Equations for Binding-Time Analysis

August 27, 2015

1 A Problem with the Binding-Time Analysis in Jones, Gomard, and Sestoft [1, §5.2]

This document gives a modified set of equations for binding-time analysis that aims to overcome certain problems with the equations given by Jones, Gomard, and Sestoft [1, §5.2]. In particular, their equations apparently do not account for a certain kind of congruence issue that arises when a function that always has static arguments calls a function that can have dynamic arguments.

Consider the following example:

```
(
  (define (goal x y z)
    (if (= x 0)
        z
        (call auxiliary y x)
  ))
  (define (auxiliary x y)
    (call goal (- y 1) 2 x)
  )
)
```

where the input binding-time pattern is (S, S, D). The division given by the binding-time-analysis equations from [1, §5.2] is $[\texttt{goal} \mapsto (S, S, D), \texttt{auxiliary} \mapsto (S, S)]$. Thus, the annotated program is

```
(
 (define (goal (x y) (z))
   (ifs (=s x 0)
        z
        (lift (calls auxiliary (y x) ())))
 ))
 (define (auxiliary (x y) ())
   (calld goal ((-s y 1) 2) ((lift x)))
 )
)
```

The specializer calls

```
reduce[(ifs (=s x 0) z (lift (calls auxiliary (y x) ()))), names, values]
```

which spawns a call

```
reduce[(lift (calls auxiliary (y x) ())), names, values]
```

where the lift case of reduce is defined by

```
(lift e) => list[quote, reduce[e, names, values]]
```

Consequently, a quoted expression is introduced whose body is the result of

```
reduce[(calls auxiliary (y x) ()), names, values]
```

The `calls` case of reduce spawns a call to reduce on the body of function `auxiliary`. However, because the body of `auxiliary` in the annotated program is

```
(calld goal ((-s y 1) 2) ((lift x)))
```

the result is a residuated call on a variant of `goal`. For example, if the value of `y` is 5, the overall quoted expression that would be created is

```
(quote (call goal-4:2 2))
```

When the specialized program is evaluated on some dynamic input, if the interpreter encounters the above quoted expression, it would yield the call expression (as an s-expression) instead of evaluating the call expression.

The problem is that function `auxiliary` is always called with static arguments (in this case `(call auxiliary y x)`), which in annotated form is `(calls auxiliary (y x) ())`; however, the body of `auxiliary` has a call on a function that has dynamic arguments, namely `goal` (i.e., `(call goal (- y 1) 2 x)`, which in annotated form is `(calld goal ((-s y 1) 2) ((lift x)))`).

There is a kind of congruence mismatch in this example: the (static) context expects a pure value to be produced by the specializer, whereas the presence of the dynamic operator `calld` causes a residual expression to be produced.

2 An Alternative Set of Binding-Time-Analysis Equations

To address the problem described above, we now give a modified set of equations for binding-time analysis below. First, we redefine the types:

$$\begin{array}{rcccc} t & \in & \text{BindingTime} & = & \{S, D\} \\ \tau & \in & \text{BTEnv} & = & \text{BindingTime}^* \times \text{BindingTime} \\ \texttt{div} & \in & \text{Monodivision} & = & \text{FuncName} \rightarrow \text{BTEnv} \end{array}$$

The main difference is the BTEnv type, which will now be a pair. The first element of the pair is a list of binding times, which correspond to the binding times of the formals of a function; the second element is the binding time of the function's result. Because we are using monovariant divisions, the environment for the formals should be greater than or equal to the binding times for all calls made to the function, and the return binding time for each call made to the function should be greater than or equal to the function's result binding time.

Next, we redefine the less-than-or-equal-to operators for binding times and binding-time environments. For binding times t and r we have

$$t \sqsubseteq r \text{ iff } t = S \text{ or } t = r.$$

For environments $\tau = ((t_1, \ldots, t_a), t)$ and $\rho = ((r_1, \ldots, r_a), r)$ we have

$$\tau \sqsubseteq \rho \text{ iff } t_i \sqsubseteq r_i \text{ for all } i = 1, \ldots, a \text{ and } t \sqsubseteq r.$$

Then, the least-upper-bound operations for these types are as follows. For binding times t and r we have:

$$t \sqcup r = \begin{cases} t & \text{if } r \sqsubseteq t \\ r & \text{otherwise} \end{cases}$$

For environments τ and ρ as above we have

$$\tau \sqcup \rho = ((s_1, \ldots, s_a), s),$$

where $s_i = t_i \sqcup r_i$ and $s = t \sqcup r$.

Now we define the new analysis functions. The function $\mathcal{B}_e[\![e]\!]$ `div h` gives the binding time of expression e (an expression that occurrs in function `h`) with respect to division `div`.

$$\mathcal{B}_e[\![\mathtt{e}]\!] : \text{Monodivision} \to \text{FuncName} \to \text{BindingTime}$$

The main change we make with respect to the definition of Jones et al. [1, §5.2] is in the `call` rule: a call is dynamic if any of its actual parameters is dynamic or the binding time for the function result is dynamic.

