Online-and-Offline Partial Evaluation: A Mixed Approach

Eijiro Sumii
Naoki Kobayashi
(University of Tokyo)

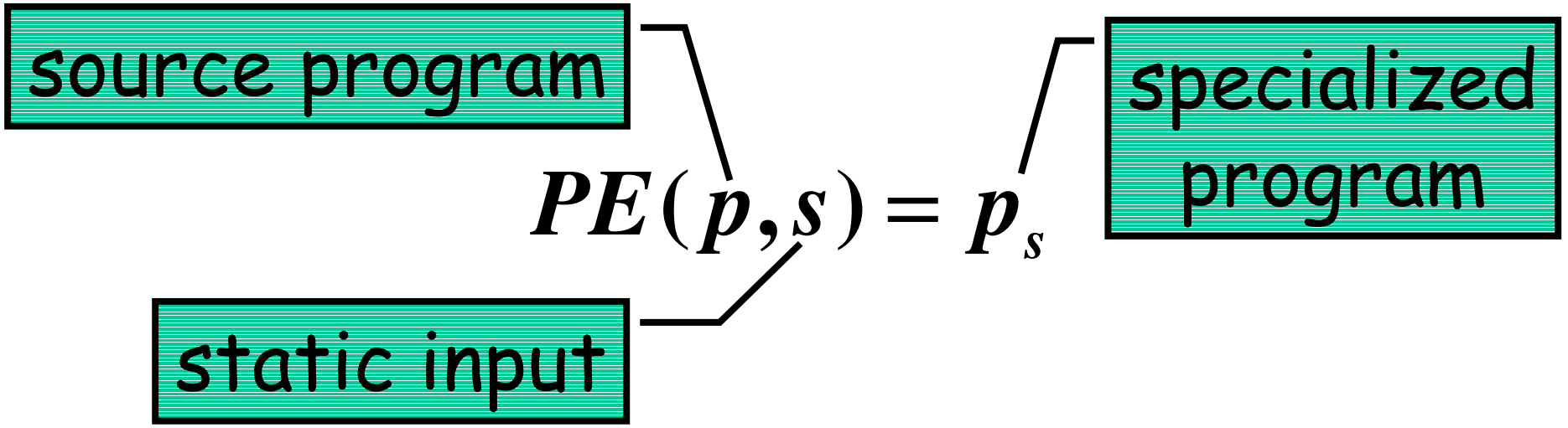
Overview

- Introduction
- Simple Online PE
- Our Method
- Experiments
- Related Work
- Conclusion

Overview

- Introduction
- Simple Online PE
- Our Method
- Experiments
- Related Work
- Conclusion

Partial Evaluation



- *Reduce* static computations
- *Residualize* dynamic computations

$$\textbf{s.t. } p_s(d) = p(s,d) \textbf{ for any } d$$

dynamic input

Two Issues in PE

1. Efficiency of *specialized programs*
   (analogy: efficiency of compiled programs)
2. Efficiency of *specialization*
   (analogy: efficiency of compilation)

Both are important.

Online PE and Offline PE

When to decide whether a computation is static or dynamic?

- Online PE:
  During the specialization, with a static input
- Offline PE:
  Before the specialization, w/o a static input

Online v.s. Offline

Online PE (analogy: dynamic typing)

- Finer decision

  $\Rightarrow$ faster *specialized programs*

Offline PE (analogy: static typing)

- Faster *specialization*
- Easier reasoning

Either approach has its advantage.

Our Approach

Hybrid of online/offline PE
(analogy: soft typing)

- Make an offline decision
  so far as it is precise
- Otherwise, make an online decision

Results

- Specialized programs:
  as fast as simple online PE
- Specialization:
  1.5-8 times faster than
  simple offline PE
  (for the same specialization)
  - thanks to the decrease of
    unnecessary let-insertions
    (explained later)

