Implementation Notes for the Termination-Guaranteeing Binding-Time Analysis

Arne John Glenstrup

December 13, 2002

Abstract

This note describes some of the implementation aspects of the termination analysis for partial evaluation described by Glenstrup and Jones (2003).

1 Algorithm overview

Figure 1 gives an overview of the modules that constitute the termination analysis for partial evaluation. Note that the call graph is only needed for efficiency. When bounded anchoring has determined a set of specialisation points, the standard BTA and bounded anchoring must be reiterated because a static function call may have become dynamic due to the specialisation point. Thus more variables may need to be reclassified from static to dynamic, which in turn may invalidate anchors and cause bounded anchoring to make more variables dynamic.

2 Algorithm modules

In the following sections we will briefly describe all the modules of the algorithm except the standard BTA which can be found in the literature (Jones et al., 1993; Christensen et al., 2000; Thiemann, 1997, 1999; Henglein, 1991).

2.1 Call graph generator

This module generates a graph with p's functions as nodes and an edge $f \longrightarrow g$ whenever the body of f contains a function call to g. Using an algorithm for grouping graph nodes into strongly connected components (SCCs) and sorting the resulting SCC DAG topologically (Cormen et al., 1990), it should be possible to implement SCC generation and sorting in time $O(f+s)$ where f is the number of functions and s the number of call sites in p.

2.2 Decreasing size approximation generator $\mathcal{E}^{\downarrow}$

The decreasing expression size approximation operator $\mathcal{E}^{\downarrow}$ is shown in Figure 2. The first argument, to $\mathcal{E}^{\downarrow}_{\mathbf{e}}$ is a fixpoint ϕ computed by applying a fixpoint operator *fix* to a function F such that $F\ \phi = \phi$.

In general a fixpoint operator can be implemented by

$$\begin{array}{lcl} \mathit{fix}\ F & = & \mathit{fix}'\ F\ \bot_\phi \\ \mathit{fix}'\ F\ \phi & = & \mathbf{if}\ \phi = F\ \phi\ \mathbf{then}\ \phi\ \mathbf{else}\ \mathit{fix}'\ F\ (F\ \phi) \end{array}$$

where $\bot_\phi$ is the bottom element of the domain of F.

The problem is how to compute the expression $\phi = F\ \phi$ because both sides of the equation are functions.

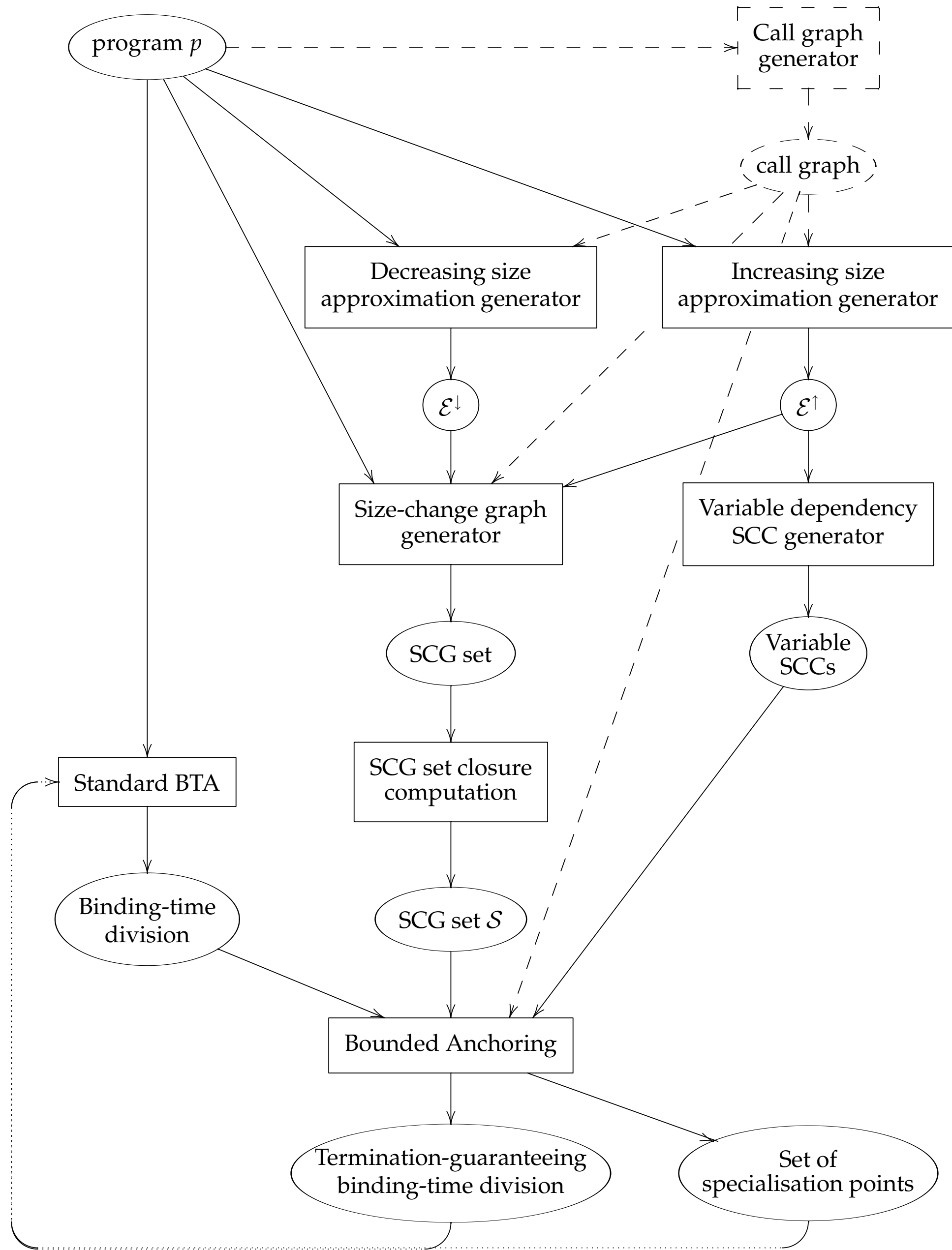

Figure 1: Module interdependencies. Dashed lines are only needed for efficiency, dotted lines indicate iteration feedback.

