

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

```
1 [hanus|
2 procedure divide(x :: Int, y :: Int, z :: Int){
3   from x >= y && z == 0 loop
4     z += 1;
5     x -= y;
6   until x < y;
7 }|]
```

REVERSE OF DIVISION

AUTOMATICALLY COMPUTED

```
1 [hanus|
2 procedure reverse_divide(x :: Int, y :: Int, z :: Int){
3   from x < y loop
4     x += y;
5     z -= 1;
6   until x >= y && z == 0;
7 }|]
```

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

► EXECUTION

```
1 [hanus|
2 procedure main(x :: Int, y :: Int, z :: Int){
3   call divide(x, y, z);
4   uncall divide(x, y, z);
5 }|]
```

Syntactic Checking

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

CODE

```
1 [hanus|procedure main() {
2   local n : Int = 10;
3   n += 10;
4   delocal n == 20;
5 }|]
```

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

```
1 count :: BinaryTree a -> Int
2 [hanus|
3   a :: BinaryTree Int;
4   procedure main() {
5     createNode a;
6     a.value += (count a.left) + a.value
7   }
8 ]|]
```

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

```
1 [hanus|
2   init :: Int;
3   a :: BinaryTree Int;
4   procedure main() {
5     createNode a;
6     a.nodeValue += map (+ 1) init;
7 }|]
```

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

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

USAGE

```
1 procedure increase(tree :: BinaryTree Int) {
2   tree ==$$ (+, 42);
3 }
```

- 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.