ART Memo 12 (Version 0)
The Redex Trail Transformation

Olaf Chitil

March 27, 2001

Abstract

Here we describe the transformation of a program into a program that computes its redex trail besides its normal result. The description is very close to the actual implementation.

1 General Comments

Everywhere where a runtime error may occur (e.g. no matching pattern) a call to the function fail is inserted by the transformation. Also the standard function error is replaced by fail (in the prelude definition). fail outputs an error message and then connects to the browser. If an exception handler could catch these errors, then the introduction of fail would be unnecessary. Note also that the current scheme would not work anymore if we added such exception handlers to the language.

The computation can only be interrupted by Control-C when a function is entered. The advantage is that the redex trail is in a well-defined state and that computation can be resumed. The latter would hardly be possible, if Control-C were caught by an exception handler.

The transformation assumes that a source reference of an expression is only the position where the expressions starts and hence some source references are equal, for example the source references of the expression fx and the f inside are equal. Source references that describe whole blocks would need to be handled with more care in the transformation.

2 The Data Types

data R a = R a Trace

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

(In the future this should be introduced as a newtype, so that in types $\rightarrow$ can simply be replaced by Fun, even when applied to less than two types.)

$$
\begin{aligned}
\text{data Trace} = \ & \text{Ap Trace [Trace] SR} \\
& \mid \text{Nm Trace NmType SR} \\
& \mid \text{Ind Trace Trace} \\
& \mid \text{Root} \\
& \mid \text{Sat Trace Trace} \\
& \mid \text{Pruned} \\
& \mid \text{Hidden Trace}
\end{aligned}
$$

3 Transformation of a Pattern

$$
\begin{aligned}
\mathcal{P}[\![C\ p_1\ \ldots\ p_i]\!] &= \text{R}\ (C\ \mathcal{P}[\![p_1]\!]\ \ldots\ \mathcal{P}[\![p_i]\!]) \\
\mathcal{P}[\![x]\!] &= x \\
\mathcal{P}[\![_]\!] &= _ \\
\mathcal{P}[\![x@p]\!] &= x@\mathcal{P}[\![p]\!] \\
\mathcal{P}[\![\sim p]\!] &= \sim \mathcal{P}[\![p]\!]
\end{aligned}
$$

4 Transformation of Expressions

The boolean argument states if the expression is in a non-projective context, that is, it is False if the expression is the result of the function in whose rhs it appears (not simply the rhs because of the case and if construct). It is used both for introduction of indirection nodes and for trusting.

Superscripts at expressions (e.g. M^s) are source references.

4.1 Variable

$x, x_1, \ldots$ are λ-bound variables. $y, y_1, \ldots$ are let-bound variables.

$$
\begin{aligned}
\mathcal{E}[\![x]\!]_t^{\text{True}} &= x \\
\mathcal{E}[\![x]\!]_t^{\text{False}} &= \text{indir}\ t\ x \\
\mathcal{E}[\![y^s]\!]_t &= y\ s\ t
\end{aligned}
$$

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

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

4.2 Data Constructor Application

$$
\mathcal{E}[\![(C\ M_1 \ldots M_i)^s]\!]_t = \begin{cases} \text{con}_i\ s\ t\ C\ \mathcal{N}[\![C]\!]\ \mathcal{E}[\![M_1]\!]_t^{\text{True}} \ldots \mathcal{E}[\![M_i]\!]_t^{\text{True}} & , \text{if arity}(C) = i \\ \text{cn}_{\text{arity}(C)-i}\ (C\ \mathcal{E}[\![M_1]\!]_t^{\text{True}} \ldots \mathcal{E}[\![M_i]\!]_t^{\text{True}}) & \\ \quad (\text{pa}_i\ s\ t\ \mathcal{N}[\![C]\!]\ \mathcal{E}[\![M_1]\!]_t^{\text{True}} \ldots \mathcal{E}[\![M_i]\!]_t^{\text{True}}) & , \text{otherwise} \end{cases}
$$

The repeated occurrences of $\mathcal{E}[\![M_i]\!]_t^{\text{True}}$ above leads to unnecessary computation and wrong sharing!

```
g x = (True : x)

f = (True :)

main = print (head (f []), head (g []))
```

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

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

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

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.

