Modeling Reaction Systems

UNIVERSITY OF PISA

Department of Computer Science

Master Degree in Computer Science

Supervisor:
Prof. Roberto Bruni
Prof. Roberta Gori
Prof. Paolo Milazzo

Candidate:
Elvis Rossi

December 2, 2025

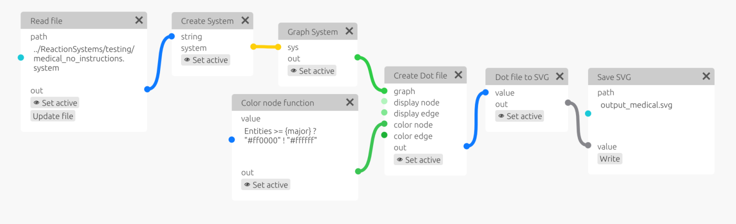
Table of Contents I

Two Rust crates have been developed to model, analyze and design Reaction Systems.

ReactionSystems
ReactionSystemsGUI

ReactionSystemsGUI implements a custom visual language using the libraries `egui` and `egui_node_graph2`.

A web version is also provided that runs locally using WebAssembly.



Read file
path
../ReactionSystems/testing/medical_no_instructions.system
out
Set active
Update file

Create System
string
system
Set active

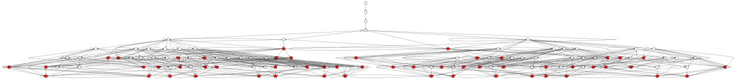
Graph System
sys
out
Set active

Color node function
value
Entities >= {major} ? "#ff0000" ! "#ffffff"
out
Set active

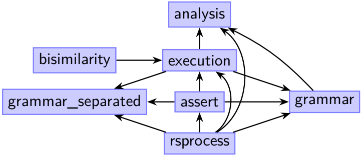
Create Dot file
graph
display node
display edge
color node
color edge
out
Set active

Dot file to SVG
value
out
Set active

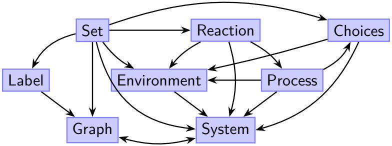
Save SVG
path
output_medical.svg
value
Write



The libraries are more than 30k lines of code, organized in workspaces that allow easy development; the GUI has 29 types and more than 70 node operations.



analysis
bisimilarity
execution
grammar_separated
assert
grammar
rsprocess



Set
Reaction
Choices
Label
Environment
Process
Graph
System

Grammars have been specified and developed for the RS structures using `lalrpop`.

Reaction Systems

Reaction System (RS) is a successful computational framework inspired by biological systems.

Key concepts: *Facilitation* and *Inhibition*

Reaction: (R, I, P)
Reaction System: $\mathcal{A} = (S, A)$

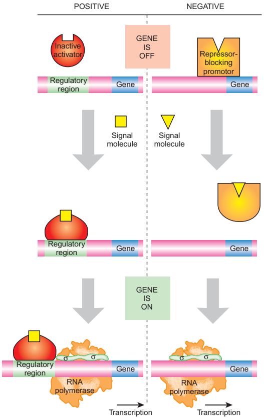


Figure: Principle of Positive and Negative Regulation[1]

Three key properties:

(i) no permanency: entities vanish unless sustained by a reaction;

(ii) no counting: the exact quantity of each entity is irrelevant;

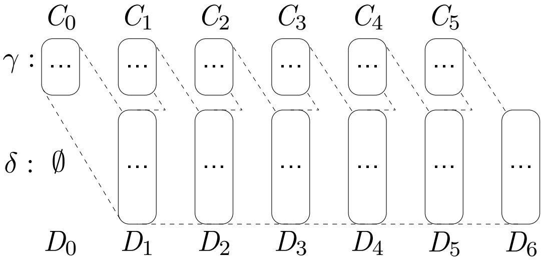
(iii) threshold nature of resources: if an entity is present, it is present for all possible reactions.