$$
\begin{aligned}
\mathcal{E}^{\downarrow} = \mathcal{E}^{\downarrow}_{\mathbf{e}}\ (\mathit{fix}\ (\lambda\phi.\{ & f1 \mapsto \lambda\Delta_1 \ldots \Delta_m . \mathcal{E}^{\downarrow}_{\mathbf{e}}\ \phi\ [\![e^{f1}]\!]\ \{x_1 \mapsto \Delta_1, \ldots, x_m \mapsto \Delta_m\}, \ldots, \\
& fn \mapsto \lambda\Delta_1 \ldots \Delta_k . \mathcal{E}^{\downarrow}_{\mathbf{e}}\ \phi\ [\![e^{fn}]\!]\ \{x_1 \mapsto \Delta_1, \ldots, x_k \mapsto \Delta_k\}\}))\ \rho^{\downarrow}_{\mathbf{id}}
\end{aligned}
$$

$$
\rho^{\downarrow}_{\mathbf{id}}\ x = \{\overline{\downarrow}(x)\}
$$

$$
\begin{aligned}
\mathcal{E}^{\downarrow}_{\mathbf{e}}\ \phi\ [\![k]\!]\ \rho &= \{\} \\
\mathcal{E}^{\downarrow}_{\mathbf{e}}\ \phi\ [\![x]\!]\ \rho &= \rho\ x \\
\mathcal{E}^{\downarrow}_{\mathbf{e}}\ \phi\ [\![\mathbf{let}\ x = e_1\ \mathbf{in}\ e_2]\!]\ \rho &= \mathcal{E}^{\downarrow}_{\mathbf{e}}\ \phi\ [\![e_2]\!]\ (\rho + \{x \mapsto \mathcal{E}^{\downarrow}_{\mathbf{e}}\ \phi\ [\![e_1]\!]\ \rho\}) \\
\mathcal{E}^{\downarrow}_{\mathbf{e}}\ \phi\ [\![\mathbf{if}\ e_1\ \mathbf{then}\ e_2\ \mathbf{else}\ e_3]\!]\ \rho &= \mathcal{E}^{\downarrow}_{\mathbf{e}}\ \phi\ [\![e_2]\!]\ \rho\ \sqcap^{\downarrow}\ \mathcal{E}^{\downarrow}_{\mathbf{e}}\ \phi\ [\![e_3]\!]\ \rho \\
\mathcal{E}^{\downarrow}_{\mathbf{e}}\ \phi\ [\![\texttt{cons}\ e_1\ e_2]\!]\ \rho &= \{\} \\
\mathcal{E}^{\downarrow}_{\mathbf{e}}\ \phi\ [\![\texttt{car}\ e_1]\!]\ \rho &= \{\downarrow(x) \mid \delta(x) \in \mathcal{E}^{\downarrow}_{\mathbf{e}}\ \phi\ [\![e_1]\!]\ \rho\} \\
\mathcal{E}^{\downarrow}_{\mathbf{e}}\ \phi\ [\![b\ e_1 \ldots e_n]\!]\ \rho &= \{\} \\
\mathcal{E}^{\downarrow}_{\mathbf{e}}\ \phi\ [\![f\ e_1 \ldots e_n]\!]\ \rho &= \phi\ f\ (\mathcal{E}^{\downarrow}_{\mathbf{e}}\ \phi\ [\![e_1]\!]\ \rho) \ldots (\mathcal{E}^{\downarrow}_{\mathbf{e}}\ \phi\ [\![e_n]\!]\ \rho)
\end{aligned}
$$

$$
\begin{aligned}
\text{where}\quad \Delta_1 \sqcap^{\downarrow} \Delta_2 = & \{\overline{\downarrow}(x) \mid \overline{\downarrow}(x) \in \Delta_1 \wedge \downarrow(x) \in \Delta_2\} \cup \\
& \{\overline{\downarrow}(x) \mid \overline{\downarrow}(x) \in \Delta_2 \wedge \downarrow(x) \in \Delta_1\} \cup (\Delta_1 \cap \Delta_2)
\end{aligned}
$$

Figure 2: Operator approximating decreasing size information for the value of an expression. `cdr` is treated like `car`, and symbol 'b' stands for any builtin function that is not given special treatment.

In our case we know that applying first a function name f and then n dependency sets (where n is the arity of f), we obtain a ground value (a dependency set). So in our case we can implement the fixpoint operator like this:

$$
\begin{aligned}
\mathit{fix}\ F &= \mathit{fix}'\ F\ (\lambda f.\lambda\Delta_1 \ldots \lambda\Delta_n.\{\downarrow(x) \mid x \text{ is a variable}\}) \\
\mathit{fix}'\ F\ \phi &= \mathbf{if}\ \forall f : \phi\ f\ \{\overline{\downarrow}(x_1)\} \ldots \{\overline{\downarrow}(x_n)\} = (F\ \phi)\ f\ \{\overline{\downarrow}(x_1)\} \ldots \{\overline{\downarrow}(x_n)\}\ \mathbf{then}\ \phi\ \mathbf{else}\ \mathit{fix}'\ F\ (F\ \phi)
\end{aligned}
$$

The fixpoint computation can in some cases be made more efficient by considering the functions of the '$\forall f$' construct in reverse topological order of the call graph.

2.3 Increasing size approximation generator $\mathcal{E}^{\uparrow}$

The increasing expression size approximation operator $\mathcal{E}^{\uparrow}$ is shown in Figure 3. It can be implemented using a fixpoint operator of this style:

$$
\begin{aligned}
\mathit{fix}\ F &= \mathit{fix}'\ F\ (\lambda f.\lambda\Delta_1 \ldots \lambda\Delta_n.\{\}) \\
\mathit{fix}'\ F\ \phi &= \mathbf{if}\ \forall f : \quad \phi\ f\ \bullet\ \{\}\ \{\}\ \underline{\updownarrow}\ \{\underline{\updownarrow}(x_1)\} \ldots \{\underline{\updownarrow}(x_n)\} \\
&\qquad = (F\ \phi)\ f\ \bullet\ \{\}\ \{\}\ \underline{\updownarrow}\ \{\underline{\updownarrow}(x_1)\} \ldots \{\underline{\updownarrow}(x_n)\}\ \mathbf{then}\ \phi\ \mathbf{else}\ \mathit{fix}'\ F\ (F\ \phi)
\end{aligned}
$$

A call to the helper operator $\mathcal{E}^{\uparrow}_{\mathbf{e}}\ \phi\ [\![e]\!]\ g\ F\ X\ \delta\ \rho$ returns a triple (Δ', X', δ') such that

- Δ' is the size dependency set for e
- X' is the set of free variables encountered in **if**-tests during recursive descent and unfolding of e to any call to g
- δ' is '$\uparrow$' if there is an increasing operation (i.e., a `cons`) during recursive descent and unfolding of e to any call to g, else it is '$\underline{\updownarrow}$.'

