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 4, 2025

Reaction Systems

RS is a qualitative model:

- no permanency,
- no counting.

Key concepts: *Facilitation* and *Inhibition*

A reaction (R, I, P) is composed by reactants, inhibitors and products.
Reaction System: $\mathcal{A} = (S, A)$

Behavior is not monotone.

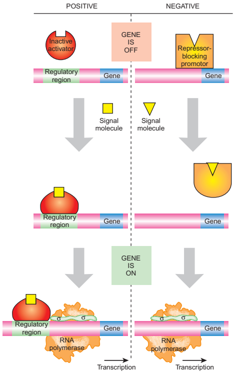


Figure: Principle of Positive and Negative Regulation

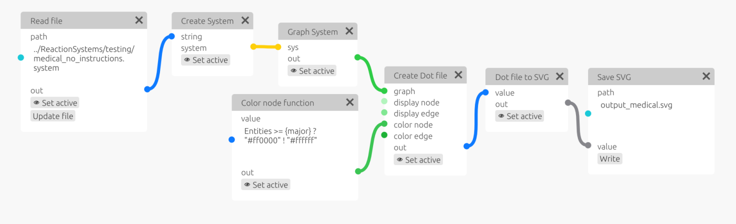
- Modeling and *in silico* experiments,
- Visualizing of LTS
- Verifying properties: reachability, model checking, equivalence.
- Analysis: causality, slicing, attractors, profiling.
- Transformations: positive form, minimization.

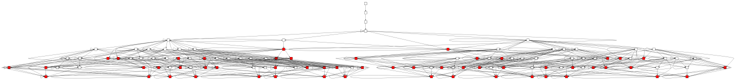
Contribution

The main contribution is a rewrite of previous Prolog code in Rust.

- ReactionSystems: contains the basic structures, parsers and a CLI.
- ReactionSystemsGUI: implements a native and web application with a custom visual language to interact with RS.

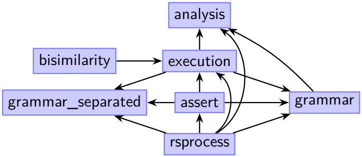
Example

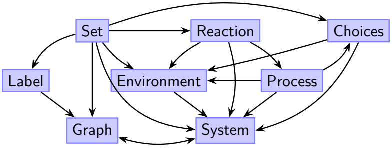

Read file
path
../ReactionSystems/testing/
medical_no_instructions.
system
out
Set active
Update file
Create System
string
system
Set active
Graph System
sys
out
Set active
Color node function
value
Entities >= {major} ?
"#ff0000" ! "#ffffff"
out
Set active
Create Dot file
graph
display node
display edge
color node
color edge
out
Set active
Dot file to SVG
value
out
Set active
Save SVG
path
output_medical.svg
value
Write



Structure

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





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

SOS rules

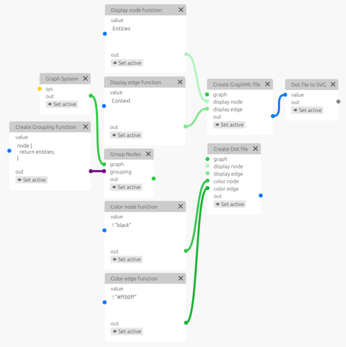
$$
\begin{array}{lcl}
P & ::= & [M] \\
M & ::= & (R, I, P) \\
 & | & D \\
 & | & K \\
 & | & M|M \\
K & ::= & 0 \\
 & | & X \\
 & | & C.K \\
 & | & K + K \\
 & | & \mathtt{rec} X.K
\end{array}
$$

