Strengthening Supercompilation for Call-By-Value Languages

Peter A. Jonsson  Johan Nordlander
Luleå University of Technology
Background
Context

• Timber: a pure and higher-order call-by-value language
• Program optimization is our goal
• We care about preserving semantics
Wadler’s algorithm

\[ T[\text{app (app } xs \hspace{0.5em} ys) \hspace{0.5em} zs] = \]

\[
\text{app } xs \hspace{0.5em} ys = \text{case } xs \text{ of } \{ [] \rightarrow ys; (x:xs) \rightarrow x: \text{app } xs \hspace{0.5em} ys \} 
\]
Wadler's algorithm

\[ T[\text{app} (\text{app} \, xs \, ys) \, zs] = \]

\[ T[\text{case} \, \text{app} \, xs \, ys \, \text{of} \]
\[ \quad \text{[]} \rightarrow zs \]
\[ \quad (x:xs) \rightarrow x:\text{app} \, xs \, zs \]
\]

\[ \text{app} \, xs \, ys = \text{case} \, xs \, \text{of} \ \{ \text{[]} \rightarrow ys; \ (x:xs) \rightarrow x:\text{app} \, xs \, ys \} \]
$T[\langle x.y \rangle e] = y$
The transformed result:
\[\text{root arguments } = \text{show (h 1 no drive 1 no drive 2)}\]

where
\[\text{no drive 1 ... total runtime, the number of allocations, the total size of allocations and the binary size are all decreased.}\]

The transformed result:
\[T[({\lambda}x.y) e] = y \quad \text{e} = \bot\]

\text{CBN: } (\lambda x.y) \downarrow \rightarrow y

\text{CBV: } (\lambda x.y) \downarrow \rightarrow (\lambda x.y) \downarrow \rightarrow \ldots
Modify CBN algorithm to transform arguments first?

CBN: $(\lambda x.y) \downarrow \rightarrow y$

CBV: $(\lambda x.y) \downarrow \rightarrow (\lambda x.y) \downarrow \rightarrow ...$

\[ T[(\lambda x.y) e] = y \quad \text{e} = \bot \]
Naïve algorithm

\[ N[\text{app (app } xs \ ys) \ zs] = \]

\[ \text{app } xs \ ys = \text{case } xs \ of \ \{ [] \rightarrow ys; \ (x:xs) \rightarrow x:app \ xs \ ys \} \]
Naïve algorithm

\[ N[\text{app } (\text{app } xs \ ys) \ zs] = \]

\[ N[\text{app } (\text{case } xs \text{ of } \{ [] \rightarrow ys; (x:xs) \rightarrow x:\text{app } xs \ ys \}) \ zs] = \]

\[ \text{app } xs \ ys = \text{case } xs \text{ of } \{ [] \rightarrow ys; (x:xs) \rightarrow x:\text{app } xs \ ys \} \]
Naïve algorithm

\[ N[\text{app } (\text{app } xs \ ys) \ zs] = \]
\[ N[\text{app } (\text{case } xs \text{ of } \{ [] \to ys; (x:xs) \to x:\text{app } xs \ ys \}) \ zs] = \]

\text{case } xs \text{ of }
\[ [] \to N[\text{app } (ys) \ zs] \]
\[ (x:xs) \to N[\text{app } (x:\text{app } xs \ ys) \ zs] \]

app xs ys = case xs of {[] -> ys; (x:xs) -> x:app xs ys}
app (app xs ys) zs

app (x:app xs ys) zs
app (app xs ys) zs <

app (x:app xs ys) zs
\[ N[\text{app (app xs ys) zs}] = \text{case xs of} \]
\[ \quad [] \rightarrow \text{h1 ys zs} \]
\[ \quad (x:xs) \rightarrow \text{h1 (h2 x xs ys) zs} \]

