Project Report

Concepts of Program Design

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

1 Problem

1.1 Experiment

1.2 Janus - Reversible Computation

2 Methodology

2.1 Expected outcome

2.2 Planning

3 Results

3.1 Achievements

3.1.1 Eval

An important part of Hanus is the eval module. Eval is responsible for the actual Hanus compilation: it takes a Hanus AST and compiles it to Haskell. In this section, the chosen compilation tactic for some of the more important or interesting types of the Hanus AST will be discussed.

Declarations
There are two types of declarations in a Hanus program:

1. Global variable declarations. These will be explained in more detail later on in this section.

2. Procedure declarations. Procedures are compiled in Haskell to functions that consist of a do statement, since do statements come closest to the imperative nature of a Hanus procedure in terms of structure. The do block consists

Variables
Variables can be declared in Hanus in two different ways.

1. Globally. In this case, a variable is declared at the top level with a name and type. The programmer cannot explicitly give it an initial value, instead it will get the default value for that type implicitly. Global variables, as their name suggests, can automatically be used anywhere in the program, i.e. they don't have to be passed around. Because of this, Hanus makes it flat out *impossible* to pass global values around as function arguments.

2. Locally. Local Hanus variables are declared as follows:

   ```
   local <varname> :: <vartype> = <initial_value>;
           codeblock in which the local variable can be used and updated.
   unlocal <final_value>;
   ```

   By the time the unlocal statement is being executed, the local variable should have the specified final value. Note however, that the final value can also be a non-literal expression. The unlocal block is there to function as initial value for the local variable if the procedure it is part of is being uncalled, i.e. reversed: in that case, the local variable needs what is usually its final value as its initial value, and vice versa. An unlocal block is implemented as a `let-in` Haskell statement, because this automatically takes care of removing a local variable from scope after the unlocal line.

Every Hanus procedure p is compiled to two Haskell functions: a function p that does exactly what Hanus procedure p should do, and a function p' that is the inverse of p.

3.2 Goal/planning adjustments

4 Reflection

4.1 Good/bad surprises

4.2 Problems along the way

5 Appendix

5.1 Repo link

The public repository can be found at https://github.com/joristt/Hanus.

5.2 Code navigation