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

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

4.3 Other Application

$$\mathcal{E}[\![M\ N_1\ \ldots\ N_i]\!]_t^{\text{True}} = \text{ap}_i\ s\ t\ \mathcal{E}[\![M_1]\!]_t^{\text{True}}\ \ldots\ \mathcal{E}[\![M_i]\!]_t^{\text{True}}$$
$$\mathcal{E}[\![M\ N_1\ \ldots\ N_i]\!]_t^{\text{False}} = \text{rap}_i\ s\ t\ \mathcal{E}[\![M_1]\!]_t^{\text{True}}\ \ldots\ \mathcal{E}[\![M_i]\!]_t^{\text{True}}$$

If a trusted function is applied in a trusted context, then the application is not registered in the trail. In that case ap_i also marks the trace as hidden. In contrast, rap_i does not do the latter. Is the trace already marked as hidden?

rap_i is used within a projective context, otherwise ap_i is used.

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

$$\text{ap}_1\ s\ t\ (\text{R}\ rf\ tf)\ a_1@(\text{R}\ _\ t_1) =$$
  if trusted $t\ tf$
    then let $t' = \text{hidden}\ t$ in t' `seq` $rf\ t'\ a_1$
    else let $t' = \text{Ap}\ t\ [tf, t_1]\ s$ in $rf\ t'\ a_1$

$$\text{ap}_i\ s\ t\ (\text{R}\ rf\ tf)\ a_1@(\text{R}\ _\ t_1)\ \ldots\ a_i@(\text{R}\ _\ t_i) =$$
  if trusted $t\ tf$
    then let $t' = \text{hidden}\ t$ in t' `seq` $\text{pap}_{i-1}\ s\ t'\ (rf\ t'\ a_1)\ a_2\ \ldots\ a_i$
    else let $t' = \text{Ap}\ t\ [tf, t_1, \ldots, t_i]\ s$ in $\text{pap}_{i-1}\ s\ t'\ (rf\ t'\ a_1)\ a_2\ \ldots\ a_i$

$$\text{hidden} :: \text{Trace} \to \text{Trace}$$

$$\text{hidden}\ h@(\text{Hidden}\ _) = h$$
$$\text{hidden}\ t = \text{Hidden}\ t$$

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

$\text{rap}_1\ s\ t\ (\text{R}\ rf\ tf)\ a_1@(\text{R}\ _\ t_1) =$
  if trusted $t\ tf$
    then $rf\ t\ a_1$
    else let $t' = \text{Ap}\ t\ [tf, t_1]\ s$ in $rf\ t'\ a_1$

$\text{rap}_i\ s\ t\ (\text{R}\ rf\ tf)\ a_1@(\text{R}\ _\ t_1)\ \ldots\ a_i@(\text{R}\ _\ t_i) =$
  if trusted $t\ tf$
    then $\text{pap}_{i-1}\ s\ t\ (rf\ t\ a_1)\ a_2\ \ldots\ a_i$
    else let $t' = \text{Ap}\ t\ [tf, t_1, \ldots, t_i]\ s$ in $\text{pap}_{i-1}\ s\ t'\ (rf\ t'\ a_1)\ a_2\ \ldots\ a_i$

trusted :: Trace → Trace → Bool

trusted $t\ tf$ = trustFun t && trustFun tf

trustFun :: Trace → Bool

trustFun (Ap _ (tf : _) _) = trustFun tf
trustFun (Nm _ (NTTrusted _) _) = True
trustFun (Nm _ (NTId _) _) = False
trustFun (Nm _ NTLambda _) = False definitely wrong, should follow trace
trustFun (Nm t NTCase _) = trustFun t
trustFun (Nm t NTIf _) = trustFun t
trustFun (Nm t NTGuard _) = trustFun t
trustFun (Sat t _) = trustFun t
trustFun (Ind _ t) = trustFun t
trustFun (Root) = False
trustFun (Pruned) = False
trustFun (Hidden _) = True

The function trusted is actually a primitive C-function (to improve performance?). It also increases a reduction counter (but surprisingly never the counter from calls of suspected functions from trusted functions).