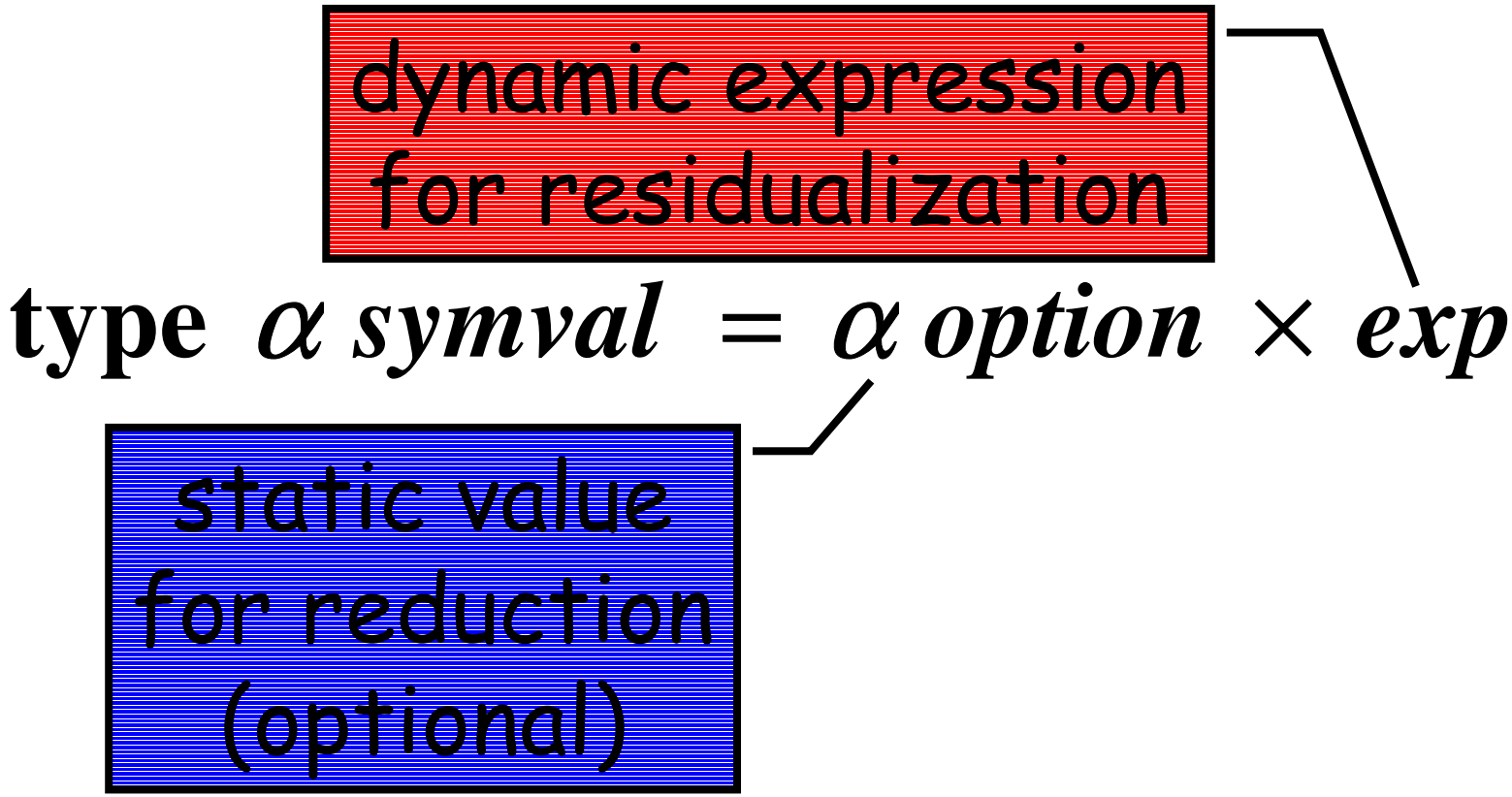
Overview

- Introduction
- Simple Online PE
- Our Method
- Experiments
- Related Work
- Conclusion

Overview

- Introduction
- Simple Online PE
- Our Method
- Experiments
- Related Work
- Conclusion

Symbolic Values



type α *option* = *Some* α | *None*

Simple Online PE

$$[|\, x \,|] \quad = x$$

$$[|\, \lambda x.e \,|] \quad = \langle \mathit{Some}(\lambda x.[|e|]),\ \underline{\lambda x'} \ldots . \rangle$$

$$[|\, e_1 e_2 \,|] \quad = \mathbf{case}\ [|e_1|]$$

$$\quad \mathbf{of}\ \langle \mathit{Some}\ s, _ \rangle \Rightarrow s\ [|e_2|]$$

$$\quad \mid \langle \mathit{None}, d \rangle \Rightarrow \langle \mathit{None}, d\ \underline{@}\ \mathbf{snd}\ [|e_2|] \rangle$$