Interactive Process

The behavior of a RS is formalized through the notion of an interactive process:

Context Sequence: $\gamma = \{C_i\}_{i \in [0,n]}$,

Result Sequence: $\delta = \{D_i\}_{i \in [0,n]}$ with $D_{i+1} := res_A(D_i \cup C_i)$



SOS rules

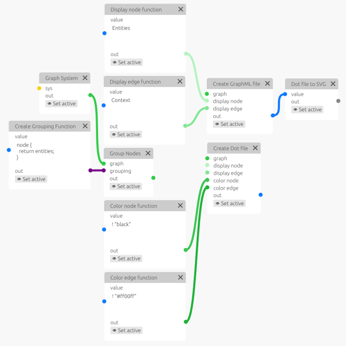
$$\begin{array}{lcl}
P & ::= & [M] \\
M & ::= & (R, I, P) \\
 & | & D \\
 & | & K \\
 & | & M|M \\
K & ::= & 0 \\
 & | & X \\
 & | & C.K \\
 & | & K + K \\
 & | & \texttt{rec}X.K
\end{array}$$

Mutual Exclusion of 2 looping processes:

```
Environment: [
  k1 = ({}.k1 + {act_1}.k1),
  k2 = ({}.k2 + {act_2}.k2)
]
Initial Entities: {out_1, out_2}
Context: [k1, k2]
Reactions: (...)
```

Graphs

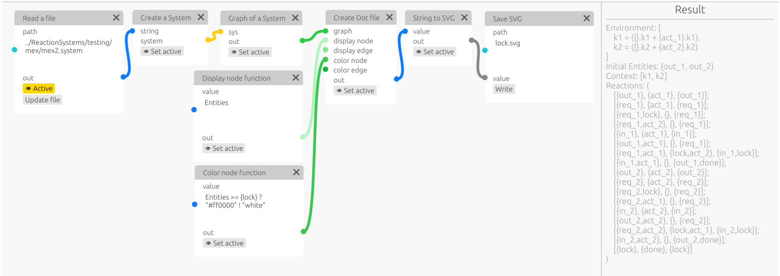
To explore the structure of RSs, methods are provided to create and manipulate graphs.



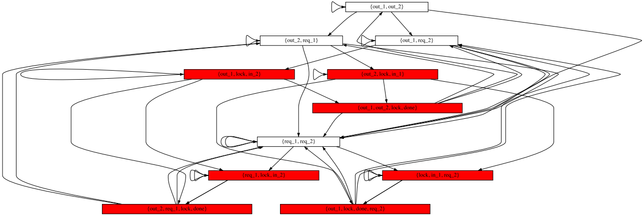
```
node {
  return entities;
}
```

Domain specific languages have been developed for:

- specifying labels of nodes & edges,
- specifying color of nodes & edges,
- grouping of nodes,
- relabeling of edges.



```
Environment: [
   k1 = ({}.k1 + {act_1}.k1),
   k2 = ({}.k2 + {act_2}.k2)
]
Initial Entities: {out_1, out_2}
Context: [k1, k2]
Reactions: (
   [{out_1}, {act_1}, {out_1}];
   [{req_1}, {act_1}, {req_1}];
   [{req_1,lock}, {}, {req_1}];
   [{req_1,act_2}, {}, {req_1}];
   [{in_1}, {act_1}, {in_1}];
   [{out_1,act_1}, {}, {req_1}];
   [{req_1,act_1}, {lock,act_2}, {in_1,lock}];
   [{in_1,act_1}, {}, {out_1,done}];
   [{out_2}, {act_2}, {out_2}];
   [{req_2}, {act_2}, {req_2}];
   [{req_2,lock}, {}, {req_2}];
   [{req_2,act_1}, {}, {req_2}];
   [{in_2}, {act_2}, {in_2}];
   [{out_2,act_2}, {}, {req_2}];
   [{req_2,act_2}, {lock,act_1}, {in_2,lock}];
   [{in_2,act_2}, {}, {out_2,done}];
   [{lock}, {done}, {lock}]
)
```

