A Logic Programming approach to Reaction Systems

Giulia Palma

Joint work with Moreno Falaschi

Department of Computer Science, University of Pisa

November 27th, 2020

Structure of the presentation

- We first present the modeling formalism of Reaction Systems (RS) introduced by A. Ehrenfeucht and G. Rozenberg in 2007.
- This formalism is suitable to model both bioinformatic and computer science problems
- Then we present an implementation of the basic formalism in LP (Prolog).
- We show that our implementation is useful as a rapid prototyping tool,
- indeed we present some extensions and their implementation.
  - We check loops over RS computations
  - We define and implement RS with nondeterministic contexts
  - We implement (the recently defined) networks of RS

Reactions

- The functioning of a **biochemical reaction** is based on facilitation and inhibition.
- A **reaction** is a chemical process in which substances act mutually on each other and are changed into different substances, or one substance changes into other substances.

Definition (Reaction)

A **reaction** is a triplet $a = (R, I, P)$, where R, I, P are finite sets.
If S is a set such that $R, I, P \subseteq S$, then a is a reaction in S.

Effects of a reaction

Definition (Result of Reaction)

Let a be a reaction, A a finite set of reactions and T a finite set.

1. a is **enabled by T** if $R_a \subseteq T$ and $I_a \cap T = \emptyset$ (indicated by $en_a(T)$);
2. The **result of a on T** is defined by:

$$res_a(T) = \begin{cases} P_a & \text{if } en_a(T) \\ \emptyset & \text{otherwise} \end{cases}$$

3. The **results of A on T** is defined by $res_A(T) = \cup_{a \in A} res_a(T)$.

Reaction Systems

Definition (Reaction Systems)

A **reaction system**, abbreviated *rs*, is an ordered pair $\mathcal{A} = (S, A)$ such that S is a finite set, and $A \subseteq rac(S)$.

S = background set of $\mathcal{A}$.

Its elements are called **entities**, they represent molecular entities (e. g. atoms, ions, molecules) that may be present in the state of a biochemical system modeled by $\mathcal{A}$.

A = set of reactions of $\mathcal{A}$.

Since S is finite, so is A.

All the notations introduced for sets of reactions carry over to reaction systems: $T \subseteq S, en_{\mathcal{A}}(T) = en_A(T); res_{\mathcal{A}}(T) = res_A(T)$;
T is active in $\mathcal{A}$, if $en_{\mathcal{A}}(T) \neq \emptyset$.

Reaction systems: basic assumptions

The theory of Reaction Systems is based on the following assumptions:

1. **No permanency.** An entity of a set T vanishes unless it is sustained by a reaction. This reflects the fact that a living cell would die for lack of energy, without chemical reactions.
2. **No counting.** The basic model of reaction systems is very abstract and qualitative, i.e. the quantity of entities that are present in a cell is not taken into account. The number of reagents does not count in the reaction systems model, unlike the stoichiometric equations, in which the quantities are fundamental.
3. **Threshold nature of resources.** Either an entity is available and there is enough of it (i.e. there are no conflicts), or it is not available at all.

Interactive Processes

The dynamic behaviour of reaction systems is captured through the notion of interactive process:

Definition (Interactive Process)

Let $\mathcal{A} = (S, A)$ be a *reaction system*. An **interactive process** in $\mathcal{A}$ is a pair $\pi = (\gamma, \delta)$ of finite sequences such that:

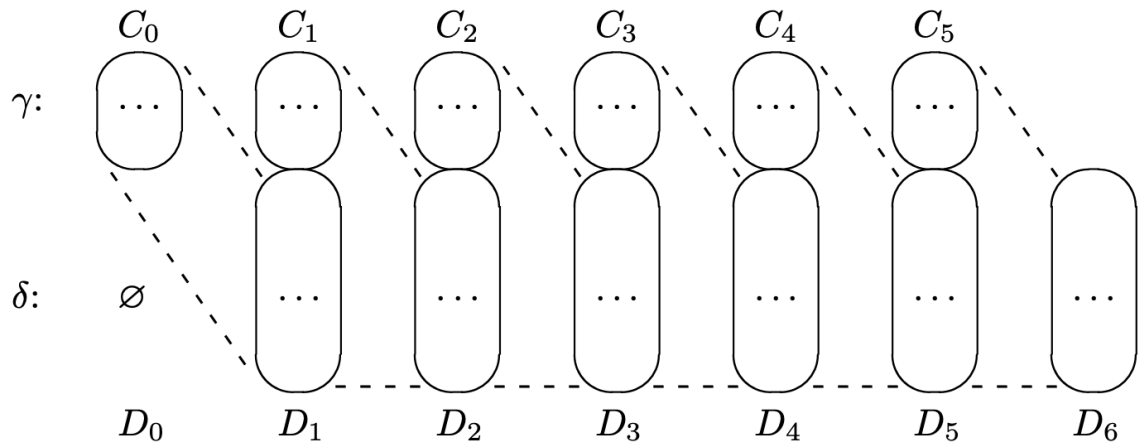
$\gamma = C_0, C_1, \ldots, C_n, \quad \delta = D_1, \ldots, D_n, \; n \geq 1, \quad$ where

$C_0, \ldots, C_n, D_1, \ldots, D_n \subseteq S, \quad D_1 = res_{\mathcal{A}}(C_0), \quad$ and

$D_i = res_{\mathcal{A}}(D_{i-1} \cup C_{i-1})$ for each $2 \leq i \leq n$.