Let-Insertion is Necessary

- to preserve semantics under effects
- to avoid code duplication

Let-Insertion is Necessary

- to preserve semantics under effects

  $\lambda f.\ \mathbf{let}\ y = f\ x\ \mathbf{in}\ 1{+}2$

  ➡ $\lambda f.\ \mathbf{let}\ y = f\ x\ \mathbf{in}\ 3$

  rather than $\lambda f.3$

- to avoid code duplication

Let-Insertion is Necessary

- to preserve semantics under effects
- to avoid code duplication

  $\mathbf{let}\ f = \lambda x.1{+}2\ \mathbf{in}\ (f, f)$

  ➡ $\mathbf{let}\ f = \lambda x.3\ \mathbf{in}\ (f, f)$

  rather than $(\lambda x.3, \lambda x.3)$

Simple Online PE

$$[|\, x \,|] \quad = x$$

$$[|\, \lambda x.e \,|] \quad = \langle \mathit{Some}(\lambda x.[|e|]),\ \underline{\lambda x'}\ldots. \rangle$$

$$[|\, e_1\ e_2 \,|] \quad = \mathbf{case}\ [|e_1|]$$
$$\mathbf{of}\ \langle \mathit{Some}\ s, _ \rangle \Rightarrow s\ [|e_2|]$$
$$\mid \langle \mathit{None}, d \rangle \Rightarrow \langle \mathit{None}, d\ \underline{@}\ \mathbf{snd}\ [|e_2|] \rangle$$

Simple Online PE with Let-Insertion

$$[|\, x \,|] \quad = x$$

$$[|\, \lambda x.e \,|] \quad = \langle \mathit{Some}(\lambda x.\mathit{delimit\text{-}let}([|e|])),\ \mathit{insert\text{-}let}(\underline{\lambda x'}\ldots.)\rangle$$

$$[|\, e_1\ e_2 \,|] \quad = \mathbf{case}\ [|e_1|]\ \mathbf{of}\ \langle \mathit{Some}\ s, _\rangle \Rightarrow s\ [|e_2|]\ |\ \langle \mathit{None}, d\rangle \Rightarrow \langle \mathit{None}, \mathit{insert\text{-}let}(d\ \underline{@}\ \mathbf{snd}\ [|e_2|])\rangle$$

Simple Online PE is Slow

because of:

- unnecessary tagging (*None*/*Some*)
- unnecessary let-insertion
- unnecessary residualization

$$[\!| (\lambda x.x)3 |\!] =$$

$$\textbf{case } \langle \mathit{Some}(\lambda x.x), \mathit{insert\text{-}let}(\underline{\lambda x'}.\ldots) \rangle$$

$$\textbf{of } \langle \mathit{Some}(s), _ \rangle \Rightarrow s\ 3$$

$$\mid \langle \mathit{None}, d \rangle \Rightarrow \ldots$$

Simple Online PE is Slow

because of:

- unnecessary tagging (*None*/*Some*)
- unnecessary let-insertion
- unnecessary residualization

Simple Online PE is Slow

because of:

- unnecessary tagging (*None*/*Some*)
- unnecessary let-insertion
- unnecessary residualization

$$[\!|\ (\lambda x.x)3\ |\!] =$$

$$\textbf{case}\ \langle \mathit{Some}(\lambda x.x),\ \mathit{insert\text{-}let}(\underline{\lambda x'}.\ldots)\rangle$$

$$\textbf{of}\ \langle \mathit{Some}(s),\ _\rangle \Rightarrow s\ 3$$

$$\mid \langle \mathit{None}, d\rangle \Rightarrow \ldots$$

Simple Online PE is Slow

because of:

