Semantic Ideas

Andrew Hirsch

September 8, 2012

1 Introduction

In Pony, we allow users to define extensions to a base language as transformations of that language. However, a user may be using more than one transformation at any particular time. If the user is attempting to apply transformations that conflict with each other, data may be overwritten by transformations in such a way that the transformations are no longer semantics-preserving; indeed, we may no longer create valid C99 code.

Most compilers will tell us when we are doing something wrong. Either they will give an error if given code that they cannot process, or they will give a warning if they recognize a pattern that is commonly a bug. We would like for Pony to detect such collisions for us, and give us a similar error.

To achieve this, we exploit the use of Data.Generics ("Scrap your boilerplate"-style coding) and the structure of programming languages as semantic trees.

2 Strong Collision

In strong collision, we detect possible collisions between two transformations. We do this by looking at the code of the transformations; it should only run on certain types of nodes. The central idea is that we create a type theory the abstract syntax tree of the (possibly extended) language the transformation runs on.

The existence of BNF suggests that this is possible: if we have a BNF formula as follows:

```
<expression> ::= <expression> + <term>
               | <expression> - <term>
               | <term>
```

Then there must be a datatype in the tree correspondingly:

```
data expression = Plus expression term
                | Minus expression term
                | Standalone term
```

It should be possible (through dynamic programming) to create a listing of all node types that can be reached from an `expression` node. Any transformation that works on an `expression` node also affects all nodes reachable from the `expression` node. Thus, we can have a type theory of nodes with a subtyping relation.

Now, the difficulty comes in detecting whether a transformation will run on nodes of a type. This is the same as giving this transformation a type.

In order to work on a type, a transformation must deconstruct the type and then construct a new type. This means that we can give such a type to a term and warn the user if collision is possible.

3 Weak Collision

In weak collision, we detect only possible collisions between two transformations *on a particular piece of code*. This requires a bit more work, but allows for pony transformations to be used in more general cases.

To do weak collision, we use the Curry-Howard correspondence to get a logical version of the transformations' Haskell code. We can then unify these over the database of tree nodes, using standard techniques of logic programming.

This requires a library for getting logical forms of Haskell code. Currently, I am only aware of one library, and it only finds the propositional forms of types. Finding such a library, or a work-around, is paramount.

4 Better Weak Collision?

Because our version of weak collision only works on the node level, it is possible that two transformations might be rejected because they work on the same node, but do not actually overwrite each other. However, detecting this kind of "better weak collision" is difficult, and is probably left for another day.

5 Better Strong Collision?

The holy grail of this work would be to find an algorithm that would detect that two transformations are incompatible in **exactly** the cases where they may overwrite one another. However, this is an uncomputable problem (see: Rice's Theorem). It may very well be possible to create better versions of strong collision. However, this is difficult work, along the same lines of the work on halting (see: Microsoft Terminator).