The arguments F, X and δ are accumulating parameters, where

- F is the set of names of functions "on the call stack"
- X is the set of free variables encountered in **if**-tests
- δ is '$\uparrow$' if an increasing operation (i.e., a `cons`) has been encountered, else it is '$\underline{\updownarrow}$'

$$
\begin{aligned}
&\mathcal{E}^{\uparrow}[\![e]\!] = \Delta,\\
&\text{where } (\Delta, X, \delta) \;=\; \mathcal{E}^{\uparrow}_{\mathbf{e}} \Big(\mathit{fix} \Big(\lambda\phi. \Big\{ f1 \mapsto \lambda g F X \delta \Delta_1 \ldots \Delta_m . \mathcal{E}^{\uparrow}_{\mathbf{e}}\ \phi\ [\![e^{f1}]\!]\ g\ F\ X\ \delta\ \{x_1 \mapsto \Delta_1, \ldots, x_m \mapsto \Delta_m\}, \ldots,\\
&\qquad\qquad fn \mapsto \lambda g F X \delta \Delta_1 \ldots \Delta_k . \mathcal{E}^{\uparrow}_{\mathbf{e}}\ \phi\ [\![e^{fn}]\!]\ g\ F\ X\ \delta\ \{x_1 \mapsto \Delta_1, \ldots, x_k \mapsto \Delta_k\} \Big\}\Big)\Big)\\
&\qquad\qquad [\![e]\!]\ \bullet\ \{\}\ \{\}\ \ddagger\ \rho^{\uparrow}_{\mathbf{id}}\\
&\rho^{\uparrow}_{\mathbf{id}}\ x \;=\; \{\ddagger(x)\}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{E}^{\uparrow}_{\mathbf{e}}\ \phi\ [\![k]\!]\ g\ F\ X\ \delta\ \rho \;&=\; (\{\}, \{\}, \ddagger)\\
\mathcal{E}^{\uparrow}_{\mathbf{e}}\ \phi\ [\![x]\!]\ g\ F\ X\ \delta\ \rho \;&=\; (\rho\ x, \{\}, \ddagger)\\
\mathcal{E}^{\uparrow}_{\mathbf{e}}\ \phi\ [\![\mathbf{let}\ x = e_1\ \mathbf{in}\ e_2]\!]\ g\ F\ X\ \delta\ \rho \;&=\; (\Delta_2', X_1 \cup X_2, \delta_1 \sqcup^{\uparrow} \delta_2)\\
&\quad \text{where } (\Delta_1', X_1, \delta_1) = \mathcal{E}^{\uparrow}_{\mathbf{e}}\ \phi\ [\![e_1]\!]\ g\ F\ X\ \delta\ \rho\\
&\quad \rho' = \rho + \{x \mapsto \Delta_1'\}\\
&\quad (\Delta_2', X_2, \delta_2) = \mathcal{E}^{\uparrow}_{\mathbf{e}}\ \phi\ [\![e_2]\!]\ g\ F\ X\ \delta\ \rho'\\
\mathcal{E}^{\uparrow}_{\mathbf{e}}\ \phi\ [\![\mathbf{if}\ e_1\ \mathbf{then}\ e_2\ \mathbf{else}\ e_3]\!]\ g\ F\ X\ \delta\ \rho \;&=\; (\Delta_2' \sqcup^{\uparrow} \Delta_3', X_1 \cup X_2 \cup X_3, \delta_1 \sqcup^{\uparrow} \delta_2 \sqcup^{\uparrow} \delta_3),\\
&\quad \text{where } (\Delta_1', X_1, \delta_1) = \mathcal{E}^{\uparrow}_{\mathbf{e}}\ \phi\ [\![e_1]\!]\ g\ F\ X\ \delta\ \rho\\
&\quad (\Delta_i', X_i, \delta_i) = \mathcal{E}^{\uparrow}_{\mathbf{e}}\ \phi\ [\![e_i]\!]\ g\ F\ (X' \cup X)\ \delta\ \rho, i > 1\\
&\quad X' = \{x \mid (\delta(x) \in \rho\ x_i) \wedge x_i \in \mathit{fv}\ e_1\}\\
\mathcal{E}^{\uparrow}_{\mathbf{e}}\ \phi\ [\![\texttt{cons}\ e_1\ e_2]\!]\ g\ F\ X\ \delta\ \rho \;&=\; (\{\uparrow(x) \mid \delta(x) \in \Delta_1' \cup \Delta_2'\}, X_1 \cup X_2, \delta_1 \sqcup^{\uparrow} \delta_2)\\
&\quad \text{where } (\Delta_i', X_i, \delta_i) = \mathcal{E}^{\uparrow}_{\mathbf{e}}\ \phi\ [\![e_i]\!]\ g\ F\ X \uparrow \rho\\
\mathcal{E}^{\uparrow}_{\mathbf{e}}\ \phi\ [\![\texttt{car}\ e_1]\!]\ g\ F\ X\ \delta\ \rho \;&=\; \mathcal{E}^{\uparrow}_{\mathbf{e}}\ \phi\ [\![e_1]\!]\ g\ F\ X\ \delta\ \rho\\
\mathcal{E}^{\uparrow}_{\mathbf{e}}\ \phi\ [\![b\ e_1 \ldots e_n]\!]\ g\ F\ X\ \delta\ \rho \;&=\; (\{\uparrow(x) \mid \delta(x) \in \Delta_1' \cup \cdots \cup \Delta_n'\}, X_1 \cup \cdots \cup X_n, \delta_1 \sqcup^{\uparrow} \cdots \sqcup^{\uparrow} \delta_n)\\
&\quad \text{where } (\Delta_i', X_i, \delta_i) = \mathcal{E}^{\uparrow}_{\mathbf{e}}\ \phi\ [\![e_i]\!]\ g\ F\ X \uparrow \rho\\
\mathcal{E}^{\uparrow}_{\mathbf{e}}\ \phi\ [\![f\ e_1 \ldots e_n]\!]\ g\ F\ X\ \delta\ \rho \;&=\; (\Delta', X' \cup X_1 \cup \cdots \cup X_n, \delta' \sqcup^{\uparrow} \delta_1 \sqcup^{\uparrow} \cdots \sqcup^{\uparrow} \delta_n)
\end{aligned}
$$

