ART Memo 13 (Version 0)
Portable File Archiving

Olaf Chitil

January 12, 2001

Abstract

Here we propose a scheme for writing the redex trail into a file instead of constructing it in the heap. The scheme is based on redefining the Haskell combinators that build the redex trail and which are introduced by the redex trail transformation so that they write a file using unsafePerformIO.

1 Goals

- Creation of the full redex trail for an arbitrary long computation. The traced program shall only use heap space proportional to the heap space used by the untraced program (aim: factor of 2). The only limit shall be the disk space.
- The redex trail can be used with various browsers.
- The redex trail can be viewed any time, it can be archived.
- The redex trail is portable. It can be viewed on any system that has a browser.
- The file archiving implementation is portable.
- The file archiving implementation requires limited effort.

Improved runtime is desirable, but it is unclear if the gain in reduced garbage collection time will offset the overhead of writing to a file.

2 General Comments

The implementation does not feature pruning, which does not make much sense in this setting anyway.

This version does not feature trusting. It seems quite easy to adapt the current trusting scheme to the file archiving version. Basically every Trace requires an additional boolean argument to indicate if it is being trusted (then recursively following a trace is no longer necessary for determining trustedness). However, the current trusting mechanism is not satisfactory and requires some rethinking. Also, for file archiving trusting may not be necessary. It may be more interesting to dynamically use trusting in the browser to reduce the amount of visible information.

The scheme is based on redefining the trail-building combinators that are introduced by the redex trail transformation. Thus the transformation can be left unchanged, only the definition of these combinators has to be changed (except that in the Hat implementation the transformation sometimes handles traces directly instead of abstracting this work). Hence for the use of the transformation functions, the transformation itself, we refer to Memo 12 on the redex trail transformation. The new definitions write trace nodes to file using unsafePerformIO.

Note that it is possible to avoid unsafePerformIO by wrapping every value into the IO data constructor, like

newtype R a = R (IO (a, Trace))

However, that would require the transformation to handle evaluation order in pattern matching and for primitive functions explicitly. Such an implementation would be closer to a Haskell interpreter.

In the definitions of the transformation functions given in the following sections care has been taken to ensure that a trace knot is written to file at the same time (or even earlier) as it would be allocated on the heap otherwise. Basically a trace shall never be a thunk. The strictness in the definitions ensures that.

Note: seq has to refer to the untraced seq of the untraced prelude. In fact, the implicitly imported prelude should be untraced.

2.1 Static Tables

The tables of source references, identifiers and strings can be stored in a separate file as discussed in Memo 3. It may be easier for a first implementation to write this file every time a traced program is run.

2.2 Output, interruption information

The program output with its traces and error messages and an interruption point (Shall it be possible to resume an interrupted computation?) can also be handled as discussed in Memo 3. (It would be nice to obtain a usable trail file even when the traced program core dumps or hangs; but that does not seem to be achievable without efficiency loss.) Note that a general scheme for handling any IO still has to be formulated (see also Memo 7 on the semantics of redex trails).

2.3 Browsing Archived Redex Trails

The discussion of Memo 3 applies as well. The best compromise is to have either a separate program or a separate part of the browser between the file and the browser, so that the browser can still use message-passing. However, this part is started by the browser, the user need not be aware of its existence.

3 The Data Types

data R a = R a !Trace

The wrapper is strict in the trace. This is already sensible for the heap redex trail version. For the file redex trail version it makes some definitions simpler, although using seq instead is possible.

type Fun a b = Trace $\rightarrow$ R a $\rightarrow$ R b

(unchanged)

newtype Trace = T FilePointer

Records in the File:

Ap FilePointer [FilePointer] SR	size info needed for the list
Nm FilePointer NmType SR	both NmType and SR are pointers to static info
Ind FilePointer Pointer	first has full history, second is creator
Sat FilePointer	only one argument which is updated
Root	only needed once, could use pointer=0

4 Writing to a File

The IO file operations of the standard library are unsatisfactory, because they only support files of type String. The system may add return characters and do even more weird things to handle unicode characters.

For the redex trail it is best to write bytes. We need to know (or fix) the size of a file position (4 bytes?) and know if a file starts with position 0 or 1. (probably BinaryMode from the hslibs should be useable; I suppose Chars are then considered as bytes).

4.1 The unsafe Operations

writeTrail writes the bytes for a trace node to the trail, returning a pointer to this node. writeTrailAt overwrites a given trace node at a given point.

```
trailFilename :: String
trailFilename = "Program.trl"    needs to be set by runtime system

trailFileHandle :: FileHandle
trailFileHandle = unsafePerformIO $ openFile trailFilename WriteMode

trailFilePosition :: IORef FilePointer
trailFilePosition = unsafePerformIO $ newIORef 0

writeTrail :: [Word8] → FilePointer
writeTrail ws = unsafePerformIO $ do
  pos ← readIORef trailFilePosition
  mapM (hPutWord8 trailFileHandle) ws
  writeIORef trailFilePosition (pos + length ws)
  return pos

writeTrailAt :: FilePointer → [Word8] → ()
writeTrailAt pos ws = unsafePerformIO $ do
  hSeek trailFileHandle AbsoluteSeek pos
  mapM (hPutWord8 trailFileHandle) ws
  endPos ← readIORef trailFilePosition
  hSeek trailFileHandle AbsoluteSeek endPos
```

It is probably cheaper to keep track of the current file position in trailFilePosition instead of always making an operating system call.

