Known limitations

for {log} version 4.6.14

Interactive environment

-

Language features (general)

- In RUQs and in Intensional Set terms the control term must be a variable (not any term). For example,

  ```
  S = {[X,Y]: X in (1) & Y in {2}}
  ```

  is not allowed.

- Evaluable compound integer expressions can occur only in the built-in binary predicates `is`, `=<`, `<`, `>=`, `>`, `=:=`, `=\=`, like in Prolog. However, differently from Prolog, they can contain uninstantiated variables. For example,

  ```
      X in int(1,10) & X is Y + 1
  ```

  is allowed, while

  ```
   - X in int(1,10) & f(X) = f(Y + 1)  → no
   - p(X,X +1) :- q(X)
   - (2 + 1) in int(1,4)  → no
   - integer(1+2)  → no
  ```

  are not allowed.

- (Mixing set and interval terms) Set terms of the form `{t1,...,tn\int(a,b)}` are not allowed in user programs and goals. However, they are dealt with correctly within the interpreter and they can be returned as the result of a set constraint. For example:

  ```
  log}=> X in {0\int(1,10)}.
  wrong set term

  {log}=> un(int(1,2),int(3,4),R).
  R = {1,2\int(3,4)}
  ```

Constraint solver

- Interval bounds must be instantiated to integer constants (no uninstantiated variables nor compound integer expressions are allowed). For example:

  ```
  {log}=> X in int(A,6).
  INSTANTIATION ERROR: interval bounds must be known values

  {log}=> X in int(1+2,6).
  no

  {log}=> 1 in int(0,B) & B = 3.
  INSTANTIATION ERROR: interval bounds must be known values

  {log}=> B = 3 & 1 in int(0,B).
  B = 3
  ```

- (Constraints over lists) The constraint solver is not able to prove that the following constraint involving lists is unsatisfiable:

  ```
  {log}=> X in L & L in X & list(L).
  true
  ```

```
Constraint: X in L, L in X, list(X), list(L)
```

- (FD solver incompleteness) {log} uses an incomplete FD solver, hence it can be not able to detect unsatisfiability of some constraints involving integer variables. Providing a finite domain for the integer variables possibly occurring in the constraint guarantees completeness of the solver. For example:

```
{log}=> X > Y & Y < X.
true
Constraint: integer(Y), integer(X)

{log}=> un(X,Y,Z) & size(X,NX) & size(Z,NZ) & NZ<NX.
true
Constraint: un(X, Y, Z), size(Y, _G7609), _G7609 in int(0, sup), size(X, NX),
size(Z, NZ), set(Y), set(X), NX in int(3, sup), set(Z), NZ in int(2, sup)
```

Adding a finite domain for X and NX, respectively, we get the (desired) failure (also with automatic labeling disabled)

```
{log}=> X > Y & X < Y & X in int(1,10).
no

{log}=> un(X,Y,Z) & size(X,NX) & size(Z,NZ) & NZ<NX & NX in int(0,10).
no
```