$$
\begin{aligned}
\text{where } (\Delta_i', X_i, \delta_i) &= \mathcal{E}^{\uparrow}_{\mathbf{e}}\ \phi\ [\![e_i]\!]\ g\ F\ X\ \delta\ \rho && \text{(analyse arguments)}\\
(\Delta'', X'', \delta'') &= \phi\ f\ g\ (F \cup \{f\})\ X\ \delta\ (\Delta_1', \ldots, \Delta_n') && \text{(analyse function call)}\\
(X', \delta') &= \text{if } f = g \text{ then } (X, \delta) \text{ else } (X'', \delta'') && \text{(return recursive increase info)}\\
(\Delta^f, X^f, \delta^f) &= \phi\ f\ f\ \{\}\ \{\}\ \ddagger\ (\{\ddagger(x_1)\}, \ldots, \{\ddagger(x_n)\}) && \text{(compute recursive increase information for } f\text{)}\\
\Delta' &= \text{if } f \in F \wedge (\delta^f = \uparrow \vee \exists x : \uparrow(x) \in \Delta^f) && \text{(if } f \text{ is recursive and has}\\
&\quad \text{then } \Delta'' \cup \{\uparrow(x) \mid \delta(x) \in \Delta_i' \wedge x_i \in X^f\} && \text{recursive increase, then add}\\
&\quad \text{else } \Delta'' && \text{free variables from } \textbf{if}\text{-tests)}
\end{aligned}
$$

$$
\begin{aligned}
\text{where } \Delta_1 \sqcup^{\uparrow} \Delta_2 &= \{\uparrow(x) \mid \uparrow(x) \in \Delta_1 \cup \Delta_2\} \cup\\
&\quad \{\ddagger(x) \mid \ddagger(x) \in \Delta_1 \cup \Delta_2 \wedge \uparrow(x) \notin \Delta_1 \cup \Delta_2\}\\
\delta_1 \sqcup^{\uparrow} \delta_2 &= \text{if } \delta_1 = \uparrow \vee \delta_2 = \uparrow \text{ then } \uparrow \text{ else } \ddagger
\end{aligned}
$$

Figure 3: Operator for approximating increasing size information for the value of an expression. `cdr` is treated like `car`. Symbol 'b' stands for any builtin function that is not given special treatment, and '•' is a symbol distinct from all function names.

Note, however, that the operator in Figure 3 is equivalent to the two mutually recursive operators $\mathcal{E}^{\uparrow}$ and $\mathcal{X}$ shown in Figures 4 and 5 that might be more efficiently implementable.

$$
\begin{array}{l}
\mathcal{E}^{\uparrow}\,[\![e]\!]\,\rho = \Delta, \\
\text{where } \Delta \;=\; \mathcal{E}^{\uparrow}_{\mathbf{e}}\,\Big(\mathit{fix}\,\Big(\lambda\phi.\Big\{f1 \mapsto \lambda\Delta_1\ldots\Delta_m.\,\mathcal{E}^{\uparrow}_{\mathbf{e}}\,\phi\,[\![e^{f1}]\!]\,\{x_1 \mapsto \Delta_1,\ldots,x_m \mapsto \Delta_m\},\ldots, \\
\qquad\qquad fn \mapsto \lambda\Delta_1\ldots\Delta_k\,.\,\mathcal{E}^{\uparrow}_{\mathbf{e}}\,\phi\,[\![e^{fn}]\!]\,\{x_1 \mapsto \Delta_1,\ldots,x_k \mapsto \Delta_k\}\Big\}\Big)\Big)\,[\![e]\!]\,\rho
\end{array}
$$

$$
\begin{array}{lcl}
\mathcal{E}^{\uparrow}_{\mathbf{e}}\,\phi\,[\![k]\!]\,\rho & = & \{\} \\
\mathcal{E}^{\uparrow}_{\mathbf{e}}\,\phi\,[\![x]\!]\,\rho & = & \rho\,x \\
\mathcal{E}^{\uparrow}_{\mathbf{e}}\,\phi\,[\![\mathbf{let}\ x = e_1\ \mathbf{in}\ e_2]\!]\,\rho & = & \Delta'_2 \\
 & & \text{where } \Delta'_1 = \mathcal{E}^{\uparrow}_{\mathbf{e}}\,\phi\,[\![e_1]\!]\,\rho \\
 & & \qquad\ \ \rho' = \rho + \{x \mapsto \Delta'_1\} \\
 & & \qquad\ \Delta'_2 = \mathcal{E}^{\uparrow}_{\mathbf{e}}\,\phi\,[\![e_2]\!]\,\rho' \\
\mathcal{E}^{\uparrow}_{\mathbf{e}}\,\phi\,[\![\mathbf{if}\ e_1\ \mathbf{then}\ e_2\ \mathbf{else}\ e_3]\!]\,\rho & = & \Delta'_2 \sqcup^{\uparrow} \Delta'_3 \\
 & & \text{where } \Delta'_i = \mathcal{E}^{\uparrow}_{\mathbf{e}}\,\phi\,[\![e_i]\!]\,\rho, i > 1 \\
\mathcal{E}^{\uparrow}_{\mathbf{e}}\,\phi\,[\![\mathtt{cons}\ e_1\ e_2]\!]\,\rho & = & \{\uparrow(x) \mid \delta(x) \in \Delta'_1 \cup \Delta'_2\} \\
 & & \text{where } \Delta'_i = \mathcal{E}^{\uparrow}_{\mathbf{e}}\,\phi\,[\![e_i]\!]\,\rho \\
\mathcal{E}^{\uparrow}_{\mathbf{e}}\,\phi\,[\![\mathtt{car}\ e_1]\!]\,\rho & = & \mathcal{E}^{\uparrow}_{\mathbf{e}}\,\phi\,[\![e_1]\!]\,\rho \\
\mathcal{E}^{\uparrow}_{\mathbf{e}}\,\phi\,[\![b\ e_1\ldots e_n]\!]\,\rho & = & \{\uparrow(x) \mid \delta(x) \in \Delta'_1 \cup \cdots \cup \Delta'_n\} \\
 & & \text{where } \Delta'_i = \mathcal{E}^{\uparrow}_{\mathbf{e}}\,\phi\,[\![e_i]\!]\,\rho \\
\mathcal{E}^{\uparrow}_{\mathbf{e}}\,\phi\,[\![f\ e_1\ldots e_n]\!]\,\rho & = & \Delta'
\end{array}
$$