- unnecessary tagging (*None*/*Some*)
- unnecessary let-insertion
- unnecessary residualization

$$[\!|\ (\lambda x.x)3\ |\!] =$$
$$\textbf{case}\ \langle \lambda x.x,\ \textit{let-insert}(\underline{\lambda x'}\ldots.)\rangle$$
$$\textbf{of}\ \langle s,\ _\rangle \Rightarrow s\ 3$$

Simple Online PE is Slow

because of:

- unnecessary tagging (*None*/*Some*)
- unnecessary let-insertion
- unnecessary residualization

$[\![(\lambda x.x)3]\!] =$

case $\langle \lambda x.x,$ *let-insert*$(\underline{\lambda x'}\ldots.)\rangle$

of $\langle s, _\rangle \Rightarrow s\ 3$

Simple Online PE is Slow

because of:

- unnecessary tagging (*None*/*Some*)
- unnecessary let-insertion
- unnecessary residualization

$$[\![\ (\lambda x.x)3\ ]\!] =$$
$$\textbf{case}\ \lambda x.x$$
$$\textbf{of}\ s \Rightarrow s\ 3$$

Overview

- Introduction
- Simple Online PE
- Our Method
- Experiments
- Related Work
- Conclusion

What Information is Useful?

- A static value is…
  - always/never available
    ⇒ Tagging becomes unnecessary
- A dynamic expression…
  - never remains
    ⇒ Residualization becomes unnecessary
  - remains at most once (& has no effects)
    ⇒ Let-insertion becomes unnecessary

Types

ρ (raw type) ::= b_i | $\tau_1 \to \tau_2$ | $\tau_1 \times \tau_2$

τ (annotated type) ::= $\rho^{(s,d)}$

s (static use) ::= **0** (never) | **ω** (always)
| **T** (sometimes)

"Whether a static value is available"

d (dynamic use) ::= **0** (never)
| **1** (at most once) | **ω** (any number of times)

"How many times
a dynamic expression remains"

Types

ρ (raw type) ::= b_i | $\tau_1 \rightarrow \tau_2$ | $\tau_1 \times \tau_2$

τ (annotated type) ::= $\rho^{(s,d)}$

Types

ρ (raw type) ::= b_i | $\tau_1 \rightarrow \tau_2$ | $\tau_1 \times \tau_2$

τ (annotated type) ::= $\rho^{(s,d)}$

s (static use) ::= **0** (never) | **ω** (always)
| **T** (sometimes)

"Whether a static value is available"

