ART Memo 9 (Version 0)
Towards a Portable Tracer

Olaf Chitil

August 20, 2000

Abstract

The current redex trail system is integrated into nhc, only the redex trail browser is a separate program. Here we collect some ideas about how the creation of the redex trail can be made portable to other Haskell compilers.

1 Introduction

The redex trail system is based on a source-to-source transformation. Taking the front end and the source-to-source transformation from the current nhc implementation and adding a Haskell pretty printer to convert the syntax tree into a Haskell program basically should make the redex trail creation part of the redex trail system portable to other Haskell systems like ghc.

However, the added source-to-source transformation is not the only modification of nhc to enable tracing. The compiler and its runtime system are modified at several other spots.

Our aim is a redex trail system that consists of a source-to-source transformation, a support library, with a clearly-defined interface to supporting operations that a Haskell implementation must provide, and the redex trail browser.

2 Using the Front End of nhc

It surely makes sense to reuse the parser and syntax tree of nhc for the portable transformation, but should we really use the whole front end? Does the transformation need to know anything about imported modules? Is a symboltable needed at all? Is not the whole transformation designed to be local?

Some rethinking might be required to recognise cafs (arity is currently used) and to construct the source reference and identifier table (Section 4), but the resulting implementation would be much simpler and smaller.

We would, however, give up the flexibility to use additional information that is availabe in nhc (interface files; even types, if transformation was moved).

The first step into the indicated direction would be to move the transformation within nhc directly behind the parser.

3 Unsafe Functions in the Tracing Program

The tracing program, more specifically the Haskell library that has to be imported by it, uses the unsafe function `sameAs` to compare pointers. There are two alternative solutions: We can demand a Haskell system to provide this function, ghc and Hugs do so. Alternatively we can modify the transformation to work without `sameAs`. It is used to test if an application is a redex, that is, if the number of arguments to a function is equal to its arity. We could implement this test by storing the arity with every function, as Freja does.

4 The Source Reference and Identifier Name Table

A trace contains references to the source reference and identifier name table. The tracing transformation also creates this table. Currently the table is part of the program. However, for file archiving it will be made separate and hence has to be redesigned and especially the linking of the tables of all program modules has to be implemented. The new mechanism for creating the table should be independent of nhc.

5 The Prelude

The Haskell prelude has to be transformed as well. This is needed, if we want to trace the prelude. However, even when the prelude is trusted, the transformation requires that the definitions of the trusted functions are transformed. At least for the definitions of some higher-order functions this seems unavoidable.

The first problem is that the prelude is not a complete Haskell module. Some data types and functions are not defined in Haskell but are hard-coded in the compiler. However, we need transformed versions of these as well.

The second problem is that a Haskell system knows the semantics of all prelude functions and is free to use them internally in the compiler. For example, pattern matching of numeric constants and $n + k$ patterns is translated into the use of overloaded `==` and `-`. The tracing transformation could remove such syntactic sugar itself, at the expense of making it more complicated, but in principal a Haskell system may make use of its knowledge of prelude functions for any kind of purposes, especially optimisation.

Hence a tracing program probably has to import both the original, unchanged prelude (as qualified) and a tracing prelude (which needs a new name). All explicit imports of the prelude have to be changed to imports of the tracing prelude. Writing the whole tracing prelude by hand would probably be rather error-prone and hard work. It would need to be created by the tracing transformation from a normal prelude. However, for the parts that are hard-coded into compilers hand-written tracing versions would still be required.

6 The Runtime System

For tracing, nhc links an extended version of the runtime system with the program. These extensions fulfill several purposes which have to be inspected.

6.1 The Garbage Collector

The garbage collector is extended to remove superfluous occurrences of `Sat` nodes from the redex trail.

The garbage collector may also be used for pruning.

File archiving is also performed by the garbage collector. When file archiving is used the previously mentioned two tasks have no longer to be performed by the garbage collector.

6.2 Interruption of the Program

The runtime system enables the user to interrupt a program by pressing Control-C. The exception handling mechanism of ghc does not suffice to implement this, because the system also requires the term that was under evaluation to have the starting point for the redex trail.

6.3 Communication with the Browser

After termination of the normal program execution or interruption the runtime system takes over, connects to the redex trail browser and communicates with it, mainly to send parts of the redex trail.

This part could be written in Haskell itself, using an unsafe function that determines if a part of a redex trail exists or if it is still unevaluated (only necessary if superfluous `Sat` nodes still exist).

6.4 Further Features

The runtime system collects some statistical data (number of reductions).

7 Pragmas

Comments delimited by `{-#` and `#-}` contain pragmas for the compiler. Unfortunately the parser already dismisses these pragmas, they are not part of the syntax tree. Hence the question arises of how to get them into the tracing program.

For some pragmas simple copying of their content into the tracing program may not suffice, they have to be transformed as well. The most obvious example for such a pragma is the RULES pragma. It contains a rewriting rule that is to be used by the compiler, mostly for optimisations. It consists of terms and types and would need to be transformed similar to a variable definition. Because of the modified types an untransformed rule does not make any sense in the tracing program. Possibly we do not want rules at all, because we may not want to trace optimised programs. However, many optimisations of ghc are now based on rules and without them ghc will produce rather inefficient code, especially for the prelude and standard libraries (so rules are connected with the problem of how to handle the prelude).