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: (...)
```

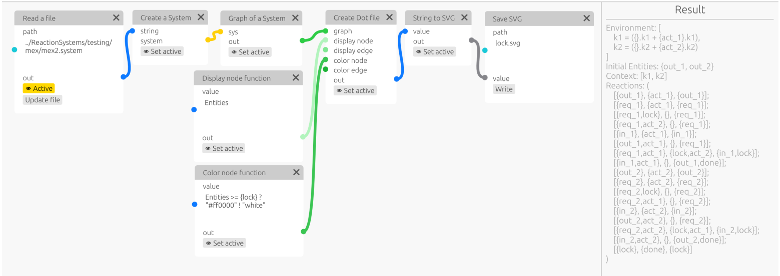
Graphs

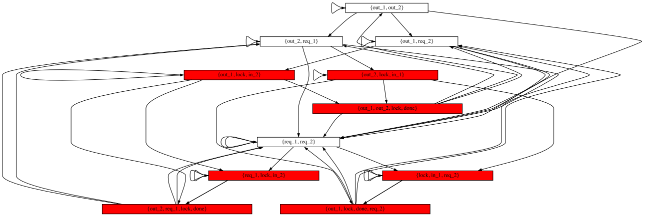
To explore the structure of RSs, methods are provided to create and manipulate graphs.



Domain specific languages have been developed for:

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


Read a file
path
../ReactionSystems/testing/mex/mex2.system
out
Active
Update file
Create a System
string
system
Set active
Graph of a System
sys
out
Set active
Display node function
value
Entities
out
Set active
Color node function
value
Entities >= {lock} ? "#ff0000" ! "white"
out
Set active
Create Dot file
graph
display node
display edge
color node
color edge
out
Set active
String to SVG
value
out
Set active
Save SVG
path
lock.svg
value
Write
Result
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}]
)

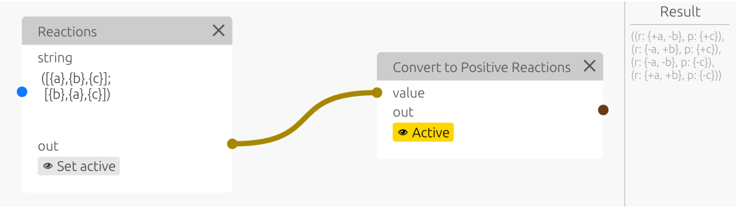

{out_1, out_2}
{out_2, req_1}
{out_1, req_2}
{out_1, lock, in_2}
{out_2, lock, in_1}
{out_1, out_2, lock, done}
{req_1, req_2}
{req_1, lock, in_2}
{lock, in_1, req_2}
{out_2, req_1, lock, done}
{out_1, lock, done, req_2}

Positive Reaction Systems

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

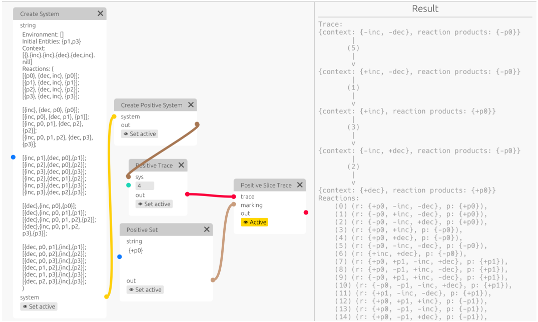
Reaction: (R, P)

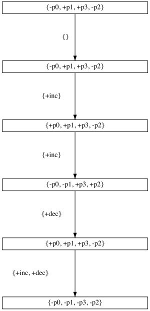
An equivalent Positive RS can be built for each RS.



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.



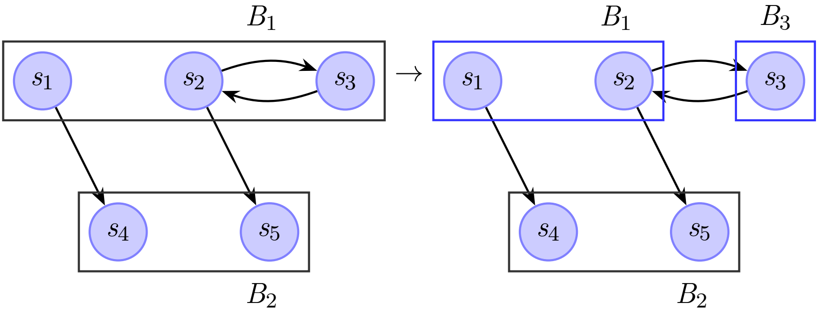


Bisimulation

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

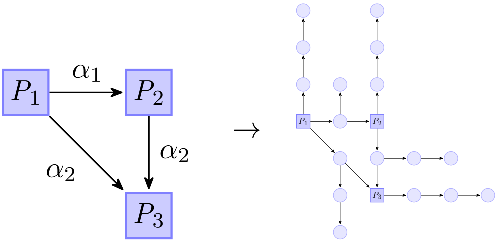
The algorithm by Kanellakis and Smolka is based on the concept of splitter.



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 transformed into equivalent ones.



Testing

During the development to validate the program automated tests and manual integration have been used.

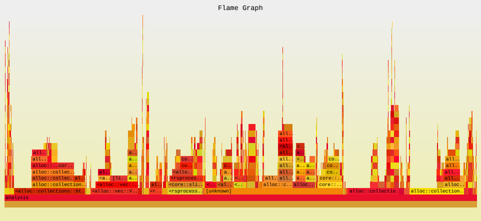


Figure: Profiling using `perf` and `flamegraph`.

Conclusion

Key contributions:

- New RS modeling platform that aids in analysis and design, implemented in 30k lines of Rust. Provides 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.