$$
\begin{array}{rcll}
\text{where}\quad \Delta'_i & = & \mathcal{E}^{\uparrow}_{\mathbf{e}}\,\phi\,[\![e_i]\!]\,\rho & \text{(analyse arguments)} \\
\Delta'' & = & \phi\,f\,\Delta'_1\ldots\Delta'_n & \text{(analyse function call)} \\
\Delta^f & = & \phi\,f\,\{\updownarrow(x_1)\}\ldots\{\updownarrow(x_n)\} & \text{(compute recursive increase information for } f\text{)} \\
(X^f, \delta^f) & = & \mathcal{X}\,[\![f]\!] & \text{(compute } \mathbf{if}\text{-test variables and recursive increase information for } f\text{)} \\
\Delta' & = & \text{if } f \text{ is recursive} & \text{(if } f \text{ is recursive and has} \\
 & & \text{and } (\delta^f = \uparrow \vee \exists x : \uparrow(x) \in \Delta^f) & \text{recursive increase, then add} \\
 & & \text{then } \Delta'' \cup \{\uparrow(x) \mid \delta(x) \in \Delta'_i \wedge x_i \in X^f\} & \text{dependencies on free variables} \\
 & & \text{else } \Delta'' & \text{from } \mathbf{if}\text{-tests)}
\end{array}
$$

$$
\begin{array}{rcl}
\text{where}\quad \Delta_1 \sqcup^{\uparrow} \Delta_2 & = & \{\uparrow(x) \mid \uparrow(x) \in \Delta_1 \cup \Delta_2\} \cup \\
 & & \{\updownarrow(x) \mid \updownarrow(x) \in \Delta_1 \cup \Delta_2 \wedge \uparrow(x) \notin \Delta_1 \cup \Delta_2\} \\
\delta_1 \sqcup^{\uparrow} \delta_2 & = & \text{if } \delta_1 = \uparrow \vee \delta_2 = \uparrow \text{ then } \uparrow \text{ else } \updownarrow
\end{array}
$$

Figure 4: Operator for approximating increasing size information for the value of an expression. `cdr` is treated like `car`. Symbol 'b' stands for any builtin function that is not given special treatment.

The operator $\mathcal{X}$ of Figure 5 collects the free variables of the **if**-tests in (possibly mutually) recursive loops, and also detects whether there are any loops in the function in which there is an increase. For instance, we expect $\mathcal{X}\,[\![f]\!] = (\{x\}, \updownarrow)$ and $\mathcal{X}\,[\![g]\!] = (\{x\}, \uparrow)$ and $\mathcal{X}\,[\![h]\!] = (\{x, z\}, \updownarrow)$ for these functions:

$$
\begin{array}{lcl}
f\,x\,y & = & \mathbf{if}\ x = [\,]\ \mathbf{then}\ y\ \mathbf{else}\ f\,(\mathtt{cdr}\ x)\,(\mathtt{cons}\ 1\ y) \\
g\,x\,y & = & \mathbf{if}\ x = [\,]\ \mathbf{then}\ (\mathbf{if}\ y = [\,]\ \mathbf{then}\ [\,]\ \mathbf{else}\ [1])\ \mathbf{else}\ \mathtt{cons}\ 1\,(g\,(\mathtt{cdr}\ x)) \\
h\,x\,y\,z & = & \mathbf{if}\ x = z\ \mathbf{then}\ y\ \mathbf{else}\ h\,x\,(\mathtt{cons}\ 1\ y)\,(\mathtt{cons}\ 1\ z)
\end{array}
$$

Note that although there is no expression "`cons` x" increasing x (in fact, in h the value of x is not changed at all) the size of the return value of each of the functions is greater than or equal to the size of x (and y