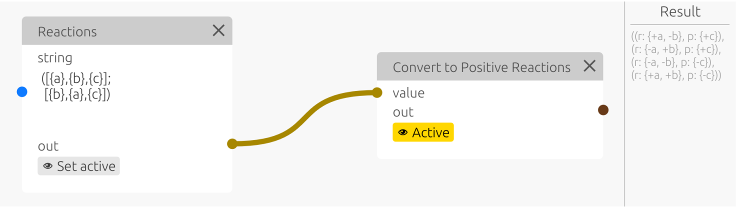


Positive Reaction Systems

Instead of considering the absence of an element, consider the presence of a negative element:

Reaction: (R, P)

An equivalent Positive RS can be built for each RS.

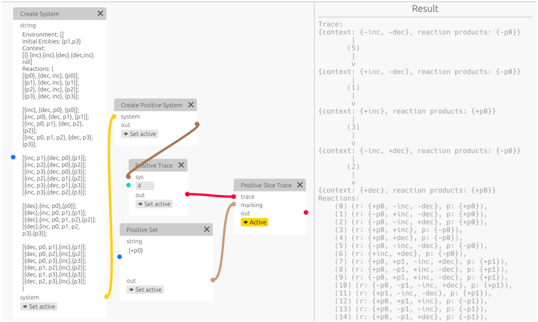


```
([{a},{b},{c}];
 [{b},{a},{c}])
```

```
((r: {+a, -b}, p: {+c}),
(r: {-a, +b}, p: {+c}),
(r: {-a, -b}, p: {-c}),
(r: {+a, +b}, p: {-c}))
```

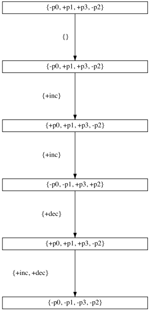
Slicing

Dynamic slicing is a technique that helps a user to debug a program by simplifying a partial execution trace.
The goal is to highlight how a subset of the elements in a state were originated.



```
Environment: []
Initial Entities: {p1,p3}
Context:
[{}.{inc}.{inc}.{dec}.{dec,inc}.
nill]
Reactions: (
[{p0}, {dec, inc}, {p0}];
[{p1}, {dec, inc}, {p1}];
[{p2}, {dec, inc}, {p2}];
[{p3}, {dec, inc}, {p3}];

[{inc}, {dec, p0}, {p0}];
[{inc, p0}, {dec, p1}, {p1}];
[{inc, p0, p1}, {dec, p2},
{p2}];
[{inc, p0, p1, p2}, {dec, p3},
{p3}];

[{inc, p1},{dec, p0},{p1}];
[{inc, p2},{dec, p0},{p2}];
[{inc, p3},{dec, p0},{p3}];
[{inc, p2},{dec, p1},{p2}];
[{inc, p3},{dec, p1},{p3}];
[{inc, p3},{dec, p2},{p3}];

[{dec},{inc, p0},{p0}];
[{dec},{inc, p0, p1},{p1}];
[{dec},{inc, p0, p1, p2},{p2}];
[{dec},{inc, p0, p1, p2,
p3},{p3}];

[{dec, p0, p1},{inc},{p1}];
[{dec, p0, p2},{inc},{p2}];
[{dec, p0, p3},{inc},{p3}];
[{dec, p1, p2},{inc},{p2}];
[{dec, p1, p3},{inc},{p3}];
[{dec, p2, p3},{inc},{p3}];
)
```

```
Trace:
{context: {-inc, -dec}, reaction products: {-p0}}
         |
        (5)
         |
         v
{context: {+inc, -dec}, reaction products: {-p0}}
         |
        (1)
         |
         v
{context: {+inc}, reaction products: {+p0}}
         |
        (3)
         |
         v
{context: {-inc, +dec}, reaction products: {-p0}}
         |
        (2)
         |
         v
{context: {+dec}, reaction products: {+p0}}
Reactions:
    (0) (r: {+p0, -inc, -dec}, p: {+p0}),
    (1) (r: {-p0, +inc, -dec}, p: {+p0}),
    (2) (r: {-p0, -inc, +dec}, p: {+p0}),
    (3) (r: {+p0, +inc}, p: {-p0}),
    (4) (r: {+p0, +dec}, p: {-p0}),
    (5) (r: {-p0, -inc, -dec}, p: {-p0}),
    (6) (r: {+inc, +dec}, p: {-p0}),
    (7) (r: {+p0, +p1, -inc, +dec}, p: {+p1}),
    (8) (r: {+p0, -p1, +inc, -dec}, p: {+p1}),
    (9) (r: {-p0, +p1, +inc, -dec}, p: {+p1}),
    (10) (r: {-p0, -p1, -inc, +dec}, p: {+p1}),
    (11) (r: {+p1, -inc, -dec}, p: {+p1}),
    (12) (r: {+p0, +p1, +inc}, p: {-p1}),
    (13) (r: {+p0, -p1, -inc}, p: {-p1}),
    (14) (r: {+p0, -p1, +dec}, p: {-p1}),
```

