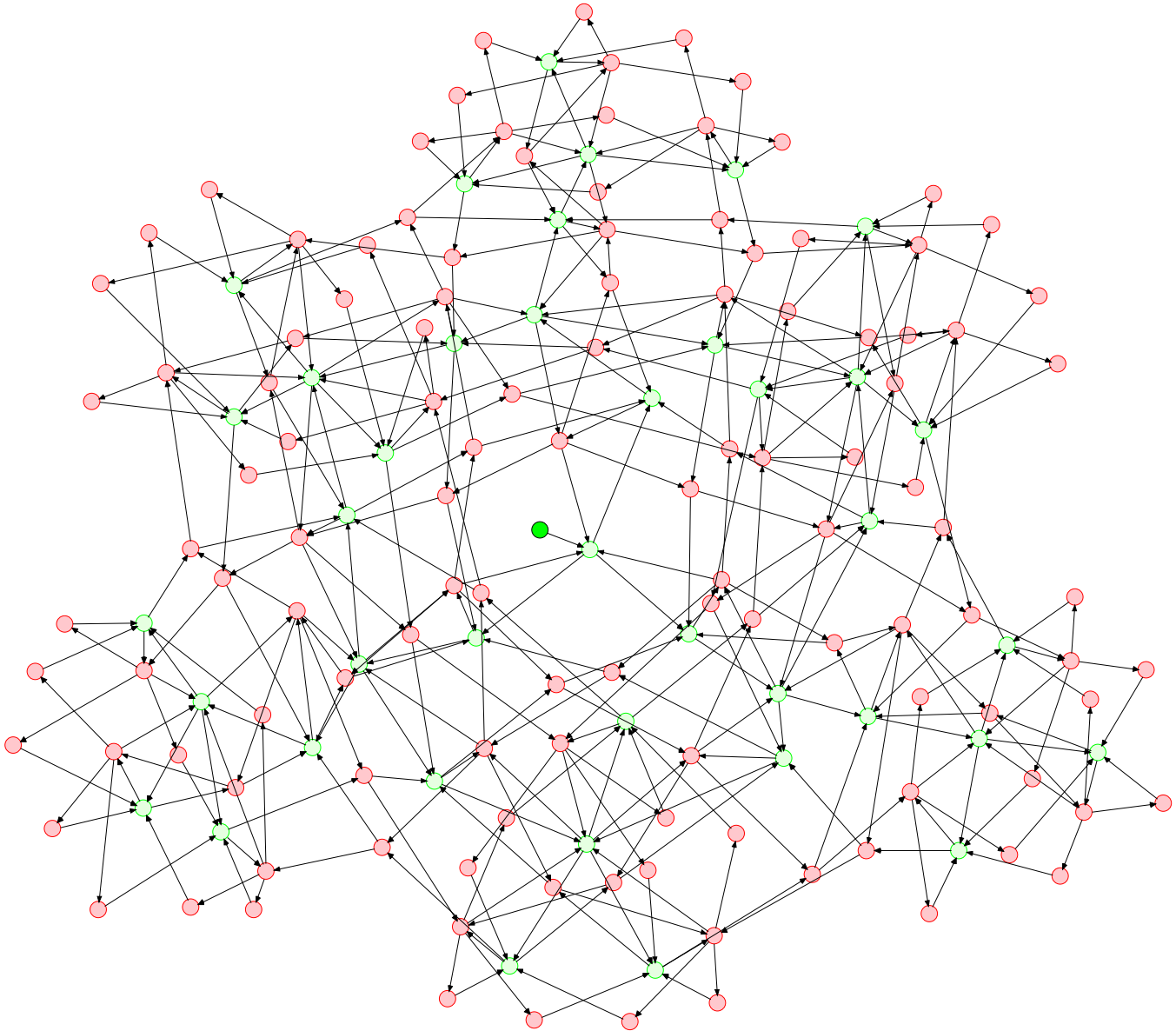
Result

EG[AND[![Buffer, m[stored]=0],NOT[EG[NOT[?[Receiver, m[received]=1]]]]]]