$$
\begin{aligned}
&\mathcal{X}[\![f]\!] = (X, \delta)\\
&\text{where } (X,\delta) \;=\; \mathcal{X}_{\mathbf{e}} \left(\mathit{fix} \left(\lambda\phi. \Big\{ f1 \mapsto \lambda g X \delta \Delta_1 \ldots \Delta_m . \, \mathcal{X}_{\mathbf{e}} \; \phi \; [\![e^{f1}]\!] \; g \; X \; \delta \; \{x_1 \mapsto \Delta_1, \ldots, x_m \mapsto \Delta_m\}, \ldots, \right.\right.\\
&\qquad\qquad \left.\left. fn \mapsto \lambda g X \delta \Delta_1 \ldots \Delta_k . \, \mathcal{X}_{\mathbf{e}} \; \phi \; [\![e^{fn}]\!] \; g \; X \; \delta \; \{x_1 \mapsto \Delta_1, \ldots, x_k \mapsto \Delta_k\} \Big\} \right) \right)\\
&\qquad\qquad [\![e^f]\!] \; f \; \{\} \; \updownarrow \; \rho^{\uparrow}_{\mathbf{id}}\\
&\rho^{\uparrow}_{\mathbf{id}} \; x \;=\; \{\updownarrow(x)\}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{X}_{\mathbf{e}} \; \phi \; [\![k]\!] \; g \; X \; \delta \; \rho \;&=\; (\{\}, \updownarrow)\\
\mathcal{X}_{\mathbf{e}} \; \phi \; [\![x]\!] \; g \; X \; \delta \; \rho \;&=\; (\{\}, \updownarrow)\\
\mathcal{X}_{\mathbf{e}} \; \phi \; [\![\mathbf{let} \; x = e_1 \; \mathbf{in} \; e_2]\!] \; g \; X \; \delta \; \rho \;&=\; (X_1 \cup X_2, \delta_1 \sqcup^{\uparrow} \delta_2)\\
&\quad \text{where } (X_1, \delta_1) = \mathcal{X}_{\mathbf{e}} \; \phi \; [\![e_1]\!] \; g \; X \; \delta \; \rho\\
&\qquad\qquad \rho' = \rho + \{x \mapsto \mathcal{E} \, [\![e_1]\!] \; \rho\}\\
&\qquad\qquad (X_2, \delta_2) = \mathcal{X}_{\mathbf{e}} \; \phi \; [\![e_2]\!] \; g \; X \; \delta \; \rho\\
\mathcal{X}_{\mathbf{e}} \; \phi \; [\![\mathbf{if} \; e_1 \; \mathbf{then} \; e_2 \; \mathbf{else} \; e_3]\!] \; g \; X \; \delta \; \rho \;&=\; (X_1 \cup X_2 \cup X_3, \delta_1 \sqcup^{\uparrow} \delta_2 \sqcup^{\uparrow} \delta_3),\\
&\quad \text{where } (X_1, \delta_1) = \mathcal{X}_{\mathbf{e}} \; \phi \; [\![e_1]\!] \; g \; X \; \delta \; \rho\\
&\qquad\qquad (X_i, \delta_i) = \mathcal{X}_{\mathbf{e}} \; \phi \; [\![e_i]\!] \; g \; (X' \cup X) \; \delta \; \rho, i > 1\\
&\qquad\qquad X' = \{x \mid (\delta(x) \in \rho \; x_i) \wedge x_i \in \mathit{fv} \; e_1\}\\
\mathcal{X}_{\mathbf{e}} \; \phi \; [\![\texttt{cons} \; e_1 \; e_2]\!] \; g \; X \; \delta \; \rho \;&=\; (X_1 \cup X_2, \delta_1 \sqcup^{\uparrow} \delta_2)\\
&\quad \text{where } (X_i, \delta_i) = \mathcal{X}_{\mathbf{e}} \; \phi \; [\![e_i]\!] \; g \; X \uparrow\\
\mathcal{X}_{\mathbf{e}} \; \phi \; [\![\texttt{car} \; e_1]\!] \; g \; X \; \delta \; \rho \;&=\; \mathcal{X}_{\mathbf{e}} \; \phi \; [\![e_1]\!] \; g \; X \; \delta \; \rho\\
\mathcal{X}_{\mathbf{e}} \; \phi \; [\![b \; e_1 \ldots e_n]\!] \; g \; X \; \delta \; \rho \;&=\; (X_1 \cup \cdots \cup X_n, \delta_1 \sqcup^{\uparrow} \cdots \sqcup^{\uparrow} \delta_n)\\
&\quad \text{where } (X_i, \delta_i) = \mathcal{X}_{\mathbf{e}} \; \phi \; [\![e_i]\!] \; g \; X \uparrow\\
\mathcal{X}_{\mathbf{e}} \; \phi \; [\![f \; e_1 \ldots e_n]\!] \; g \; X \; \delta \; \rho \;&=\; (X' \cup X_1 \cup \cdots \cup X_n, \delta' \sqcup^{\uparrow} \delta_1 \sqcup^{\uparrow} \cdots \sqcup^{\uparrow} \delta_n)\\
\text{where } (X_i, \delta_i) &= \mathcal{X}_{\mathbf{e}} \, \phi \; [\![e_i]\!] \; g \; X \; \delta \; \rho \quad \text{(analyse arguments)}\\
(X'', \delta'') &= \phi \; f \; g \; X \; \delta \; (\mathcal{E} \, [\![e_1]\!] \; \rho) \ldots (\mathcal{E} \, [\![e_n]\!] \; \rho) \quad \text{(analyse function call)}\\
(X', \delta') &= \text{if } f = g \text{ then } (X, \delta) \text{ else } (X'', \delta'') \quad \text{(return recursive increase info)}
\end{aligned}
$$

Figure 5: Operator for computing whether a function computes an increasing operation (e.g., `cons`) in a recursive loop, and for computing the set of free variables in the **if**-tests in recursive loops. `cdr` is treated like `car`. Symbol 'b' stands for any builtin function that is not given special treatment.

in the case of f and h). Note also that we also expect $\mathcal{X}\ [\![g]\!] = (\{x\}, \uparrow)$ for

$$g\ x\ y \quad = \quad \textbf{if}\ x = [\,]\ \textbf{then}\ (\textbf{if}\ y = [\,]\ \textbf{then}\ [\,]\ \textbf{else}\ [1])\ \textbf{else}\ \texttt{cons}\ 1\ (g\ (\texttt{cdr}\ x)\ (\texttt{cons}\ 1\ y))$$

because y is not tested in the recursive loop.