4.2 Creation of the Redex Trail

```
mkAp :: Trace → [Trace] → SR → Trace

mkAp t ts s = T $ writeTrail $
  1 : (trace2Word8 t) ++ int2Word8 (length ts) ++ concatMap trace2Word8 ts
  ++ sr2Word8 s

mkNm :: Trace → NmType → SR → Trace

mkNm t nm s = T $ writeTrail $
  2 : (trace2Word8 t) ++ (nm2Word8nm) ++ sr2Word8 s

mkInd :: Trace → Trace → Trace

mkInd t1 t2 = T $ writeTrail $
  3 : (trace2Word8 t1) ++ (trace2Word8 t2)

mkSat :: Trace → Trace

mkSat t = T $ writeTrail $
  4 : (trace2Word8 t)
```

Saving updates of Sats in a separate file and processing this file after termination of the traced program is probably more efficient. However, a first version can simply update in the redex file directly.

The first argument of updateSat is the Sat that shall be updated and the second argument contains the new trace it shall point to. The function returns the second argument. Note that a Sat node is basically an indirection (different from the indirection node).

$$\text{updateSat} :: \text{Trace} \to \text{R}\ a \to \text{R}\ a$$

$$\text{updateSat}\ (\text{T}\ pos)\ e@(\text{R}\ r\ rt) = \text{writeTrailAt}\ (pos + 1)\ (\text{trace2Word8}\ rt)\ \text{`seq`}\ e$$

5 Transformation of a Pattern

No transformation function used.

6 Transformation of Expressions

6.1 Variable

For projections:

$$\text{indir} :: \text{Trace} \to \text{R}\ a \to \text{R}\ a$$

$$\text{indir}\ t\ (\text{R}\ v\ t') = \text{R}\ v\ (\text{mkInd}\ t\ t')$$

6.2 Data Constructor Application

ToDo.

Apply data constructor to wrapped arguments and wrap it with a trace, an application node for the constructor application.

$$\text{con}_i :: \text{SR} \to \text{Trace} \to (\text{R}\ a_1 \to \ldots \to \text{R}\ a_n \to b) \to \text{NmType} \to \text{R}\ a_1 \to \ldots \to \text{R}\ a_n \to \text{R}\ b$$

$$\begin{aligned}
&\text{con}_0\ s\ t\ c\ nm = \text{R}\ c\ (\text{Nm}\ t\ nm\ s)\\
&\text{con}_i\ s\ t\ c\ nm\ a_1@(\text{R}\ _\ t_1)\ \ldots\ a_i@(\text{R}\ _\ t_i) = \text{R}\ (c\ a_1\ \ldots\ a_i)\ (\text{Ap}\ t\ [\text{Nm}\ t\ nm, t_1, \ldots, t_i]\ s)
\end{aligned}$$

Transform a function into a wrapped function with the given trace. For $i > 0$:

$$\text{cn}_i :: (a_1 \to \ldots \to a_i \to b) \to \text{Trace} \to (a_1\ \text{`Fun`}\ \ldots\ \text{`Fun`}\ a_i\ \text{`Fun`}\ b)$$

$$\text{cn}_i\ f\ t = t\ \text{`seq`}\ (\text{R}\ (\lambda t_1\ a_1 \to \ldots \text{R}\ (\lambda t_i\ a_i \to R\ (f\ a_1\ \ldots\ a_i)\ t)\ t \ldots)\ t)$$

Create an application node.

$$\text{pa}_i :: \text{SR} \to \text{Trace} \to \text{NmType} \to \text{R}\ a_1 \to \ldots \to \text{R}\ a_i \to \text{Trace}$$

$$\begin{aligned}
&\text{pa}_0\ s\ t\ nm = \text{Nm}\ t\ nm\ s\\
&\text{pa}_i\ s\ t\ nm\ a_1@(\text{R}\ _\ t_1)\ \ldots\ a_i@(\text{R}\ _\ t_i) = \text{Ap}\ t\ [\text{Nm}\ t\ nm\ s, t_1, \ldots, t_i]\ s
\end{aligned}$$

6.3 Other Application

Except for some additional seqs there is nothing special here.