- AND[![Buffer, m[stored]=0],NOT[EG[NOT[?[Receiver, m[received]=1]]]]]
  - ![Buffer, m[stored]=0]
  - NOT[EG[NOT[?[Receiver, m[received]=1]]]]
    - EG[NOT[?[Receiver, m[received]=1]]]
      - NOT[?[Receiver, m[received]=1]]
        - ?[Receiver, m[received]=1]

Specification	EG(!(Buffer,m(stored)=0) ∧ AF?(Receiver,m(received)=1))
Result:	TRUE
Root Net:	System[10786]
Normalized Spec.:	EG[AND[![Buffer, m[stored]=0],NOT[EG[NOT[?[Receiver, m[received]=1]]]]]]

Show Reachability Graph



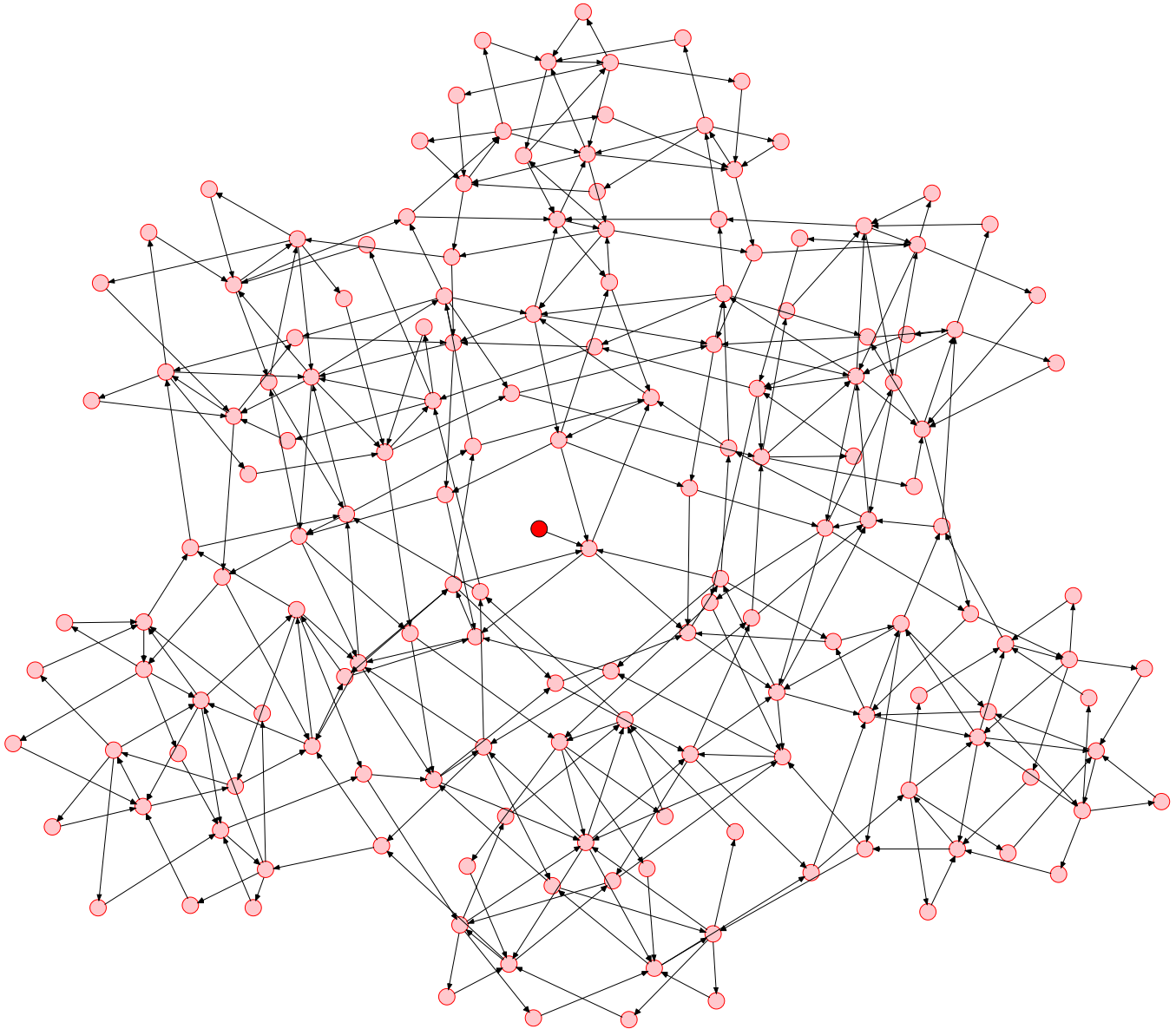
Result

EG[AND[![Buffer, m[stored]=0],NOT[EG[NOT[?[Receiver, m[received]=1]]]]]]

- AND[![Buffer, m[stored]=0],NOT[EG[NOT[?[Receiver, m[received]=1]]]]]
  - ![Buffer, m[stored]=0]
  - NOT[EG[NOT[?[Receiver, m[received]=1]]]]
    - EG[NOT[?[Receiver, m[received]=1]]]
      - NOT[?[Receiver, m[received]=1]]
        - ?[Receiver, m[received]=1]

Specification	EG(!(Buffer,m(stored)=0) ∧ AF?(Receiver,m(received)=1))
Result:	TRUE
Root Net:	System[10786]
Normalized Spec.:	EG[AND[![Buffer, m[stored]=0],NOT[EG[NOT[?[Receiver, m[received]=1]]]]]]

Show Reachability Graph



Result

EG[AND[![Buffer, m[stored]=0],NOT[EG[NOT[?[Receiver, m[received]=1]]]]]]

▼ AND[![Buffer, m[stored]=0],NOT[EG[NOT[?[Receiver, m[received]=1]]]]]

  ![Buffer, m[stored]=0]

  ▼ NOT[EG[NOT[?[Receiver, m[received]=1]]]]

    ▼ EG[NOT[?[Receiver, m[received]=1]]]

      ▼ NOT[?[Receiver, m[received]=1]]

        ?[Receiver, m[received]=1]