2.4 Size-change graph generator

The operator $\mathcal{G} : \mathit{Program} \rightarrow \mathcal{P}(\mathit{SCG})$ for generating the set of size-change graphs $\mathcal{G}[\![p]\!]$ is shown in Figure 6. The reference to the call graph SCC is only for efficiency reasons: only SCGs for recursive and

$$
\begin{array}{lcl}
\mathcal{G}\ [\![d_1 \ldots d_n]\!] & = & \mathcal{G}_{\mathbf{d}}\ [\![d_1]\!] \cup \ldots \cup \mathcal{G}_{\mathbf{d}}\ [\![d_n]\!] \\
\mathcal{G}_{\mathbf{d}}\ [\![f\ x_1 \ldots x_n = e]\!] & = & \mathcal{G}_{\mathbf{e}}\ f\ [\![e]\!]\ \{x_i \mapsto \{\overline{\downarrow}(x_i)\}\}\ \{x_i \mapsto \{\updownarrow(x_i)\}\} \\
\mathcal{G}_{\mathbf{e}}\ g\ [\![k]\!]\ \rho^{\downarrow}\ \rho^{\uparrow} & = & \{\} \\
\mathcal{G}_{\mathbf{e}}\ g\ [\![x]\!]\ \rho^{\downarrow}\ \rho^{\uparrow} & = & \{\} \\
\mathcal{G}_{\mathbf{e}}\ g\ [\![\textbf{let}\ x = e_1\ \textbf{in}\ e_2]\!]\ \rho^{\downarrow}\ \rho^{\uparrow} & = & \mathcal{G}_{\mathbf{e}}\ g\ [\![e_1]\!]\ \rho^{\downarrow}\ \rho^{\uparrow} \cup \mathcal{G}_{\mathbf{e}}\ g\ [\![e_2]\!]\ \rho'^{\downarrow}\ \rho'^{\uparrow} \\
 & & \text{where}\ \ \rho'^{\downarrow} \ = \ \rho^{\downarrow} + \{x \mapsto \mathcal{E}\ g\ [\![e_1]\!]\ \rho^{\downarrow}\} \\
 & & \qquad\quad\ \rho'^{\uparrow} \ = \ \rho^{\uparrow} + \{x \mapsto \mathcal{E}\ g\ [\![e_1]\!]\ \rho^{\uparrow}\} \\
\mathcal{G}_{\mathbf{e}}\ g\ [\![\textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3]\!]\ \rho^{\downarrow}\ \rho^{\uparrow} & = & \mathcal{G}_{\mathbf{e}}\ g\ [\![e_1]\!]\ \rho^{\downarrow}\ \rho^{\uparrow} \cup \mathcal{G}_{\mathbf{e}}\ g\ [\![e_2]\!]\ \rho^{\downarrow}\ \rho^{\uparrow} \cup \mathcal{G}_{\mathbf{e}}\ g\ [\![e_3]\!]\ \rho^{\downarrow}\ \rho^{\uparrow} \\
\mathcal{G}_{\mathbf{e}}\ g\ [\![b\ e_1 \ldots e_n]\!]\ \rho^{\downarrow}\ \rho^{\uparrow} & = & \mathcal{G}_{\mathbf{e}}\ g\ [\![e_1]\!]\ \rho^{\downarrow}\ \rho^{\uparrow} \cup \ldots \cup \mathcal{G}_{\mathbf{e}}\ g\ [\![e_n]\!]\ \rho^{\downarrow}\ \rho^{\uparrow} \\
\mathcal{G}_{\mathbf{e}}\ g\ [\![c : f\ e_1 \ldots e_n]\!]\ \rho^{\downarrow}\ \rho^{\uparrow} & = & \gamma \cup \mathcal{G}_{\mathbf{e}}\ g\ [\![e_1]\!]\ \rho^{\downarrow}\ \rho^{\uparrow} \cup \ldots \cup \mathcal{G}_{\mathbf{e}}\ g\ [\![e_n]\!]\ \rho^{\downarrow}\ \rho^{\uparrow}
\end{array}
$$

$$
\begin{array}{ll}
\text{where}\ \gamma = & \text{if}\ f\ \text{and}\ g\ \text{are in the same call graph SCC} \\
 & \text{then}\ \{(\{x \xrightarrow{\delta} f_i \mid \delta(x) \in \mathcal{E}^{\downarrow}\ [\![e_i]\!]\ \rho^{\downarrow}\}, \{x \xrightarrow{\delta} f_i \mid \delta(x) \in \mathcal{E}^{\uparrow}\ [\![e_i]\!]\ \rho^{\uparrow}\}, \{c\})\} \\
 & \text{else}\ \{\}
\end{array}
$$

Figure 6: Operators for generating the size-change graphs for program p.

mutually recursive calls need to be generated and tested.

2.5 Variable dependency SCC generator

The variable dependencies needed in bounded anchoring are the increasing dependencies, so they are based on $\mathcal{E}^{\uparrow}$. The nodes of the variable dependency graph are all function parameters, and the edges can be generated while generating the SCGs (cf. Figure 6). For clarity, Figure 7 shows how they can be generated separately using operator $\mathcal{V}$.

2.6 SCG set closure computation

The closure $\mathcal{S}$ of the set of call graphs generated by operator $\mathcal{G}$ can be computed by a simple worklist algorithm sketched in Figure 8 For efficiency reasons it is sensible to store the size-change graphs indexed by their caller and callee functions, and choose a representation that permits fast composition.

2.7 Bounded anchoring

The algorithm for bounded anchoring of variables is sketched in Figure 9. The notation '$\beta(C)$' denotes the binding time assigned to the SCC C as computed by bounded anchoring, while '$\beta(x)$' denotes the binding time of x computed by the standard BTA. Following the bounded anchoring of variables, a set of sets of call sites, *CallSitesSets* can be computed by the algorithm shown in Figure 10; this set is then the basis for computing a minimised set of specialisation points.