Because rap_i only differs with respect to trusting from ap_i, here the functions are equal.

$$\mathrm{ap}_i :: \mathrm{SR} \to \mathrm{Trace} \to \mathrm{R}\ (a_1\ \text{‘Fun‘}\ \ldots\ \text{‘Fun‘}\ a_i\ \text{‘Fun‘}\ b) \to \mathrm{R}\ a_1 \to \ldots \to \mathrm{R}\ a_i \to R\ b$$

$$\begin{array}{l} \mathrm{ap}_1\ s\ t\ (\mathrm{R}\ rf\ tf)\ a_1@(\mathrm{R}\ \_\ t_1) = \\ \quad \text{let } t' = \mathrm{mkAp}\ t\ [tf, t_1]\ s \text{ in } t'\ \text{‘seq‘}\ rf\ t'\ a_1 \end{array}$$

$$\begin{array}{l} \mathrm{ap}_i\ s\ t\ (\mathrm{R}\ rf\ tf)\ a_1@(\mathrm{R}\ \_\ t_1)\ \ldots\ a_i@(\mathrm{R}\ \_\ t_i) = \\ \quad \text{let } t' = \mathrm{mkAp}\ t\ [tf, t_1, \ldots, t_i]\ s \text{ in } t'\ \text{‘seq‘}\ \mathrm{pap}_{i-1}\ s\ t'\ (rf\ t'\ a_1)\ a_2\ \ldots\ a_i \end{array}$$

$i \geq 0$:

$$\mathrm{pap}_i :: \mathrm{SR} \to \mathrm{Trace} \to \mathrm{R}\ (a_1\ \text{‘Fun‘}\ \ldots\ \text{‘Fun‘}\ a_i\ \text{‘Fun‘}\ b) \to \mathrm{R}\ a_1 \to \ldots \to \mathrm{R}\ a_i \to R\ b$$

$$\mathrm{pap}_0\ s\ t\ e = e$$

$$\begin{array}{l} \mathrm{pap}_i\ s\ t\ (\mathrm{R}\ rf\ tf)\ a_1@(\mathrm{R}\ \_\ t_1)\ \ldots\ a_i@(\mathrm{R}\ \_\ t_i) = \\ \quad \text{if } t\ \text{‘sameAs‘}\ tf \\ \qquad \text{then } \mathrm{pap}_{i-1}\ s\ t\ (rf\ t\ a_1)\ a_2\ \ldots\ a_i \\ \qquad \text{else let } t' = \mathrm{mkAp}\ t\ [tf, t_1, \ldots, t_i]\ s \text{ in } t'\ \text{‘seq‘}\ \mathrm{pap}_{i-1}\ s\ t'\ (rf\ t'\ a_1)\ a_2\ \ldots\ a_i \end{array}$$

6.4 Lambda Abstraction

$$\begin{array}{l} \mathrm{fun}_i :: \mathrm{NmType} \to (\mathrm{Trace} \to a_1 \to \ldots \to a_i \to b) \to SR \to \mathrm{Trace} \\ \qquad \to \mathrm{R}(a_1\ \text{‘Fun‘}\ \ldots\ \text{‘Fun‘}\ a_i\ \text{‘Fun‘}\ b) \end{array}$$

$$\begin{array}{l} \mathrm{fun}_0\ nm\ rf\ \ s\ t = \text{let } t' = \mathrm{Nm}\ t\ nm\ s \\ \qquad\qquad\qquad\qquad ts = \mathrm{mkSat}\ t'\mathrm{R}\ r\ rt = \mathrm{enter}\ nm\ t'\ \$\ \mathrm{updateSat}\ ts\ \$\ (rf\ \ t) \\ \qquad\qquad\qquad \text{in } \mathrm{R}\ r\ ts \\ \mathrm{fun}_i\ nm\ rf\ \ s\ t = \\ \quad \mathrm{R}\ (\lambda t_1 a_1 \to \ldots \mathrm{R}\ (\lambda t_i a_i \to \text{let } ts = \mathrm{mkSat}\ t_i \\ \qquad\qquad\qquad\qquad\qquad\qquad \mathrm{R}\ r\ rt = \mathrm{enter}\ nm\ t_i\ \$\ \mathrm{updateSat}\ ts\ \$\ (rf\ \ t_i\ a_1\ \ldots\ a_i) \\ \qquad\qquad\qquad\qquad\qquad \text{in } \mathrm{R}\ r\ ts)\ t_{i-1} \\ \qquad \ldots t_1) \\ \qquad (\mathrm{Nm}\ t\ nm\ s) \end{array}$$

Note that updateSat is evaluated just after the application has been evaluated.

6.5 Let Expression

No transformation function used.

6.6 Case Expression

Only ap_1 and fun_1 used, if we ignore guards for the moment.

6.7 If Expression

$$\text{tif} :: \text{SR} \rightarrow \text{R Bool} \rightarrow (\text{Trace} \rightarrow a) \rightarrow (\text{Trace} \rightarrow a) \rightarrow \text{Trace} \rightarrow a$$

$$\begin{aligned}\text{tif}\ s\ (\text{R}\ iv\ it)\ a_1\ a_2\ t = {} & \text{let}\ t' = \text{Ap}\ t\ [\text{Nm}\ t\ \text{NTIf}\ s, it, t]\ s \\ & \text{in}\ t'\ \text{`seq`}\ \text{if}\ iv\ \text{then}\ a_1\ t'\ \text{else}\ a_2\ t'\end{aligned}$$

7 Transformation of Declarations