Types

ρ (raw type) ::= b_i | $\tau_1 \to \tau_2$ | $\tau_1 \times \tau_2$

τ (annotated type) ::= $\rho^{(s,d)}$

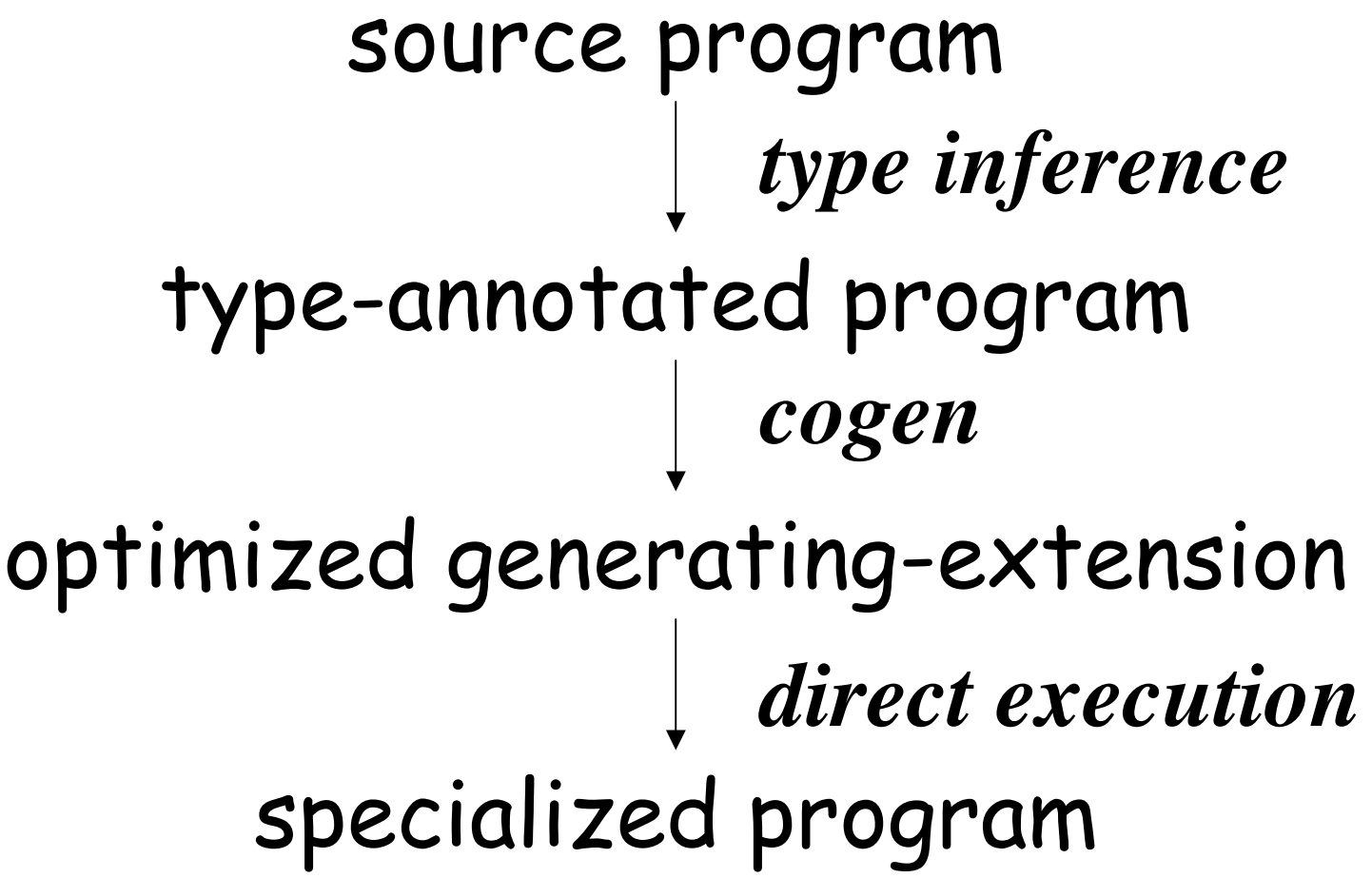
s (static use) ::= **0** (never) | ω (always)
| **T** (sometimes)

"Whether a static value is available"

d (dynamic use) ::= **0** (never)
| **1** (at most once) | ω (any number of times)

"How many times
a dynamic expression remains"

Framework of our Method



Examples of Typing (I): Static Uses (0, ω, and T)

$\lambda g.\ \mathbf{let}\ f : \mathbf{int} \rightarrow^{(0,1)} \alpha = g\ \mathbf{in}\ f\ 3$

⇨ $\underline{\lambda g}.\ \mathbf{let}\ f = \underline{g}\ \mathbf{in}\ f\underline{@}3$

➡ $\underline{\lambda g}.\ g\underline{@}3$

$\lambda g.\ \mathbf{let}\ f : \mathbf{int} \rightarrow^{(\omega,0)} \mathbf{int} = \lambda x.x\ \mathbf{in}\ f\ 3$

⇨ $\underline{\lambda g}.\ \mathbf{let}\ f = \lambda x.x\ \mathbf{in}\ f\ 3$

➡ $\underline{\lambda g}.\ 3$

$\lambda g.\ \mathbf{let}\ f : \mathbf{int} \rightarrow^{(T,1)} \mathbf{int} = (\mathbf{if}\ \mathit{true}\ \mathbf{then}\ \lambda x.x\ \mathbf{else}\ g)$
$\mathbf{in}\ f\ 3$

⇨ $\underline{\lambda g}.\ \mathbf{let}\ f = (\mathbf{if}\ \mathit{true}\ \mathbf{then}\ \langle \mathit{Some}(\lambda x.x), \ldots \rangle$

Examples of Typing (I): Static Uses ($\mathbf{0}$, ω, and $\mathbf{T}$)

$\lambda g.\ \mathbf{let}\ f = g\ \mathbf{in}\ f\ 3$

Examples of Typing (I): Static Uses (**0**, ω, and **T**)