Interactive Processes

- The sequence $C_0, \ldots, C_n$ is the **context sequence of** π, $con(\pi)$
- The sequence $D_1, \ldots, D_n$ is the **result sequence of** π, $res(\pi)$
- C_i and D_i do not have to be disjoint.
- The notion of an interactive process is visualized in the following figure:



An Interpreter of RS in logic programming

- Now we briefly describe a prototypical implementation of the Reaction Systems framework in a logic programming language (Prolog).
- Our implementation is available on-line at `https://www3.diism.unisi.it/~falaschi/ReactionSystems`, together with a small manual to use it.
- In the paper, you can also find examples concerning a nondeterministic context and interaction between components of a biological system.

An Interpreter of RS in logic programming

- Sets are represented by corresponding lists of values. The background set S of a reaction system is represented by a list of distinct constant symbols.
- This formalism is suitable to model both bioinformatic and computer science problems.
- A reaction (R, I, P) is represented by a triple of lists, where R is the list of reactants, I is the list of inhibitors and P is the list of products.
- The set of reactions in a reaction system is defined by a list of reactions and is introduced by using the predicate reactionSet/1.

A computation with the interpreter of RS

- For evaluating a query to our interpreter, the predicate to be called is computation(InitialState, ListOfStates).
- The input argument InitialState is the list of the entities (reagents) in the initial state.
- The second (output) argument ListOfStates is the list of states computed in the reaction system by our interpreter.

A computation with the interpreter of RS

- `computation/2` starts by making some preliminary checks (predicate `preliminaryCheck/1`) to verify the basic assumptions on RS.
- Then, the user will get a choice:
  1. to make a context independent computation or a computation which interacts with the context.
  2. to make a computation with a limited maximal number of steps, or a computation with a possibly unlimited length.

A computation with the interpreter of RS

A **single step of the computation** is performed as follows.

- The result of a single reaction (without the context) is computed by the predicate result(T, R, I, P, P1).
- Given the state `T` and the reaction `(R, I, P)`, the result `P1` will be `P` if the reaction is enabled in `T` (that is, if the predicate `enable` is true), otherwise it will return the empty set.
- The predicate enable(R, I, T) checks if the reaction with reactants `R` and inhibitors `I` is enabled in the state T.
- Then the result of all reactions on T is computed by the predicate resultallreactions(T, ReactionSet, T1), which recursively calls `result/5` and collects the union of its outcomes.

A computation with the interpreter of RS

- To consider the context, we need the unit fact for the predicate context/1.
- The input argument of context/1 is a list of context reagent lists. The list in position k is the context to be added at step k of the computation.

Stopping unlimited computations with memoization

- A computation in a RS may enter in a loop created either directly in one reaction or with dependencies between different reactions.
- We have extended our interpreter by using a technique in the style of '**memoization**'.
- Predicate unlimitedComputationContextIndependentMemoized/2 keeps track of the states of the computation, and when a state is repeated, the computation is stopped and returned.

Interaction of two RS

- It is possible to define two interacting reaction systems, in such a way that the entities that usually come from the context of one reaction system will be provided instead from the other reaction system.
- A slight modification of our program allows us to model this behaviour.
- Two separate unit predicates for the reactions in the two RS, reactionsetF/1 and reactionsetS/1.
- Then, we modify computation/2: the context for the first reaction system is given by context/1;
- for the second reaction system the context is the output computed by the first reaction system.

Networks of RS

- An extension of our interpreter for modeling **networks of reaction systems**.
- The **context** for a reaction system originates from a network of RS: a graph where nodes represent RS, and where each RS contributes to defining the context of all its neighbours.
- The interaction of two reaction systems can be seen as a special case of this definition.

(Simulation of) Networks of RS

- The RS (nodes) of the network are numbered by distinct positive integers 1, 2, 3, and so on.
- The computation proceeds for a limited number of steps K, where number K is provided by the user, or until an empty state.
- The complete computation for RS 1 is reported.

Networks of RS

- We present an example of a network of two RS (nodes), but the program is valid for an arbitrary finite number of nodes.
- As output we obtain the list of final states of all the RS in the network, and the complete computation of the reaction system 1.
- Query: call main(F, D). F and D are (computed) output argument. F is the list of network final states (one state for each node RS of the network) and D is the complete derivation for reaction system 1.

Networks of RS

- The edges of the network are represented by a predicate network/1 introducing a list of pairs of the form [m,n] meaning that there is an arc from node m to node n.
- The predicate search(N,Net,S0,S1) looks for all pairs [N1,N] in Net and returns S1 = union of the states in position N1 of S0, and computes the context for N in the network of RS.
- The predicate InitialStates/1 is defined by a unit fact which introduces a list of the initial states of the nodes in the network.
- So state in position k corresponds to the initial state of node k.

Conclusions and future work

- We have presented the first implementation in Prolog of the framework of RS.
- Our interpreter is flexible and suitable for rapid prototyping and implementing extensions.
- Users can choose to make indefinitely long computations or computations limited to a maximum of k steps.
- We have introduced a kind of memoization mechanism based on accumulators for stopping a looping computation when a state gets repeated.
- **Extensions**: we have implemented a network of RSs, and (don't care) nondeterministic contexts.
- **Future work**: implementation more efficient and user friendly, by using CLP and graphical representations of the computations. Analysis tools for RS.

Thanks for your attention

and

Happy Birthday, Maurizio!