$$
\begin{array}{lcl}
\mathcal{V}\,[\![d_1 \ldots d_n]\!] & = & \mathcal{V}_{\mathbf{d}}\,[\![d_1]\!] \cup \ldots \cup \mathcal{V}_{\mathbf{d}}\,[\![d_n]\!] \\
\mathcal{V}_{\mathbf{d}}\,[\![f\; x_1 \ldots x_n = e]\!] & = & \mathcal{V}_{\mathbf{e}}\; f\,[\![e]\!]\;\; \{x_i \mapsto \{\updownarrow(x_i)\}\} \\
\mathcal{V}_{\mathbf{e}}\; g\,[\![k]\!]\;\rho^{\uparrow} & = & \{\} \\
\mathcal{V}_{\mathbf{e}}\; g\,[\![x]\!]\;\rho^{\uparrow} & = & \{\} \\
\mathcal{V}_{\mathbf{e}}\; g\,[\![\mathbf{let}\; x = e_1\; \mathbf{in}\; e_2]\!]\;\rho^{\uparrow} & = & \mathcal{V}_{\mathbf{e}}\; g\,[\![e_1]\!]\;\rho^{\uparrow} \cup \mathcal{V}_{\mathbf{e}}\, g\,[\![e_2]\!]\;\rho'^{\uparrow} \\
 & & \text{where}\;\; \rho'^{\uparrow} \;\; = \;\; \rho^{\uparrow} + \{x \mapsto \mathcal{E}\; g\,[\![e_1]\!]\;\rho^{\uparrow}\} \\
\mathcal{V}_{\mathbf{e}}\; g\,[\![\mathbf{if}\; e_1\; \mathbf{then}\; e_2\; \mathbf{else}\; e_3]\!]\;\rho^{\uparrow} & = & \mathcal{V}_{\mathbf{e}}\; g\,[\![e_1]\!]\;\rho^{\uparrow} \cup \mathcal{V}_{\mathbf{e}}\; g\,[\![e_2]\!]\;\rho^{\uparrow} \cup \mathcal{V}_{\mathbf{e}}\; g\,[\![e_3]\!]\;\rho^{\uparrow} \\
\mathcal{V}_{\mathbf{e}}\; g\,[\![b\; e_1 \ldots e_n]\!]\;\rho^{\uparrow} & = & \mathcal{V}_{\mathbf{e}}\; g\,[\![e_1]\!]\;\rho^{\uparrow} \cup \ldots \cup \mathcal{V}_{\mathbf{e}}\; g\,[\![e_n]\!]\;\rho^{\uparrow} \\
\mathcal{V}_{\mathbf{e}}\; g\,[\![c : f\; e_1 \ldots e_n]\!]\;\rho^{\uparrow} & = & E \cup \mathcal{V}_{\mathbf{e}}\; g\,[\![e_1]\!]\;\rho^{\uparrow} \cup \ldots \cup \mathcal{V}_{\mathbf{e}}\; g\,[\![e_n]\!]\;\rho^{\uparrow} \\
 & & \text{where}\; E = \left\{\{x \xrightarrow{\delta} f_i \mid \delta(x) \in \mathcal{E}^{\uparrow}\,[\![e_i]\!]\;\rho^{\uparrow}\}\right\}
\end{array}
$$

Figure 7: Operators for generating the edges in the variable dependency graph for program p.

$W \leftarrow \mathcal{S}$
while $W \neq \{\}$ **do**
  $G \leftarrow \mathit{remove}(W)$
  Assume $G : f \rightarrow g$
  for $G' \in \{G' \in \mathcal{S} \mid \exists h\; G' : g \rightarrow h\}$ **do**
    $G'' \leftarrow G; G'$
    if $G'' \notin \mathcal{S}$ **then**
      $\mathcal{S} \leftarrow \mathcal{S} \cup \{G''\}$
      $W \leftarrow W \cup \{G \in \mathcal{S} \mid \exists k\; G : k \rightarrow f\}$

Figure 8: Computing the transitive closure $\mathcal{S}$ of the size-change graphs

for $C \in \mathit{Variable\ SCCs}$ **do**
  if $\forall C' \in \mathit{preds}(C) : \beta(C') = B$ **then**
    let $(x, f) \in \{(x, f) \mid x \in C \wedge x$ is a parameter of $f\}$
    if $\beta(x) \neq D$ **then**
      $\mathit{anchored} \leftarrow \mathsf{true}$
      for $G \in \{G \mid G : f \rightarrow f\}$ **do**
        if $G = G; G$ **then**
          if $x \xrightarrow{\uparrow} x \in G^{\uparrow} \wedge \neg\exists y \xrightarrow{\downarrow} y \in G^{\downarrow} : \beta(C_y) = B \wedge \beta(y) \neq D$ **then** $\mathit{anchored} \leftarrow \mathsf{false}$
      if $\mathit{anchored}$ **then** $\beta(C) \leftarrow B$

Figure 9: Algorithm for bounded anchoring of variables

$\mathit{CallSitesSets} \leftarrow \{\}$
for $C \in \mathit{Call\ graph\ SCCs}$ **do**
  let $f \in C$
  for $G \in \{G \mid G : f \rightarrow f\}$ **do**
    if $G = G; G$ **then**
      if $\neg\exists y \xrightarrow{\downarrow} y \in G^{\downarrow} : \beta(C_y) = B \wedge \beta(y) \neq D$ **then** $\mathit{CallSitesSets} \leftarrow \mathit{CallSitesSets} \cup \{\mathit{calls}(G)\}$
Compute a minimised set of call sites Γ such that $\forall \Gamma' \in \mathit{CallSitesSets} : \Gamma' \cap \Gamma \neq \{\}$

Figure 10: Algorithm for computing sets of potential specialisation points

References

Niels H. Christensen, Robert Glück, and Søren Laursen. Binding-time analysis in partial evaluation: One size does *Not* fit all. In Dines Bjørner, M. Broy, and A. Zamulin, editors, *PSI'99*, volume 1755 of *Lecture Notes in Computer Science*, pages 80–92. Springer-Verlag, Berlin Heidelberg, 2000.

Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts, USA, 1990.

Arne John Glenstrup and Neil D. Jones. Termination analysis and specialization-point insertion in off-line partial evaluation. *ACM Transactions on Programming Languages and Systems*, 2003. submitted.

Fritz Henglein. Efficient type inference for higher-order binding-time analysis. In John Hughes, editor, *International Conference on Functional Programming Languages and Computer Architecture*, volume 523 of *Lecture Notes in Computer Science*, pages 448–472. 5th ACM Conference, Cambridge, MA, USA, Springer-Verlag, August 1991.

Neil D. Jones, Carsten K. Gomard, and Peter Sestoft. *Partial Evaluation and Automatic Program Generation*. International Series in Computer Science. Prentice Hall International, New York, June 1993. ISBN number 0-13-020249-5 (pbk).

Peter Thiemann. A unified framework for binding-time analysis. In M. Bidoit and M. Dauchet, editors, *TAPSOFT '97: Theory and Practice of Software Development, Lille, France*, volume 1214 of *Lecture Notes in Computer Science*, pages 742–756. Springer-Verlag, April 1997.

Peter Thiemann. Aspects of the PGG system: Specialization for standard scheme. pages 412–432, 1999.