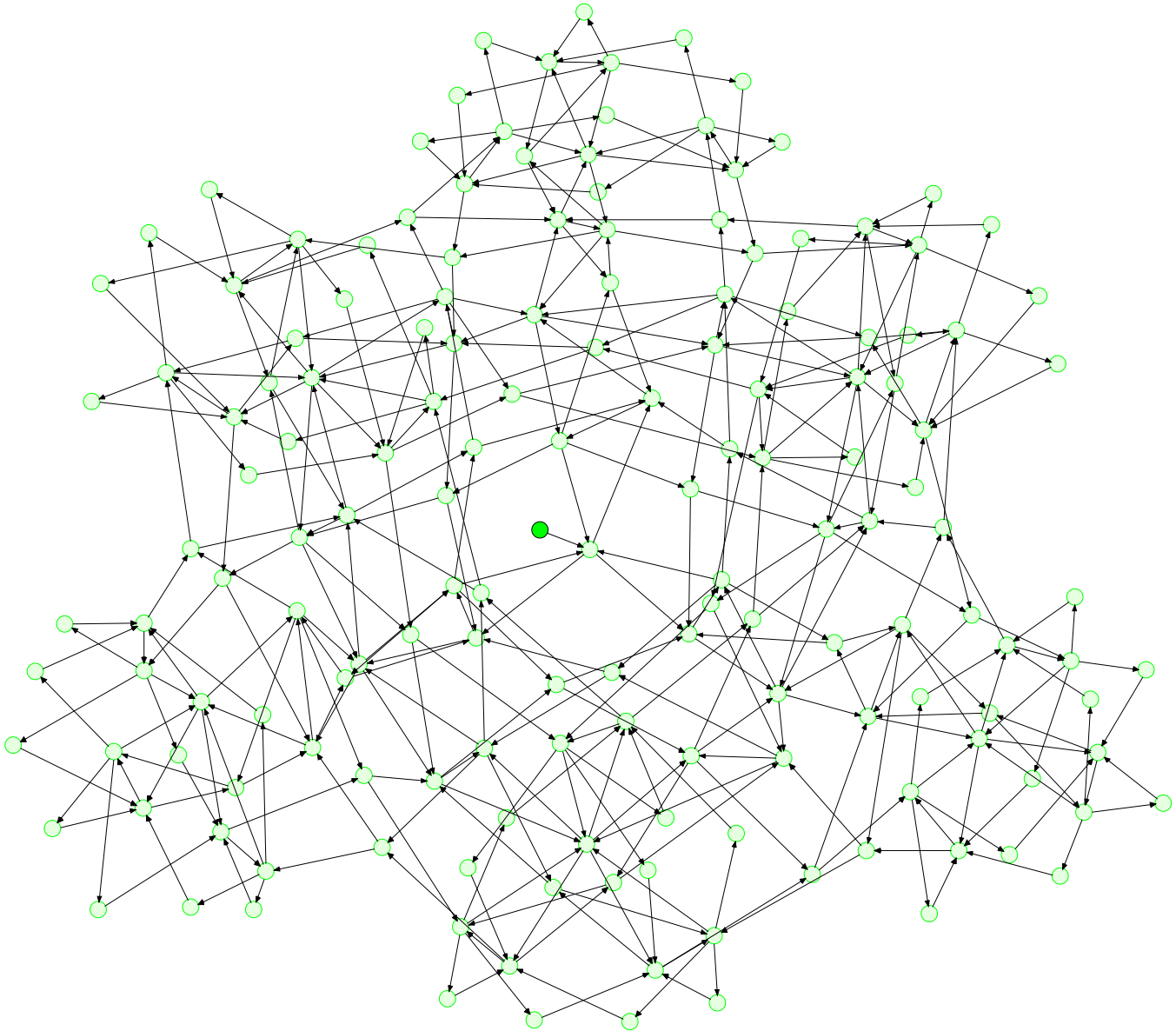
Specification: EG(!(Buffer,m(stored)=0) ∧ AF?(Receiver,m(received)=1))

