RtToy

Here we present two variants of Toy for experimenting with mixing call-time choice and run-time choice parameter passing mechanisms. One of them starts with the regular call-time choice framework of Toy and provides a primitive to express run-time choice [1]. In the other the default call-time choice mechanism of Toy has been substituted to provide run-time choice by default and a primitive `where` is used to define local bindings to express sharing of values and therefore call-time choice [2].

Error: Macro TOC(None) failed

```
'NoneType' object has no attribute 'endswith'
```

Call-time choice based variant

The main ideas behind this prototype can be found in [1].

Download

A first prototype can be found here

Basic usage

To use the prototype:

0. Unzip the rile `RtToy.zip`

    This will create a directory *rttoy* with many files and some folders.

1. Start a Sicstus Prolog session (3.12 or earlier).

2. Change the Sicstus working directory to the created *rttoy*. This can be done by invoking:

```
?- use_module(library(system)),working_directory(Old,<write here the path to 'rttoy'>).
```

For instance:

```
?- use_module(library(system)),working_directory(Old,'f:/systems/rttoy').
```

3. Invoke

```
?- compile(rttoy).
```

This starts a Toy session.

4. You can move to the folder with examples by executing the Toy command

```
Toy> /cd(bancopruebas)
```

5. You can compile the program ej-runtime.toy by executing the Toy command

```
/run(ej-runtime)
```

6. You can solve goals. In particular, to evaluate one expression *e*, you execute:

```
Toy> e==X
```

and this will return the value(s) of *e* as binding(s) for *X*

Example: Consider the program *ej-runtime.toy* included with the prototype

```
/*********************************************************************
Some examples to test RTToy, an extension of Toy to support combination
of call-time choice and run-time choice
*********************************************************************/

/*********** First, some general functions **********/

infixr 5 //
X // Y = X
X // Y = Y

infixr 6 ++
[] ++ Ys = Ys
[X|Xs] ++ Ys = [X|Xs++Ys]

reverse [] = []
reverse [X|Xs] = reverse Xs ++ [X]

take N [] = []
take N [X|Xs] = if N<=0 then [] else [X|take (N-1) Xs]

/*********** Example 1: string generation by grammar rules **********/

letter = "a" // "b" // "c"

star X = [] // X ++ star X

word = star (rt letter) % or equivalently, rt (star letter)

palindrome = palaux word

palaux X = X++([]//letter)++reverse X

/*********** Example 2: variations upon coin **********/

coin = 0
coin = 1

double X = X+X

f X = g X coin
g X Y = (X,X,Y,Y)

a = double (rt coin)  % or equivalently, rt (double coin)
b = double coin
c = f (rt coin)  % or equivalently, rt (f coin)
a' = rt (double coin)

/******* Example 3: numbers and repetitions ********/

repeat X = [X|repeat X]

numberCt N = take N (repeat (0//1//2))
numberRt N = take N (repeat (rt (0//1//2)))
```

Some goal examples with this program follow:

```
Toy> a == L
      { L -> 0 }
      Elapsed time: 0 ms.
sol.1, more solutions (y/n/d/a) [y]?
      { L -> 1 }
      Elapsed time: 0 ms.
sol.2, more solutions (y/n/d/a) [y]?
      { L -> 1 }
      Elapsed time: 0 ms.
sol.3, more solutions (y/n/d/a) [y]?
      { L -> 2 }
      Elapsed time: 0 ms.
sol.4, more solutions (y/n/d/a) [y]?
      no
      Elapsed time: 0 ms.
```

```
Toy> c == X
      { X -> (0, 0, 0, 0) }
sol.1, more solutions (y/n/d/a) [y]?
      { X -> (0, 0, 1, 1) }
sol.2, more solutions (y/n/d/a) [y]?
      { X -> (0, 1, 0, 0) }
sol.3, more solutions (y/n/d/a) [y]?

      { X -> (0, 1, 1, 1) }
sol.4, more solutions (y/n/d/a) [y]?
      { X -> (1, 0, 0, 0) }
sol.5, more solutions (y/n/d/a) [y]?
      { X -> (1, 0, 1, 1) }
 sol.6, more solutions (y/n/d/a) [y]?
      { X -> (1, 1, 0, 0) }
sol.7, more solutions (y/n/d/a) [y]?
      { X -> (1, 1, 1, 1) }
```

7. The use of *rt* annotations is also allowed in goals. For instance:

```
Toy> rt (double coin) == L
      { L -> 0 }
sol.1, more solutions (y/n/d/a) [y]?
      { L -> 1 }
sol.2, more solutions (y/n/d/a) [y]?
      { L -> 1 }
sol.3, more solutions (y/n/d/a) [y]?
      { L -> 2 }
sol.4, more solutions (y/n/d/a) [y]?
      no
```

Run-time choice based variant

The main ideas behind this prototype can be found in [2].

- Excepting the use of sharing bindings (see below) the programs behave like term rewriting systems. In other words, the default behaviour is run-time choice.
- The express that the evaluation of a certain expression is subject to sharing explicit bindings are used, written using the `where` construction. The reason to use `where` instead of `let`, as it is done in [2] has been that it makes the implementation easier, as the `where` construction was already present in the syntax of Toy.

Download

A first prototype can be found here

Basic usage

1. Open a Sicstus Prolog session (the Sisctus version must be previous to 4)

2. Execute
   ```
   ?- use_module(library(system)),working_directory(Old,<rttoy directory>).
   ```
   For example
   ```
   use_module(library(system)),working_directory(Old,'f:/0.trabajo/0.juanrhortala/svn/wlpe2008-1/rttoypep
   ```
3. Execute
   ```
   ?- compile(rttoy).
   ```
4. Now, inside the Toy session, execute
   ```
   /prolog(assert(option(rt_all)))
   ```
   As a result now we have a run-time choice behaviour by default.
5. Call-time choice behaviour can be achieved by using sharing bindings by means of the `where` syntax. For example:
   ```
   double X = Y+Y where Y=X.
   ```
   In [2] can be found a precise formal specification of the bindings which must be introduced in a program in order to get call-time choice behaviour for the whole program.

6. To find some example: inside the Toy session, move to:
   ```
   /cd(bancopruebas)
   /run(ej-runtime-pepm)
   ```

Other resources

References

1. ^ F.J. López-Fraguas, J. Rodríguez-Hortalá and J. Sánchez-Hernández , A Lightweight Combination of Semantics for Non-deterministic Functions, In Proc. WLPE'08, 2008, http://gpd.sip.ucm.es/fraguas/papers/WLPE08.pdf
2. ^ F.J. López-Fraguas, J. Rodríguez-Hortalá and J. Sánchez-Hernández , A Flexible Framework for Programming with Non-deterministic Functions, In Proc. PEPM'09, ACM Press, 2009, http://gpd.sip.ucm.es/fraguas/papers/PEPM09.pdf

Contact

This sofware is developed by Francisco Javier López Fraguas, Juan Rodríguez Hortalá and Jaime Sánchez Hernández, at the Department of Computer Systems and Computing, Universidad Complutense de Madrid, Spain.