$\lambda g.\ \mathbf{let}\ f : \mathbf{int} \rightarrow^{(\mathbf{0},1)} \alpha = g\ \mathbf{in}\ f\ 3$

Examples of Typing (I): Static Uses (**0**, ω, and **T**)

$\lambda g.\ \mathbf{let}\ f : \mathbf{int} \rightarrow^{(0,1)} \alpha = g\ \mathbf{in}\ f\ 3$

⇨ $\underline{\lambda g}.\ \mathbf{let}\ f = g\ \mathbf{in}\ f\underline{@}3$

Examples of Typing (I): Static Uses (**0**, ω, and **T**)

$\lambda g.\ \mathbf{let}\ f : \mathbf{int} \rightarrow^{(0,1)} \alpha = g\ \mathbf{in}\ f\ 3$

⇨ $\underline{\lambda g}.\ \mathbf{let}\ f = g\ \mathbf{in}\ f\underline{@}3$

➡ $\underline{\lambda g}.\ g\underline{@}3$

Examples of Typing (I): Static Uses (**0**, ω, and **T**)

$\lambda g.\ \mathbf{let}\ f : \mathbf{int} \rightarrow^{(0,1)} \alpha = g\ \mathbf{in}\ f\ 3$

⇨ $\underline{\lambda g}.\ \mathbf{let}\ f = g\ \mathbf{in}\ f \underline{@} 3$

➡ $\underline{\lambda g}.\ g \underline{@} 3$

$\lambda g.\ \mathbf{let}\ f = \lambda x.x\ \mathbf{in}\ f\ 3$

Examples of Typing (I): Static Uses (**0**, ω, and **T**)

$\lambda g.\ \mathbf{let}\ f : \mathbf{int} \rightarrow^{(0,1)} \alpha = g\ \mathbf{in}\ f\ 3$

⇨ $\underline{\lambda g}.\ \mathbf{let}\ f = g\ \mathbf{in}\ f\underline{@}3$

➡ $\underline{\lambda g}.\ g\underline{@}3$

$\lambda g.\ \mathbf{let}\ f : \mathbf{int} \rightarrow^{(\omega,0)} \mathbf{int} = \lambda x.x\ \mathbf{in}\ f\ 3$

Examples of Typing (I): Static Uses (**0**, ω, and **T**)

$\lambda g.\ \mathbf{let}\ f : \mathbf{int} \rightarrow^{(0,1)} \alpha = g\ \mathbf{in}\ f\ 3$

⇨ $\underline{\lambda g}.\ \mathbf{let}\ f = g\ \mathbf{in}\ f \underline{@} 3$

➡ $\underline{\lambda g}.\ g \underline{@} 3$

$\lambda g.\ \mathbf{let}\ f : \mathbf{int} \rightarrow^{(\omega,0)} \mathbf{int} = \lambda x.x\ \mathbf{in}\ f\ 3$

⇨ $\underline{\lambda g}.\ \mathbf{let}\ f = \lambda x.x\ \mathbf{in}\ f\ 3$

Examples of Typing (I): Static Uses (**0**, ω, and **T**)

$\lambda g.\ \mathbf{let}\ f : \mathbf{int} \rightarrow^{(0,1)} \alpha = g\ \mathbf{in}\ f\ 3$

⇨ $\underline{\lambda g}.\ \mathbf{let}\ f = g\ \mathbf{in}\ f \underline{@} 3$

➡ $\underline{\lambda g}.\ g \underline{@} 3$

$\lambda g.\ \mathbf{let}\ f : \mathbf{int} \rightarrow^{(\omega,0)} \mathbf{int} = \lambda x.x\ \mathbf{in}\ f\ 3$

⇨ $\underline{\lambda g}.\ \mathbf{let}\ f = \lambda x.x\ \mathbf{in}\ f\ 3$

➡ $\underline{\lambda g}.\ 3$

Examples of Typing (I): Static Uses (**0**, ω, and **T**)

$\lambda g.\ \mathbf{let}\ f : \mathbf{int} \rightarrow^{(0,1)} \alpha = g\ \mathbf{in}\ f\ 3$ ➡ $\lambda g.\ g\ 3$

