

Instructions to use the BioHML Prolog interpreter of Moreno Falaschi, Linda Brodo and Roberto Bruni.

The interpreter is written in SWI-Prolog, but it has been tested also under GNU Prolog. It extends to verification of BioHML formulas and biosimilarity a previous version defined by Moreno Falaschi and Giulia Palma for the execution of basic Reaction Systems.

A Reaction System is a term `sys(Delta, E, Ks, Rs)`, where:

1. `Delta` is the environment,
2. `E` the current set of entities,
3. `Ks` the list of context processes, and
4. `Rs` the list of reactions.

The syntax of each argument is explained later.

For the underlying theory please see the paper:

“SOS rules for Equivalences of Reaction Systems”, by L. Brodo, R. Bruni and M. Falaschi.

Installation and usage instructions:

1. Download the program file and save it into a local folder.
2. Edit the file to define the Reaction System you are interested in (see below).
3. Launch SWI-Prolog and consult the file.
4. Query the interpreter using the predicate `main/1`.

For using this program you have to modify the last lines of the source code, where the following predicates are defined:

- `myentities/1`, `myreactions/1`, `mycontext/1`, `myenvironment/1`:
they are used to define the main Reaction System RS of interest.

- `mybhml/1`:

it is used to define a BioHML formula `G` and to check if it is satisfied by the main Reaction System RS.

The query `main(X)` . either returns `ok` (if the BioHML formula `G` is satisfied by RS), or the reason why `G` is not satisfied.

1) Normally, the predicate `myentities/1` defines the empty list of reagents (as the context sequence will provide other reagents, including the initial ones). However, the predicate can be updated to define a (non empty) list of reagents, like in

```
myentities([a1,...,an]).
```

from which the computation of the reaction system will start.

2) the predicate `myreactions/1` defines the list of reactions of the Reaction System. Its unique argument must be a list of terms `react(R, I, P)`, like in

```
myreactions([react(R1,I1,P1),...,react(Rn,In,Pn)]).
```

Each term `react(Ri, Ii, Pi)` contains a list of reagents `Ri`, a list of inhibitors `Ii` and a list of products `Pi`. Single reagents, inhibitors and products are just constant symbols.

3) the predicate `mycontext/1` defines the list of contexts. The Prolog Herbrand syntax for contexts is the following:

```
K ::= nil | rec(X) | pre(C,K) | plus(K1,K2)
```

Where `nil` stops the computation, `rec(X)` invokes the declaration of the constant `x` from the environment, `pre(C,K)` makes available the reagents `C` in the current step and then behaves as `K` at the next step, `plus(K1,K2)` behaves as either `K1` or `K2`.

For example, you can set:

```
mycontext([plus(pre([a,b],rec(x)),pre([a],pre([a],nil))))).
```

4) the predicate `myenvironment/1` takes a list of constant declarations `def(X,K)`.

For example, you can set:

```
mycontext([def(x,pre([a],rec(x))))).
```

5) the predicate `mybhtml/1` takes a BioHML formula `G`, defined over some assertion `F`. The BioHML formulas abstract syntax is expressed by Prolog Herbrand terms as follows:

```
G ::= true|false|and(G1,G2)|or(G1,G2)|diamond(F,G)|box(F,G)
F ::= sub(C,N)|nonempty(N)|and(F1,F2)|or(F1,F2)|xor(F1,F2)|not(F)
```

where `N` is a natural number between 1 and 4, which identifies the corresponding set in a transition label `obs(E,R,I,P)`. So number 1 is list `E`, number 2 is `R`, and so on.

For example, you can set:

```
mybhtml([box(not(sub([c],1)),diamond(not(sub([c],1)),true))]).
```