$$\begin{array}{lcl}
\mathcal{B}_e[\![\mathtt{c}]\!]\ \mathtt{div\ h} & = & S \\
\mathcal{B}_e[\![\mathtt{x}_j]\!]\ \mathtt{div\ h} & = & t_j \text{ where } \mathtt{div\ h} = ((t_1, \ldots, t_a), t) \\
\mathcal{B}_e[\![(\mathtt{if\ e_1\ e_2\ e_3})]\!]\ \mathtt{div\ h} & = & \mathcal{B}_e[\![\mathtt{e_1}]\!]\ \mathtt{div\ h} \sqcup \mathcal{B}_e[\![\mathtt{e_2}]\!]\ \mathtt{div\ h} \sqcup \mathcal{B}_e[\![\mathtt{e_3}]\!]\ \mathtt{div\ h} \\
\mathcal{B}_e[\![(\mathtt{call\ f\ e_1\ \ldots\ e_a})]\!]\ \mathtt{div\ h} & = & (\bigsqcup_{j=1}^{a} \mathcal{B}_e[\![\mathtt{e_j}]\!]\ \mathtt{div\ h}) \sqcup t \text{ where } (\mathtt{div\ f}) = (_, t) \\
\mathcal{B}_e[\![(\mathtt{op\ e_1\ \ldots\ e_a})]\!]\ \mathtt{div\ h} & = & \bigsqcup_{j=1}^{a} \mathcal{B}_e[\![\mathtt{e_j}]\!]\ \mathtt{div\ h}
\end{array}$$

Next, we define $\mathcal{B}_v[\![\mathtt{e}]\!]$ `div h g` which collects the binding-time descriptions for function `g` in expression `e`, where `e` occurs in `h`.

$$\mathcal{B}_v[\![\mathtt{e}]\!] : \text{Monodivision} \to \text{FuncName} \to \text{FuncName} \to \text{BTEnv}$$

Again, the main change we make is in the `call` rule. The notation $(\bigsqcup u)$ is shorthand for $\bigsqcup_{j=1}^{a} u_j$, where $u = (u_1, \ldots, u_a)$. $(p \uparrow 2)$ selects the second element of a pair p.

$$\begin{array}{lcl}
\mathcal{B}_v[\![\mathtt{c}]\!]\ \mathtt{div\ h\ g} & = & ((S, \ldots, S), S) \\
\mathcal{B}_v[\![\mathtt{v}]\!]\ \mathtt{div\ h\ g} & = & ((S, \ldots, S), S) \\
\mathcal{B}_v[\![(\mathtt{if\ e_1\ e_2\ e_3})]\!]\ \mathtt{div\ h\ g} & = & \mathcal{B}_v[\![\mathtt{e_1}]\!]\ \mathtt{div\ h\ g} \sqcup \mathcal{B}_v[\![\mathtt{e_2}]\!]\ \mathtt{div\ h\ g} \sqcup \mathcal{B}_v[\![\mathtt{e_3}]\!]\ \mathtt{div\ h\ g} \\
\mathcal{B}_v[\![(\mathtt{call\ f\ e_1\ \ldots\ e_a})]\!]\ \mathtt{div\ h\ g} & = & \begin{cases} (t, s) & \text{if } \mathtt{f} \neq \mathtt{g} \\ (t, s) \sqcup (u, ((\mathtt{div}\ g) \uparrow 2) \sqcup (\bigsqcup u)) & \text{if } \mathtt{f} = \mathtt{g} \end{cases} \\
 & & \text{where } \begin{cases} (t, s) & = & \bigsqcup_{j=1}^{a} \mathcal{B}_v[\![\mathtt{e_j}]\!]\ \mathtt{div\ h\ g} \\ u & = & (\mathcal{B}_e[\![\mathtt{e_1}]\!]\ \mathtt{div\ h}, \ldots, \mathcal{B}_e[\![\mathtt{e_a}]\!]\ \mathtt{div\ h}) \end{cases} \\
\mathcal{B}_v[\![(\mathtt{op\ e_1\ \ldots\ e_a})]\!] & = & \bigsqcup_{j=1}^{a} \mathcal{B}_v[\![e_j]\!]\ \mathtt{div\ h\ g}
\end{array}$$

Finally, we define the equation for `div` as follows:

$$\mathtt{div\ g} = \begin{cases} (\bigsqcup_{i=1}^{n} \mathcal{B}_v[\![\mathtt{e_{f_i}}]\!]\ \mathtt{div\ f}_i\ \mathtt{g}) \sqcup (\mathtt{goal\text{-}args}, D) & \text{if g is the goal function} \\ (\bigsqcup_{i=1}^{n} \mathcal{B}_v[\![\mathtt{e_{f_i}}]\!]\ \mathtt{div\ f}_i\ \mathtt{g}) \sqcup ((S, \ldots, S), \mathcal{B}_e[\![\mathtt{e_g}]\!]\ \mathtt{div\ g}) & \text{otherwise} \end{cases}$$

where $\texttt{e}_\texttt{f}$ denotes the body of function $\texttt{e}_\texttt{f}$ and `goal-args` denotes the binding-time pattern of the goal function's static arguments. The terms of the form $((S, \ldots, S), \mathcal{B}_e[\![\texttt{e}_\texttt{g}]\!]\ \texttt{div g})$ for each non-goal function `g` allow the binding time of `g`'s return value to account for the binding time of the body of `g`.

Note that if in the least-fixed-point solution for `div` we have $\texttt{div goal} \sqsupset (\texttt{goal-args}, D)$, then some of the arguments of the goal function that are intended to be static will have to be supplied as dynamic arguments.

References

[1] N.D. Jones, C.K. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice-Hall International, 1993.