$\lambda g.\ \mathbf{let}\ f : \mathbf{int} \rightarrow^{(\omega,0)} \mathbf{int} = \lambda x.x\ \mathbf{in}\ f\ 3$ ➡ $\lambda g.\ 3$

$\lambda g.$ let f = (if *true* then $\lambda x.x$ else g)

in f 3

Examples of Typing (I): Static Uses (0, ω, and T)

$\lambda g.\ \text{let } f : \text{int} \to^{(0,1)} \alpha = g \text{ in } f\ 3 \ \Rightarrow\ \lambda g.\ g\ 3$

$\lambda g.\ \text{let } f : \text{int} \to^{(\omega,0)} \text{int} = \lambda x.x \text{ in } f\ 3 \ \Rightarrow\ \lambda g.\ 3$

$\lambda g.\ \text{let } f : \text{int} \to^{(T,1)} \text{int} = (\text{if } \mathit{true} \text{ then } \lambda x.x \text{ else } g)$
in f 3

Examples of Typing (I): Static Uses (0, ω, and T)

$\lambda g.\ \text{let } f : \text{int} \to^{(0,1)} \alpha = g \text{ in } f\ 3 \ \Rightarrow\ \lambda g.\ g\ 3$

$\lambda g.\ \text{let } f : \text{int} \to^{(\omega,0)} \text{int} = \lambda x.x \text{ in } f\ 3 \ \Rightarrow\ \lambda g.\ 3$

$\lambda g.\ \text{let } f : \text{int} \to^{(T,1)} \text{int} = (\text{if } \mathit{true} \text{ then } \lambda x.x \text{ else } g)$
in f 3

$\Rightarrow\ \underline{\lambda g}.\ \text{let } f = (\text{if } \mathit{true} \text{ then } \langle \mathit{Some}(\lambda x.x), \ldots \rangle$
$\text{else } \langle \mathit{None}, g \rangle)$
$\text{in } (\text{case } f \text{ of } \langle \mathit{Some}\ s, _ \rangle \Rightarrow s\ 3$
$\mid \langle \mathit{None}, d \rangle \Rightarrow d\underline{@}3)$

Examples of Typing (I): Static Uses (0, ω, and T)

$\lambda g.\ \text{let } f : \text{int} \to^{(0,1)} \alpha = g \text{ in } f\ 3 \ \Rightarrow\ \lambda g.\ g\ 3$

$\lambda g.\ \text{let } f : \text{int} \to^{(\omega,0)} \text{int} = \lambda x.x \text{ in } f\ 3 \ \Rightarrow\ \lambda g.\ 3$

$\lambda g.\ \text{let } f : \text{int} \to^{(T,1)} \text{int} = (\text{if } \mathit{true} \text{ then } \lambda x.x \text{ else } g)$
in f 3

$\Rightarrow\ \underline{\lambda g}.\ \text{let } f = (\text{if } \mathit{true} \text{ then } \langle \mathit{Some}(\lambda x.x), \ldots \rangle$
$\text{else } \langle \mathit{None}, g \rangle)$
$\text{in } (\text{case } f \text{ of } \langle \mathit{Some}\ s, _ \rangle \Rightarrow s\ 3$
$\mid \langle \mathit{None}, d \rangle \Rightarrow d\underline{@}3)$

$\Rightarrow\ \underline{\lambda g}.\ 3$

Examples of Typing (II): Dynamic Uses (0, 1, and ω)

let f **:** $\mathbf{int} \rightarrow^{(0,0)} \mathbf{int} = \lambda x.1+2+x$ **in** 7

⇨ **let** $f = \langle\rangle$ **in** 7

➡ 7

let f **:** $\mathbf{int} \rightarrow^{(0,1)} \mathbf{int} = \lambda x.1+2+x$ **in** f

⇨ **let** $f = \underline{\lambda x}.1+2\underline{+x}$ **in** f

➡ $\underline{\lambda x}.3\underline{+x}$

let f **:** $\mathbf{int} \rightarrow^{(0,\omega)} \mathbf{int} = \lambda x.1+2+x$ **in** $\langle f, f \rangle$

⇨ **let** $f =$ *insert-let*$(\underline{\lambda x}.1+2\underline{+x})$ **in** (f, f)