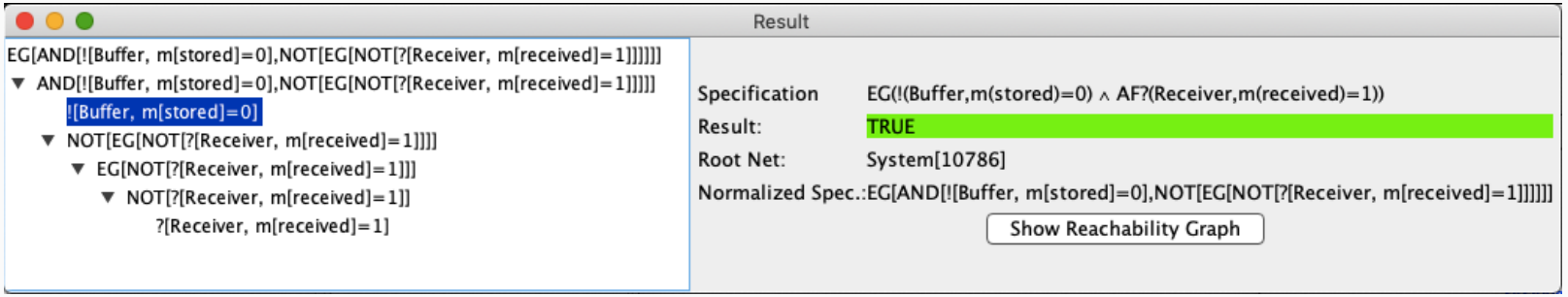
Result: TRUE

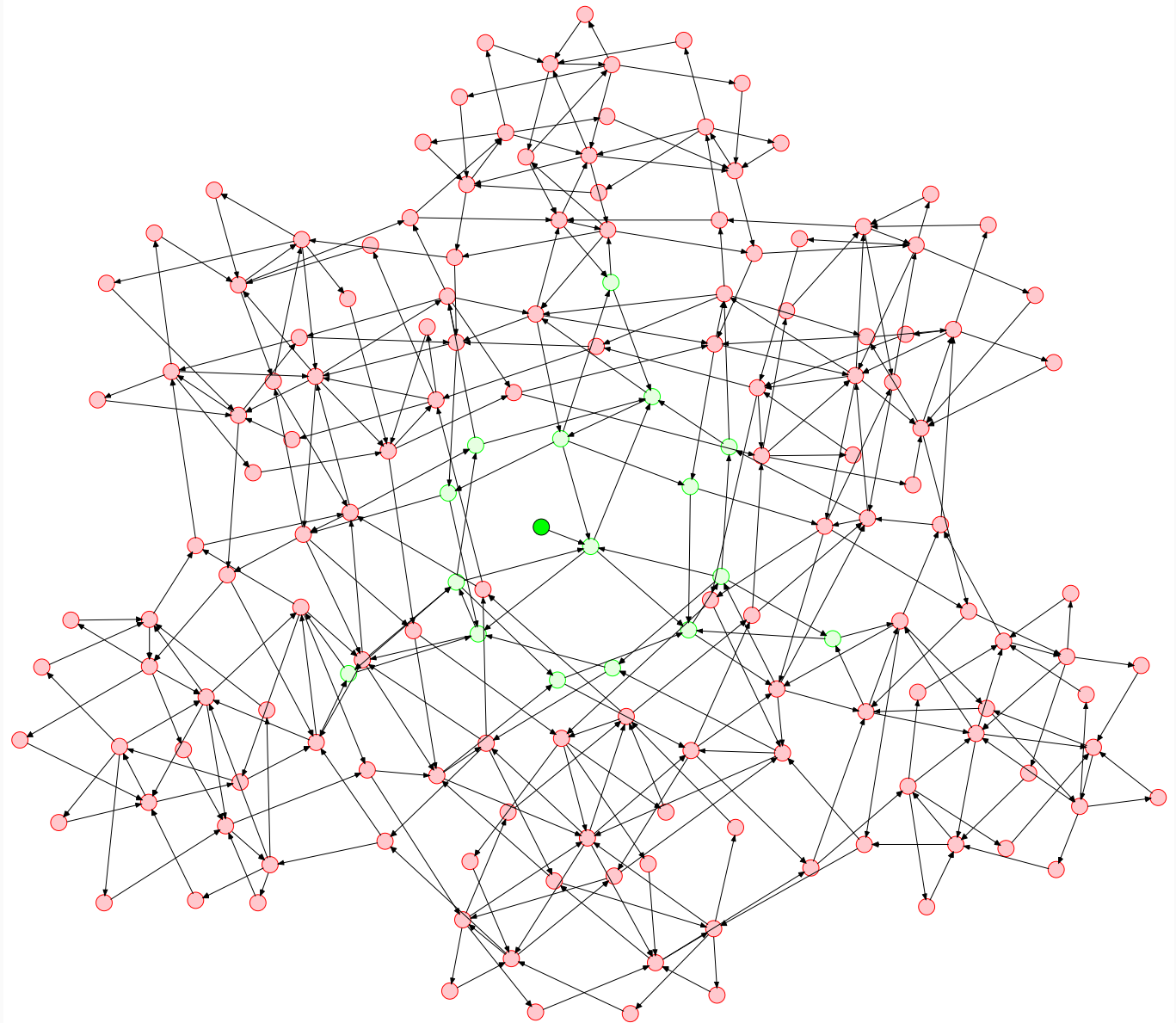
Root Net: System[10786]