\[ \text{h1 xs ys} = \text{case xs of} \{ [] \rightarrow ys; (x:xs) \rightarrow x: \text{h1 xs ys} \} \]
\[ \text{h2 x xs ys} = x: \text{case xs of} \{ [] \rightarrow ys; (x:xs) \rightarrow h2 x xs ys \} \]
N[app (app xs ys) zs] = case xs of
    [] -> h1 ys zs
    (x:xs) -> h1 (h2 x xs ys) zs

h1 xs ys = case xs of {[] -> ys; (x:xs) -> x:h1 xs ys}
h2 x xs ys = x:case xs of {[] -> ys; (x:xs) -> h2 x xs ys}
Our algorithm

\[ D[\text{app} (\text{app} \; xs \; ys) \; zs] = \]

\text{app} \; xs \; ys = \text{case} \; xs \; \text{of} \; \{ [] \rightarrow ys; \; (x:xs) \rightarrow x:\text{app} \; xs \; ys \}
Our algorithm

\[ D[\text{app (app } xs \text{ } ys) \text{ } zs] = \]

\[ D[\text{let } xs = \text{app } xs \text{ } ys, \text{ } ys = \text{zs in case } xs \text{ of} \]
\[ [] \rightarrow ys \]
\[ (x:xs) \rightarrow x:\text{app } xs \text{ } ys] \]

\[ \text{app } xs \text{ } ys = \text{case } xs \text{ of} \{ [] \rightarrow ys; (x:xs) \rightarrow x:\text{app } xs \text{ } ys \} \]
Our algorithm

\[ D[\text{app (app } xs \text{ ys)} zs] = \]

\[ D[\text{let } xs = \text{app } xs \text{ ys}, \text{ys} = zs \text{ in case } xs \text{ of} \]
\[ \text{[]} \rightarrow \text{ys} \]
\[ (x:xs) \rightarrow x:\text{app } xs \text{ ys} \]

app \text{ } xs \text{ } ys = \text{case } xs \text{ of } \{ \text{[]} \rightarrow \text{ys}; (x:xs) \rightarrow x:\text{app } xs \text{ ys} \}
Our algorithm

\[ D[\text{app (app } \times s \ \text{ys}) \ zs] = \]

\[ D[\text{let } \times s = \text{app } \times s \ \text{ys}, \ \text{ys} = \ zs \ \text{in} \ \text{case } \times s \ \text{of} \]

\[ [\] \rightarrow \ \text{ys} \]

\[ (x:\times s) \rightarrow x:\text{app } \times s \ \text{ys} \]

\[ D[\text{case } \text{app } \times s \ \text{ys} \ \text{of} \]

\[ [\] \rightarrow \ \text{zs} \]

\[ (x:\times s) \rightarrow x:\text{app } \times s \ \text{zs} \]

\[ \text{app } \times s \ \text{ys} = \text{case } \times s \ \text{of} \ \{[\] \rightarrow \ \text{ys}; (x:\times s) \rightarrow x:\text{app } \times s \ \text{ys}\} \]
\[
D[\text{app} (\text{app} \; xs \; ys) \; zs] = h3 \; xs \; ys \; zs
\]

\[
h3 \; xs \; ys \; zs = \text{case} \; xs \; \text{of}
\]
\[
[] \rightarrow \text{case} \; ys \; \text{of}
\]
\[
[] \rightarrow zs
\]
\[
(y:ys) \rightarrow y:h4 \; ys \; zs
\]
\[
(x:xs) \rightarrow x:h3 \; xs \; ys \; zs
\]

\[
h4 \; xs \; ys = \text{case} \; xs \; \text{of} \; \{ [ ] \rightarrow ys; (x:xs) \rightarrow x:h4 \; xs \; ys \}\]
D[app (app xs ys) zs] = h3 xs ys zs

h3 xs ys zs = case xs of
    [] -> case ys of
        [] -> zs
        (y:ys) -> y : h4 ys zs
    (x:xs) -> x : h3 xs ys zs

