# ART Memo 32 (Version 0)
# Tracing IO Actions

Olaf Chitil

April 8, 2002

## 1 Introduction

Values of type `IO a` currently look rather strange in the trace and the output file with its pointers into the trace is rather primitive and only fully describes output to standard output.

There are two separate issues: First, it needs to be determined which are the values of type `IO a` and how they are represented in the trace. Second, when actions are performed they need to be recorded in a log with pointers between the trace and the log in both directions.

## 2 Action Values

The type `IO a` is abstract. So we could just show all action values, that is values of type `IO a`, as something like {`IO`}.

Alternatively, we can consider every constant or application of type `IO a` that is (the application of) an *untraced* variable as action value. Thus `print 3`, `return True`, `print 3 >> return True` and `read >> (\..)` are all action values.

```
getNumber = do
  print "Number ? "
  read

getNumber = print "Number ? " >> read
```

Many action values will consists of un-informative $\lambda$-expressions. So a better presentation of $\lambda$-expressions would also improve the presentation of action values.

The problem with $\lambda$-expressions is that currently there is no representation of a binding construct in the trace. Trace expressions do not contain variables that are bound by an enclosing binding construct. It does not seem to be easy to extend the current method for writing trace expressions to binding constructs.

In practise the trace may represent an abstract value as *Abstract* node with the type name, and optionally the viewer may represent a value as the untraced expression that evaluated to that value.

These trace values are recorded in the trace by the normal tracing combinators.

# 3 Action Log

The current output file mixes the output of all actions, even if different parts are directed at different devices. Input is not registered properly at all; input values do not have parents.

The action log should be a sequence of performed actions. A performed action consists of two pointers to trace expressions: The first points to the result value of the performed action, the second points to the action value (as described above) itself. The parent of the result value is the whole performed action.

```
() <- print "Number ? "
"342" <- read
"content of file" <- hGetLine Handle
```

The performed actions are recorded in the trace by additional IO actions. Every transformed action is prefixed by an additional trace recording action. It is still unclear how the partitioning into actions according to tracedness status can be automated in the transformation. How avoid overlapping actions when untraced actions call other untraced actions?

Furthermore the problem of actions that were not performed although they were expected to be performed needs to be addressed. So any part of the action log, including an empty action log, must be linked to an action value which can be explored further.