Liberal typing in Toy

This page contains a branch of the Toy 2.3.1 system which uses the liberal type system presented in [1]. Since it is based in Toy 2.3.1 it has only support for 'classical' data declarations. Therefore neither GADT-like data declarations nor *non transparent* constructors as `cont :: A -> container` or `key :: A -> (A -> int) -> key` (also known as *existentially quanlified constructors*) appearing in [1] are supported.

Error: Macro TOC(None) failed

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

Download & instalation

The zip file toy2liberal.zip contains the complete Toy system with multiplatform support (Linux/Windows) and the user manual. There is not need for any instalation procedure, simply unzip the file. Once extracted, you will have a `toy2liberal` folder with all the toy source files (Prolog files) and several directories. The most important ones 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@computer:~/toy2liberal$ ./toyLinux


Toy 2.3.1c: A Constraint Functional Logic Language.
<< Liberal Types 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 equality as a *type-indexed* function. The source code can be found in EqualityTypeIndexed.toy or in `examples/typeSystem/EqualityTypeIndexed.toy` in the instalation directory.

```
em

.....................................................................................................
........Done.


.........................Done

.....................................................................................................
```

The first step is changing the working directory to the `examples/typeSystem` directory. Then we compile and load the file `EqualityTypeIndexed.toy` using the command `run`. Since it is well-typed, we perform some reductions. The first one evaluates `eq true false` to `false`, the second one binds the variable `X` in `eq false X == true` to `false`, and the third one reduces the equality of lists `eq [true,false] [true,false] == L` to `true`. Finally, we check that the type of `eq z (s z)` is `bool` and exit the interpreter.

The following example shows the rejection of the program `Examples.toy` when the rules for `unpack` are uncommented. The program can be found in Examples.toy or in `examples/typeSystem/Examples.toy` in the instalation directory.

```
xamples/typeSystem/Examples)

ntax
.....................................................Done.

pendencies...Done.


pes...

 The types <_A, X::_A> inferred for the right-hand side do not match <_B, X::_C> inferred for the left-hand sid




OME TYPE ERROR. NO GENERATED CODE !!!!
```

We first compile and load the program `examples/typeSystem/Examples`. It is rejected because the types <_A, X::_A> inferred for the right-hand side do not match <_B, X::_C> in rule unpack/1, which violates the well-typedness criterion.

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

- GenericSize.toy: `size` as a generic function
- HOFOapply.toy: example of the translation from HO to FO using `apply`
- Opacity.toy: examples using opaque HO patterns
- Size.toy: `size` as a type-indexed function
- TypeClassTranslation.toy: translation of type classes using type-indexed functions and type witnesses

References

1. ^ Francisco J. López-Fraguas and Enrique Martin-Martin and Juan Rodríguez-Hortalá, Liberal Typing for Functional Logic Programs, Lecture Notes in Computer Science, 6461, 2010, 6461, pages 80-96

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.