Normalized Spec.: EG[AND[![Buffer, m[stored]=0],NOT[EG[NOT[?[Receiver, m[received]=1]]]]]]

Show Reachability Graph




Result
EG[AND[![Buffer, m[stored]=0],NOT[EG[NOT[?[Receiver, m[received]=1]]]]]]
AND[![Buffer, m[stored]=0],NOT[EG[NOT[?[Receiver, m[received]=1]]]]]
![Buffer, m[stored]=0]
NOT[EG[NOT[?[Receiver, m[received]=1]]]]
EG[NOT[?[Receiver, m[received]=1]]]
NOT[?[Receiver, m[received]=1]]
?[Receiver, m[received]=1]
Specification
EG(!(Buffer,m(stored)=0) ∧ AF?(Receiver,m(received)=1))
Result:
TRUE
Root Net:
System[10786]
Normalized Spec.:EG[AND[![Buffer, m[stored]=0],NOT[EG[NOT[?[Receiver, m[received]=1]]]]]]
Show Reachability Graph



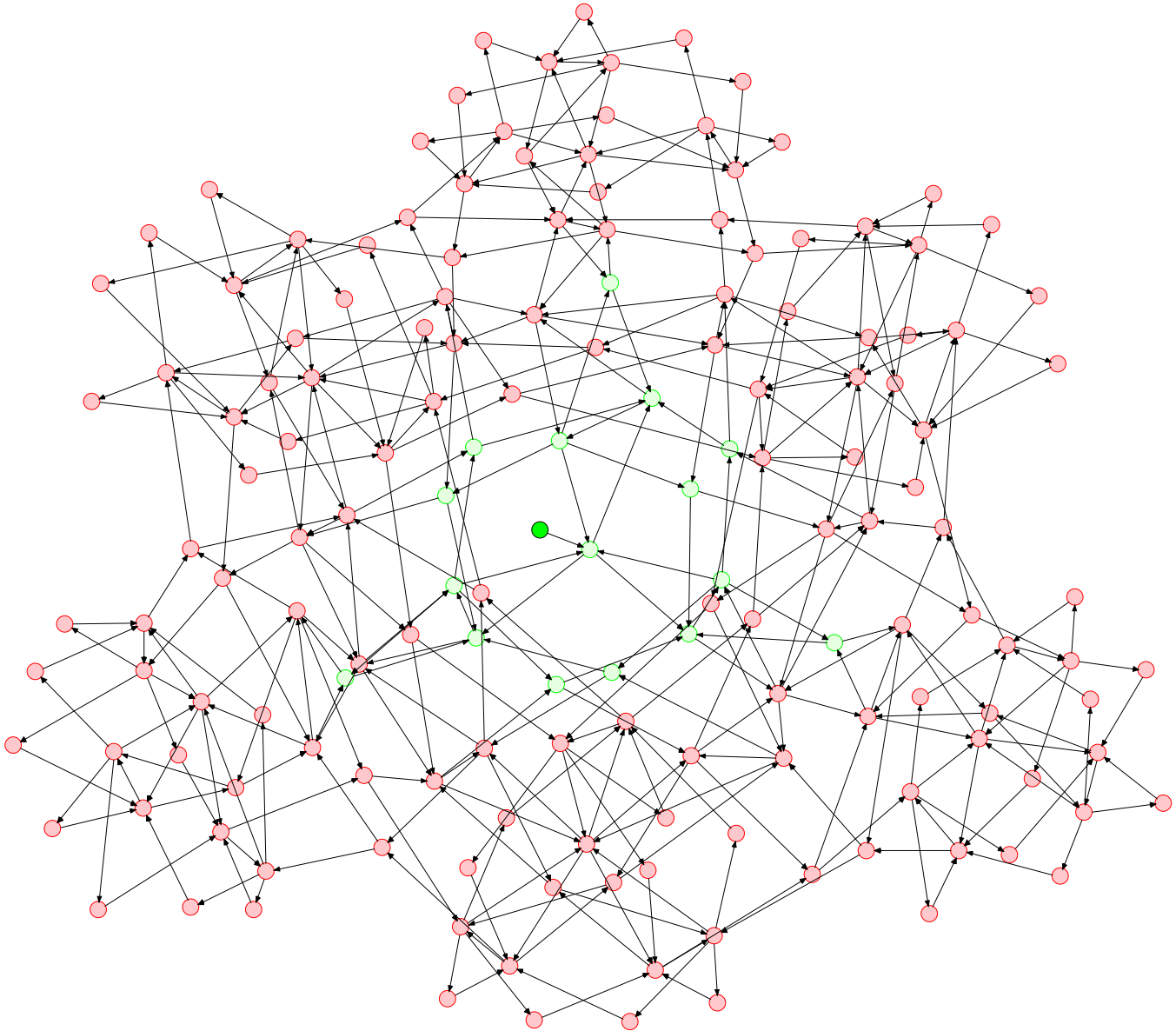
Result

EG[AND[![Buffer, m[stored]=0],NOT[EG[NOT[?[Receiver, m[received]=1]]]]]]

- AND[![Buffer, m[stored]=0],NOT[EG[NOT[?[Receiver, m[received]=1]]]]]
  - ![Buffer, m[stored]=0]
  - NOT[EG[NOT[?[Receiver, m[received]=1]]]]
    - EG[NOT[?[Receiver, m[received]=1]]]
      - NOT[?[Receiver, m[received]=1]]
        - ?[Receiver, m[received]=1]

Specification	EG(!(Buffer,m(stored)=0) ∧ AF?(Receiver,m(received)=1))
Result:	TRUE
Root Net:	System[10786]
Normalized Spec.:	EG[AND[![Buffer, m[stored]=0],NOT[EG[NOT[?[Receiver, m[received]=1]]]]]]