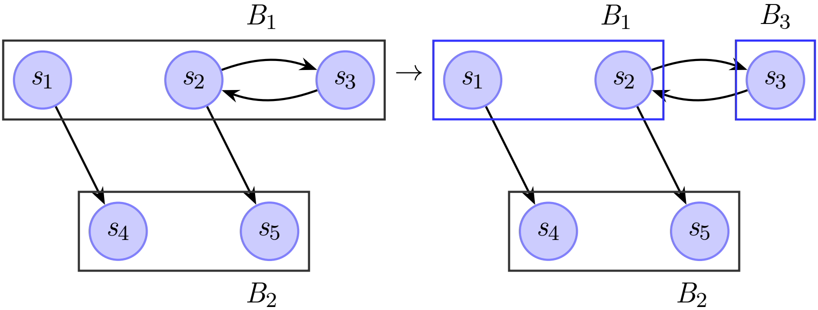


Bisimulation

A common question given two RS processes is if they behave the same. Bisimulation is a binary relation between transition systems defined in therms of coinductive games, of fixed point theory and of logic.
Two algorithms have been implemented: by Kanellakis and Smolka, and by Paige and Tarjan.

Kanellakis and Smolka

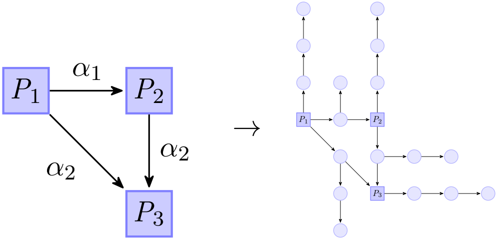
The algorithm by Kanellakis and Smolka is based on the concept of splitter.



Paige and Tarjan

The algorithm by Kanellakis and Smolka has time complexity $O(n \cdot m)$. By reducing to the coarsest stable partition problem, the complexity can be reduced to $O(n \cdot \log(m))$, since three-way splitting can be performed in time proportional to the size of the smaller of the two blocks.

The algorithm is specified over systems with only one action, but other systems can be translated into equivalent ones.



Conclusion

Key contributions:

- New RS Modeling Platform that aids in analysis and design, implemented in Rust. Provided both a CLI and a GUI.
- Comprehensive Feature Set: simulation of RS, bisimulation of graphs, trace slicing, graph generation with Dot, GraphML and SVG outputs, loop analysis, automated conversion between RS types.
- Improved performance and usability compared to previous software written in prolog and python.

References I

[1] David P. Clark, Nanette J. Pazdernik, and Michelle R. McGehee. "Regulation of Transcription in Prokaryotes". In: *Molecular Biology*. 3rd. Academic Press, 2018, pp. 532–542.