Safe opaque patterns in Toy

This page contains a branch of the Toy 2.3.1 system that detects problematic uses of opaque patterns. The system is the same as Toy 2.3.1 but incorporates the type system presented in [1] instead of the classical Damas-Milner type system. Although this branch of Toy implements the type system of [1] it presents some differences from the theory in [1]:

- There is not support for let expressions, since they are not supported in the original Toy system. However, the system support `where` declarations which play a role similar to monomorphic let expressions (see monomorphicWhere.toy, also contained in the directory `examples/typeSystem`).
- The system only support *classical* data declarations, so constructors with *non transparent* types as `cont :: A -> container` or `key :: A -> (A -> int) -> key` are not supported. Therefore the only way of constructing opaque patterns is using partially applied function symbols.

Error: Macro TOC(None) failed

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

Download & instalation

The zip file toy2safeOP.zip contains the complete Toy system with multiplatform support (Linux/Windows) and the user manual. There is not needed any instalation procedure, simply unzip the file. Once extracted, you will have a `toy2safeOP` folder with all the toy source files (Prolog files) and several directories. The most important are:

- `examples` contains examples of different features of Toy
- `docs` contains the user manual of the original Toy 2.3.1 system

Usage

In the folder there are two executable files for the different platforms:

- For Linux systems, run `toyLinux`
- For Windoes systems, run `toywin.exe`

You will see an interactive interpreter:

```
user@machine:~/toy2safeOP$ ./toyLinux


Toy 2.3.1b: A Constraint Functional Logic Language.
<< Safe Opaque Patterns Edition >>
(c) 1997-2011


Type "/h" for help.

Toy>
```

The Toy system accepts several commands (see section 1.5 in the user manual for a complete description). The following is a list of the most important ones:

- `/h`: shows the help menu
- `/cd(<Dir>)`: changes the current working directory to <Dir>
- `/q` or `/e`: exits the system
- `/run(<File>)`: compiles and loads the file <File>.toy
- `/type(<Expr>)`: shows the type of the expression <Expr>

Examples

Here is an example of how the system rejects the problematic polymorphic casting. The code of the program can be found in `examples/typeSystem/polymorphicCasting.toy` in the Toy directory and also in polymorphicCasting.toy.

```
Toy> /cd(examples/typeSystem)

Current working directory is /home/user/toy2safeOP/examples/typeSystem

Toy> /run(polymorphicCasting)

Checking syntax
..................................................Done.

Checking dependencies...Done.



Checking types..

TYPE ERROR: Variables X are critical in rule unpack/1
Variable type is assumed to go on the inference.

.Done



THERE ARE SOME TYPE ERROR. NO GENERATED CODE !!!!
Toy> /q
```

The first step is changing the working directory to the `examples/typeSystem` directory. Then, we compile and load the program `polymorphicCasting.toy`. As can be observed, the compilation fails since there is a type error in the first rule of `unpack` because the variable `X` is critical. Finally, we exit the system.

If the program is correct, we can show the inferred types for the functions and perform some evaluations. In the following example we use `examples/typeSystem/nocritical` (also in nocritical.toy):

```
Toy> /run(examples/typeSystem/nocritical)

Checking syntax
..............................................Done.

Checking dependencies...Done.



Checking types..Done


Generating code...
```

```
...................Done


PROCESS COMPLETE
Toy> /type(f)
f::(_A -> _A) -> bool

Elapsed time: 0 ms.

Toy> /type(f (snd [true,true]))
f(snd(true:true:[]))::bool

Elapsed time: 0 ms.

Toy> f (snd [true,true]) == L
      { L -> true }
      Elapsed time: 0 ms.
sol.1, more solutions (y/n/d/a) [y]? y
      no
      Elapsed time: 0 ms.
Toy> f (snd [X,Y]) == true
      { X -> true,
        Y -> true }
      Elapsed time: 0 ms.
sol.1, more solutions (y/n/d/a) [y]?
      no
      Elapsed time: 0 ms.
Toy> /q
```

First, we compile and load the program `examples/typeSystem/nocritical`. Since it is well-typed, we proceed to check the type of the function `f`} and the expression `f (snd [true,true])`. We also perform the evaluation of `f (snd [true,true])` obtaining `true` as the only solution, and the evaluation of `f (snd [X,Y]) == true` which finds `X -> true` and `Y -> true` as the only bindings for the variables.

Apart from polymorphicCasting.toy, nocritical.toy and monomorphicWhere.toy, the directory `examples/typeSystem` contains more examples showing different features of the type system. The majority of them appear in [1]:

- illtyped.toy: Functions with critical variables (therefore rejected by the system) which do no produce problems during evaluation
- illtypedWhere.toy: Similar to `illtyped.toy` but opaque patterns are moved to `where` declarations
- illtyped2.toy: Functions with critical variables which break the type preservation property
- illtyped2Where.toy: Similar to `illtyped2.toy` but opaque patterns are moved to `where` declarations
- polymorphicCastingWhere.toy: Similar to `polymorphicCasting.toy` but opaque patterns are moved to `where` declarations
- stratified.toy: Well-typed program with dependent functions, so the inference must proceed in order of dependency
- ugly.toy: Ill-typed program because the inferred typed for the rules is not a variant of the declared type
- welltyped.toy: Well-typed program with independent functions

References

1. ^ Francisco J. López-Fraguas and Enrique Martin-Martin and Juan Rodríguez-Hortalá, New Results on Type Systems for Functional Logic Programming, Lecture Notes in Computer Science, 5979, 2010, 5979, pages 128-144

Contact

The Toy system is developed by the Declarative Programming Group of the Universidad Complutense de Madrid. However, this particular branch has been developed by Francisco Javier López Fraguas, Enrique Martín Martín and Juan Rodríguez Hortalá, at the Department of Computer Systems and Computing, Universidad Complutense de Madrid, Spain.