h4 xs ys = case xs of { [] -> ys; (x:xs) -> x : h4 xs ys }
(Almost) Follow Sørensen et al. (1996)

\[ E ::= [] | E \text{ es} | \text{case } E \text{ of alts} \]

\[ \mathcal{D}[x] = x \]
\[ D[k \text{ es}] = k \ D[\text{es}] \]
\[ D[E<\langle\\text{xs.e}\rangle \text{ es}>] = D[E<\text{let } \text{xs} = \text{ es } \text{ in } e>] \]
\[ D[E<\text{let } x = e \text{ in } f>] = D[E<[e/x]f>] \text{ if } x \in \text{strict}(f) \]
\[ \quad \text{and } x \in \text{linear}(f) \]
\[ \quad = \text{let } x = D[e] \text{ in } D[E<f>] \]
\[ D[E<\text{case } x \text{ of } \{k_i \ x_i -> e_i\}>] \]
\[ \quad = \text{case } x \text{ of } \{k_i \ x_i -> D[[k_i \ x_i/x]E<e_i>]\} \]
\[ D[E<\text{case } k_j \text{ es of } \{k_i \ x_i -> e_i\}>] = D[E<\text{let } x_j = \text{ es } \text{ in } e_j>] \]
(Almost) Follow Sørensen et al. (1996)

\[E ::= [] \mid E \text{ es} \mid \text{case } E \text{ of } \text{alts}\]

\[
\begin{align*}
\hat{D}[x] &= x \\
D[k \text{ es}] &= k \ D[\text{es}] \\
D[E<(\backslash x\ es) \ es>] &= D[E<\text{let } xs = es \text{ in } e>] \\
D[E<\text{let } x = e \text{ in } f>] &= D[E<[e/x]f>] \text{ if } x \in \text{strict}(f) \\
&\quad \text{ and } x \in \text{linear}(f) \\
&= \text{let } x = D[e] \text{ in } D[E<f>] \\
D[E<\text{case } x \text{ of } \{k_i \ x_i \to e_i\}>] &= \text{case } x \text{ of } \{k_i \ x_i \to D[[k_i \ x_i/x]E<e_i>]\} \\
D[E<\text{case } k_j \ es \text{ of } \{k_i \ x_i \to e_i\}>] &= D[E<\text{let } x_j = es \text{ in } e_j>] 
\end{align*}
\]
\[(\text{Almost}) \text{ Follow Sørensen et al. (1996)}\]

\[
E ::= [] \mid E \mathbf{es} \mid \text{case } E \text{ of alts}
\]

\[
\begin{align*}
\hat{D}[x] &= x \\
D[k \ es] &= k \ D[es] \\
D[E<\langle \mathbf{\lambda} xs. e \rangle \ es>] &= D[E<\mathbf{let} \ xs = es \ \text{in} \ e>] \\
D[E<\mathbf{let} \ x = e \ \text{in} \ f>] &= D[E<\mathbf{[e/x]} f>] \text{ if } x \in \text{strict}(f) \\
& \quad \text{and } x \in \text{linear}(f) \\
& \quad = \mathbf{let} \ x = D[e] \ \text{in} \ D[E<f>] \\
D[E<\mathbf{case} \ x \ \text{of} \ \{ k_i \ x_i \rightarrow e_i \} >] \\
& \quad = \mathbf{case} \ x \ \text{of} \ \{ k_i \ x_i \rightarrow D[[k_i \ x_i/x]E<e_i>] \} \\
D[E<\mathbf{case} \ k_j \ es \ \text{of} \ \{ k_i \ x_i \rightarrow e_i \} >] &= D[E<\mathbf{let} \ x_j = es \ \text{in} \ e_j>] 
\end{align*}
\]
Sum Square