Show Reachability Graph



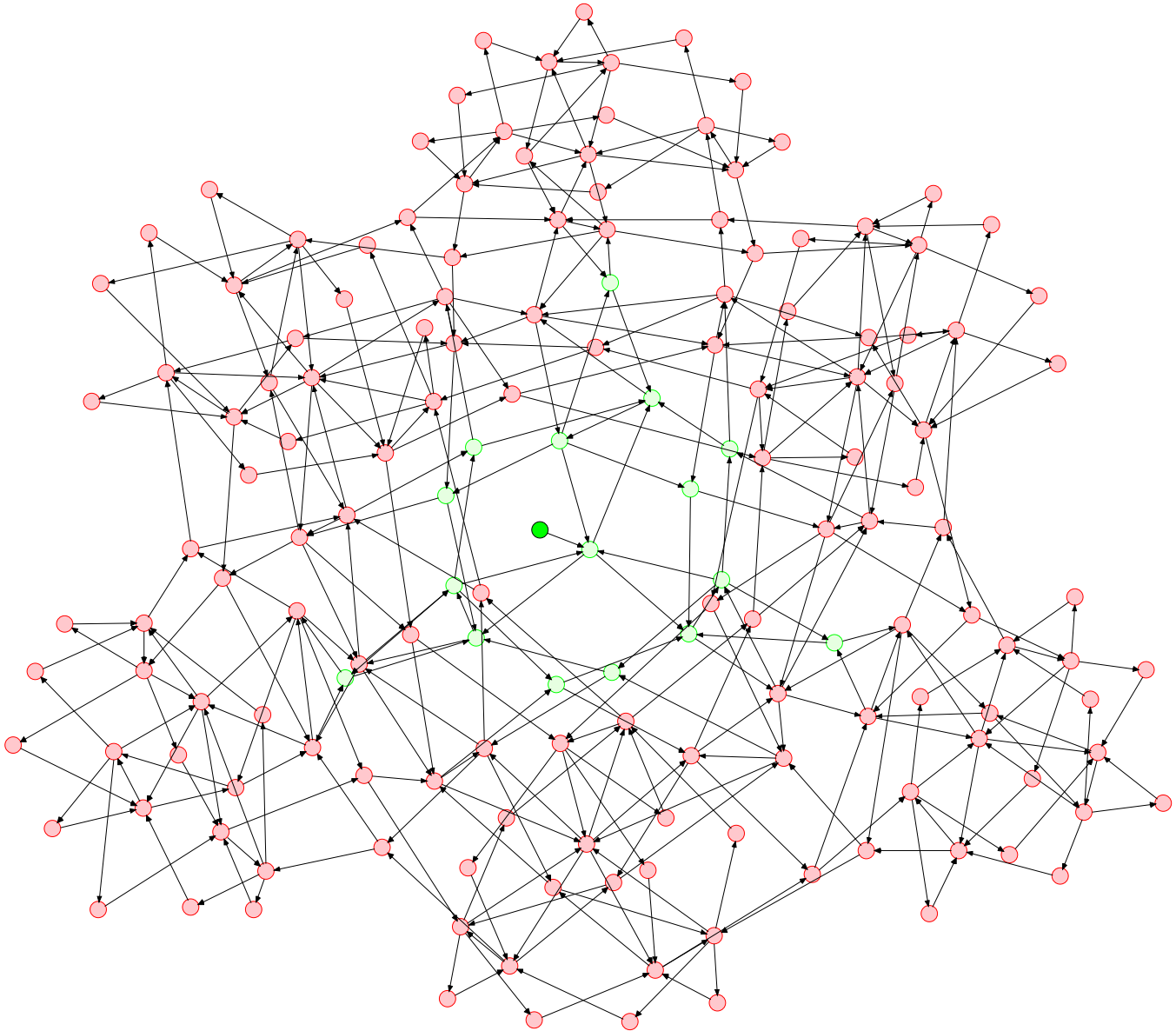
Result

EG[AND[![Buffer, m[stored]=0],NOT[EG[NOT[?[Receiver, m[received]=1]]]]]]

- AND[![Buffer, m[stored]=0],NOT[EG[NOT[?[Receiver, m[received]=1]]]]]
  - ![Buffer, m[stored]=0]
  - NOT[EG[NOT[?[Receiver, m[received]=1]]]]
    - EG[NOT[?[Receiver, m[received]=1]]]
      - NOT[?[Receiver, m[received]=1]]
        - ?[Receiver, m[received]=1]

Specification	EG(!(Buffer,m(stored)=0) ∧ AF?(Receiver,m(received)=1))
Result:	TRUE
Root Net:	System[10786]
Normalized Spec.:	EG[AND[![Buffer, m[stored]=0],NOT[EG[NOT[?[Receiver, m[received]=1]]]]]]

Show Reachability Graph



Evaluation

Evaluation

- Teaching-size problems ($<$10k states) are unproblematic with average computing power
- Colorization is helpful

Conclusion

- CTL Model-Checking of Reference nets
- Modular architecture
- Net Instance Quantifier
- Result visualization

Outlook

Outlook

Teaching-oriented goals:

- LTL-Model Checking
- Coverability graph
- Visualization of large RGs
  $\rightarrow$ Interactive trace visualization

Outlook

Teaching-oriented goals:

- LTL-Model Checking
- Coverability graph
- Visualization of large RGs
  $\rightarrow$ Interactive trace visualization

Efficiency-oriented goals:

- Code-specific improvements
- Transfer of known techniques to Reference nets
- Techniques that exploit the structural information contained in Reference nets