$i \geq 0$:

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

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

$\text{pap}_i\ s\ t\ (\text{R}\ rf\ tf)\ a_1@(\text{R}\ _\ t_1)\ \ldots\ a_i@(\text{R}\ _\ t_i) =$
  if t `sameAs` tf
    then $\text{pap}_{i-1}\ s\ t\ (rf\ t\ a_1)\ a_2\ \ldots\ a_i$
    else let $t' = \text{Ap}\ t\ [tf, t_1, \ldots, t_i]\ s$ in $\text{pap}_{i-1}\ s\ t'\ (rf\ t'\ a_1)\ a_2\ \ldots\ a_i$

4.4 Lambda Abstraction

$$\mathcal{E}[\![(\lambda x_1 \ldots x_i.M)^s]\!]_t = \text{fun}_i\ \mathcal{N}[\![\lambda]\!]\ (\lambda t' x_1 \ldots x_n.\mathcal{E}[\![M]\!]_{t'}^{\text{False}})\ s\ t$$

Actually the translated λ-abstraction uses a case-construct to test if the patterns match. Otherwise it calls a function that outputs an error message and directly connects to the browser.

$$\begin{aligned}
&\text{fun}_i :: \text{NmType} \to (\text{Trace} \to a_1 \to \ldots \to a_i \to b) \to SR \to \text{Trace} \\
&\qquad \to \text{R}(a_1\ \text{`Fun`}\ \ldots\ \text{`Fun`}\ a_i\ \text{`Fun`}\ b)
\end{aligned}$$

$$\begin{aligned}
&\text{fun}_0\ nm\ rf\ s\ t = \text{let}\ t' = \text{Nm}\ t\ nm\ s \\
&\qquad\qquad\qquad\qquad \text{R}\ r\ rt = \text{enter}\ nm\ t'\ (rf\ t) \\
&\qquad\qquad\qquad \text{in R}\ r\ (\text{Sat}\ t'\ rt) \\
&\text{fun}_i\ nm\ rf\ s\ t = \text{R}\ (\lambda t_1 a_1 \to \ldots \text{R}\ (\lambda t_i a_i \to \text{let R}\ r\ rt = \text{enter}\ nm\ t_i\ (rf\ t_i\ a_1\ \ldots\ a_i) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{in R}\ r\ (\text{Sat}\ t_i\ rt))\ t_{i-1} \\
&\qquad\qquad \ldots t_1) \\
&\qquad\qquad (\text{Nm}\ t\ nm\ s)
\end{aligned}$$

The primitive C-function enter connects to the redex trail browser, if the user pressed Control-C, execution is in single-stepping mode, etc. Otherwise it just returns the third argument.

4.5 Let Expression

$$\mathcal{E}[\![\text{let}\ \{decl_1; \ldots ; decl_i\}\ \text{in}\ M]\!]_t = \text{let}\ \{\mathcal{D}[\![decl_1]\!]_t; \ldots ; \mathcal{D}[\![decl_i]\!]_t\}\ \text{in}\ \mathcal{E}[\![M]\!]_t^{\text{False}}$$

4.6 Case Expression

$$\begin{aligned}
&\mathcal{E}[\![(\text{case}\ M\ \text{of}\ \{alt_1; \ldots ; alt_i\})^s]\!]_t^b = \\
&\text{ap}_1\ s\ t\ (\text{fun}_1\ \mathcal{N}[\![\text{case}]\!]\ (\lambda t' x \to \text{case}\ x\ \text{of}\ \{\mathcal{A}[\![alt_1]\!]_{t'}; \ldots ; \mathcal{A}[\![alt_i]\!]_{t'}\})\ s\ t)\ \mathcal{E}[\![M]\!]_t^b
\end{aligned}$$

Actually the transformation introduces a last alternative for the case that none of the previous patterns matches. That alternative calls a function that outputs an error message and directly connects to the browser.

$$\mathcal{A}[\![p \to M]\!]_t = \mathcal{P}[\![p]\!] \to \mathcal{E}[\![M]\!]_t^{\text{False}}$$

Simplified, without guards and local declarations.

4.7 If Expression

$$\mathcal{E}[\![\text{if}\ M\ \text{then}\ N_1\ \text{else}\ N_2]\!]_t^b = \text{tif}\ s\ \mathcal{E}[\![M]\!]_t^b\ (\lambda t' \to \mathcal{E}[\![N_1]\!]_{t'}^{\text{False}})\ (\lambda t' \to \mathcal{E}[\![N_2]\!]_{t'}^{\text{False}})\ t$$

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

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

trust is a C-function similar to trustFun.

5 Transformation of Declarations