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

The online interpreter 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. The source code of the online interpreter has been adapted for compatibility issues with Tau-Prolog.

A Reaction System is a term `sys(Delta,E,Ks,Rs)`, where `Delta` is the environment, `E` the current set of entities, `Ks` the list of context processes and `Rs` the list of reactions (their syntax 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.

To customize the program you have to modify the following predicates:

- `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.

- `myassert/1, advenvironment/1, adventities/1, advreactions/1, advcontext/1`: they are used to define an assertion F and an adversary Reaction System ARS and to check if RS and ARS are F-biosimilar.

On the web page, the code is divided in two parts: in the upper textarea you find all main predicates; in the lower textarea you find the code that you must edit to experiment with custom Reaction Systems. You can press the following buttons:

LTS: the query `main(digraph,X).` is invoked, which returns a graph description in .dot format representing (a summary of) the Labelled Transition System of the Reaction System RS. The answer if automatically copied in the bottom left panel. By pressing the button **Draw** the graph is drawn in the bottom right panel.
The graph visualization tool is based on the library Vis.js. The nodes of the graph show the current set of entities and the context processes (omitting all sorts of brackets), the labels of transitions show just the available and produced entities. To change the graph, you can edit the graph description and click the button **Draw**.

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

biosim: the query `main(biosim,X).` is invoked, which either returns a F-biosimulation relation proving that RS and ARS are F-biosimilar or `false`.

Adv: the query `main(adv,X).` is invoked, which returns the LTS of ARS.

In all the above cases the result is shown in the **Result** section of the page.

How to customize the queries:

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`.
By using the predicate `parse_ctx/2`, a more convenient syntax can be used in the defining clause
```
K ::= nil  | x  | C.K  |  (K1 + … + Kn)
C ::= {a1,…,an}
```
For example the following two clauses are equivalent:
```
mycontext([plus(pre([a,b],rec(x)),pre([a],pre([a],nil)))]).
mycontext(Ks):- parse_ctx('[({a,b}.x + {a}.{a}.nil)]',Ks).
```

4) the predicate `myenvironment/1` takes a list of constant declarations `def(X,K)`. By using the predicate `parse_env/2`, the more convenient syntax `X=K` can be used.
For example the following two clauses are equivalent:
```
mycontext([def(x,pre([a],rec(x)))]).
mycontext(Ks):- parse_ctx('[x={a}.x)]',Ks).
```

5) the predicate `mybhml/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.
By using the predicate `parse_bhml/2`, and `parse_assert/2` a more convenient syntax can be used in the defining clauses
```
G ::= true  |  false  |  (G1  &  …  & Gn)  |  (G1  *  …  *  Gn)  |  <F>G  |  [F]G
F ::= C inE  |  C inR  |  C inI  |  C inP
    |  ? inE  |  ? inR  |  ? inI  |  ? inP
    |  (F1  &  …  & Fn)  |  (F1  *  …  *  Fn)  |  (F1^F2)  |  -F
```
For example the following two clauses are equivalent:
```
mybhml([box(not(sub([c],1)),diamond(not(sub([c],1)),true))]).
mybhml(G):-parse_bhml('[-{c} inE]<-{c} inE>true',G).
```

6) the predicate `myassert/1` takes an assertion `F`, as defined above.

Examples:

In the customizable textarea there is an already included example:

```
/* a Reaction System Process */
myenvironment([]).
myentities([]).
myreactions([react([a,b],[c],[b])]).
mycontext(Ks):-parse_ctx('[({a,b}.{a}.{a,c}.nil + {a,b}.{a}.{a}.nil)]',Ks).

/* a BioHML formula to check */
mybhml(G) :- parse_bhml('<-{c} inE>[-{c} inE]<-{c} inE>true',G).

/* a F-biosimilarity check against an adversary process*/
myassert(F) :- parse_assert('-{c} inE',F).
advenvironment([]).
adventities([]).
advreactions([react([a,b],[c],[b])]).
advcontext(Ks) :- parse_ctx('[{a,b}.{a}.{a,c}.nil]',Ks).
```

Feel free to change it, or you can choose to load in the textarea one of the other predefined examples just clicking on their names.