Hanus: Embedding Janus in Haskell

Joris ten Tusscher, Joris Burgers, Ivo Gabe de Wolff,
Cas van der Rest, Orestis Melkonian

Faculty of Science, Utrecht University

Introduction

- **Janus** is an imperative reversible programming language, meaning that every computation and function can be reversed.
- **Hanus** is an extended implementation of Janus that can be compiled straight to Haskell. Because of this, Hanus contains many awesome Haskell features that are unthinkable in regular Janus!

Reverse your program

- The inverse of a program can be computed automatically.

Division

```
[hanus|
procedure divide(x :: Int, y :: Int, z :: Int){
    from x >= y && z == 0 loop
        z += 1;
        x -= y;
    until x < y;
}|]
```

Reverse of division

```
[hanus|
procedure divide'(x :: Int, y :: Int, z :: Int){
    from x < y loop
        x += y;
        z -= 1;
    until x >= y && z == 0;
}|]
```

- Procedures are called with either the *call* or the *uncall* keyword. The *main* procedure is called automatically.

Execution

```
[hanus|
procedure main(x :: Int, y :: Int, z :: Int){
        call divide(x, y, z);
        uncall divide(x, y, z);
}|]
```

Syntactic checking

- By using *QuasiQuotation*, the programmer gets notified of syntactic errors at compile-time!

Code

```
[hanus|procedure main() {
  local n : Int = 10;
    n += 10;
  delocal 20;
}|]
```

Error

```
Exception when trying to run compile-time code:
  Parsing of Janus code failed in file ....
  First error:
--  Expecting "::" at position LineCol 2 10
```

Semantic checking (Janus side)

- *Hanus* also reports semantic errors, such as violating Janus-specific constraints for expressions.

Code

```
count :: BinaryTree a -> Int
[hanus|
    a :: BinaryTree Int;
    procedure main() {
        createNode a;
        a.value += (count a.left) + a.value
    }
|]
```

Error

```
Semantic Error (line 6, col 23):
    Assigned variable appears on the left-hand side.
```

Semantic checking (Haskell side)

- Since regular Haskell programs are generated, users also get error messages for *anti-quoted* Haskell expressions.

Code

```
[hanus|
    init :: Int;
    a :: BinaryTree Int;
    procedure main() {
        createNode a;
        a.nodeValue += map (+ 1) init;
}|]
```

Error

```
- Could not match expected type Int with actual
  type [Integer]
- In the expression: map (+ 1) i
```

Haskell Power

- The programmer can add additional operators by defining functions for forward and backward execution.
- We can define an operator that works on all Functors:

Definition

```
(=$$) :: Functor f => Operator (f a) (Operator a b, b)
(=$$) = Operator forward backward
  where
    forward  f (Operator fwd _, x) = fmap (`fwd` x) f
    backward f (Operator _ bwd, x) = fmap (`bwd` x) f
```

Usage

```
procedure increase(tree :: BinaryTree Int) {
  tree =$$ (+=, 42);
}
```

- Besides operators, the programmer can also define field and array indexers which allow you to use `tree.leftChild` and `array[x]` on the left hand side.