Before optimization
After optimization

Double append

0 ticks 100,000,000 ticks 200,000,000 ticks 300,000,000 ticks
Memory allocated

Before optimization
After optimization

Sum Square

Double append
But..
CBV-supercompilation is weaker than CBN-supercompilation

\[ D[\text{zip } (\text{map } f \ xs') \ (\text{map } g \ ys')] = \]
CBV-supercompilation is weaker than CBN-supercompilation

\[
D[\text{zip} \ (\text{map} \ f \ xs') \ (\text{map} \ g \ ys')] = \\
D[\text{let} \ ys = \text{map} \ g \ ys' \\
\text{in} \ \text{case} \ \text{map} \ f \ xs' \ \text{of} \\
\quad [] \to [] \\
\quad (x':xs') \to \text{case} \ ys \ \text{of} \\
\quad \quad [] \to [] \\
\quad \quad (y':ys') \to (x', y') : \text{zip} \ xs' \ ys']
\]
CBV-supercompilation is weaker than CBN-supercompilation

\[ D[\text{zip} \ (\text{map} \ f \ \text{x}s') \ \text{(map} \ g \ \text{y}s')] = \]

\[ D[\text{let} \ \text{y}s = \text{map} \ g \ \text{y}s' \]
\[ \text{in} \ \text{case} \ \text{map} \ f \ \text{x}s' \ \text{of} \]
\[ [\] \rightarrow [] \]
\[ (x' : \text{x}s') \rightarrow \text{case} \ \text{y}s \ \text{of} \]
\[ [\] \rightarrow [] \]
\[ (y' : y's') \rightarrow (x', y') : \text{zip} \ \text{x}s' \ \text{y}s'] \]
Let $ys = \text{map } g \ ys'$
in case (case $xs'$ of
  $[] \rightarrow []$
  $(x'':xs'') \rightarrow f \ x'':\text{map } f \ xs''$) of
  $[] \rightarrow []$
  $(x'':xs'') \rightarrow \text{case } ys \text{ of}$
    $[] \rightarrow []$
    $(y'':ys'') \rightarrow (x', y'):\text{zip } xs' \ ys'$]
```haskell
case xs of
  [] -> D[let ys = map g ys'
    in case [] of
      [] -> []
      (x':xs') -> case ys of
        [] -> []
        (y':ys') -> (x', y'):zip xs' ys'
    ]
  (x'':xs'') -> D[let ys = map g ys'
    in case f x'':map f xs'' of
      [] -> []
      (x':xs') -> case ys of
        [] -> []
        (y':ys') -> (x', y'):zip xs' ys'
    ]
```
D[zip (map f xs') (map g ys')] = h5 f xs' g ys'

h5 f xs' g ys' =
  case xs' of
    [] -> let ys = map g ys' in []
    (z:zs) -> case ys' of
      [] -> let x' = f z, xs' = map f zs in []
      (z':zs') -> (f z, g z') : h5 f zs g zs'
h5 f xs' g ys' =
  case xs' of
    [] -> let ys = map g ys' in []
    (z:zs) -> case ys' of
      [] -> let x' = f z, xs' = map f zs in []
      (z':zs') -> (f z, g z') : h5 f zs g zs'
Extended Let Rule

\[
D[E<\text{let } x = e \text{ in } f>] B \\
= D[E<\text{let } B \text{ in } f>] O \text{ if terminates}(e) \\
\quad \text{ and } x \not\in \text{fv}(f) \\
= D[E<\text{let } B \text{ in } [e/x]f>] O \text{ if } x \in \text{strict}(f) \\
\quad \text{ and } x \in \text{linear}(f) \\
= D[E<f>] (B \cup (x, e))
\]
Future Work

• More measurements on real programs
• Investigate cheaper methods for ensuring termination
• What about accumulating parameters?
Related Work

• Supercompilers:
  • Scp4 (Nemytykh 2003)
  • Mitchell (2008, 2010)
  • Bolingbroke & Peyton Jones (2010)
  • Reich et al. (2010)
Conclusions

Supercompilation for call-by-value languages:

• can be strengthened to close in on call-by-name supercompilers