➡ **let** $f' = \underline{\lambda x}.3\underline{+x}$ **in** (f', f')

Examples of Typing (II): Dynamic Uses ($\mathbf{0}$, $\mathbf{1}$, and ω)

$\mathbf{let}\ f = \lambda x.1+2+x\ \mathbf{in}\ 7$

Examples of Typing (II): Dynamic Uses (**0**, **1**, and ω)

let f **: int** $\rightarrow^{(0,\mathbf{0})}$ **int** = $\lambda x.1+2+x$ **in 7**

Examples of Typing (II): Dynamic Uses (**0**, **1**, and ω)

$\textbf{let } f : \textbf{int} \rightarrow^{(0,\mathbf{0})} \textbf{int} = \lambda x.1+2+x \textbf{ in } 7$

⇨ $\textbf{let } f = \langle\rangle \textbf{ in } 7$

Examples of Typing (II): Dynamic Uses (0, 1, and ω)

$\textbf{let } f : \textbf{int} \rightarrow^{(0,0)} \textbf{int} = \lambda x.1+2+x \textbf{ in } 7$

⇨ $\textbf{let } f = \langle\rangle \textbf{ in } 7$

➡ 7

Examples of Typing (II): Dynamic Uses (**0**, **1**, and ω)

$\mathbf{let}\ f : \mathbf{int} \rightarrow^{(0,0)} \mathbf{int} = \lambda x.1+2+x\ \mathbf{in}\ 7$

⇨ $\mathbf{let}\ f = \langle\rangle\ \mathbf{in}\ 7$

➡ 7

$\mathbf{let}\ f = \lambda x.1+2+x\ \mathbf{in}\ f$

Examples of Typing (II): Dynamic Uses (**0**, **1**, and ω)

$\mathbf{let}\ f : \mathbf{int} \to^{(0,0)} \mathbf{int} = \lambda x.1+2+x\ \mathbf{in}\ 7$

⇨ $\mathbf{let}\ f = \langle\rangle\ \mathbf{in}\ 7$

➡ 7

$\mathbf{let}\ f : \mathbf{int} \to^{(0,1)} \mathbf{int} = \lambda x.1+2+x\ \mathbf{in}\ f$

Examples of Typing (II): Dynamic Uses (0, 1, and ω)

$\mathbf{let}\ f : \mathbf{int} \to^{(0,0)} \mathbf{int} = \lambda x.1+2+x\ \mathbf{in}\ 7$

⇨ $\mathbf{let}\ f = \langle\rangle\ \mathbf{in}\ 7$

➡ 7

$\mathbf{let}\ f : \mathbf{int} \to^{(0,1)} \mathbf{int} = \lambda x.1+2+x\ \mathbf{in}\ f$

⇨ $\mathbf{let}\ f = \underline{\lambda x}.1+2\underline{+x}\ \mathbf{in}\ f$

Examples of Typing (II): Dynamic Uses (0, 1, and ω)

$\mathbf{let}\ f : \mathbf{int} \to^{(0,0)} \mathbf{int} = \lambda x.1+2+x\ \mathbf{in}\ 7$

⇨ $\mathbf{let}\ f = \langle\rangle\ \mathbf{in}\ 7$

➡ 7

$\mathbf{let}\ f : \mathbf{int} \to^{(0,1)} \mathbf{int} = \lambda x.1+2+x\ \mathbf{in}\ f$

⇨ $\mathbf{let}\ f = \underline{\lambda x}.1+2\underline{+x}\ \mathbf{in}\ f$

➡ $\underline{\lambda x}.3\underline{+x}$

Examples of Typing (II): Dynamic Uses (**0**, **1**, and ω)

$\mathbf{let}\ f : \mathbf{int} \to^{(0,0)} \mathbf{int} = \lambda x.1+2+x\ \mathbf{in}\ 7$

⇨ $\mathbf{let}\ f = \langle\rangle\ \mathbf{in}\ 7$

➡ 7

$\mathbf{let}\ f : \mathbf{int} \to^{(0,1)} \mathbf{int} = \lambda x.1+2+x\ \mathbf{in}\ f$

⇨ $\mathbf{let}\ f = \underline{\lambda x}.1+2\underline{+x}\ \mathbf{in}\ f$

➡ $\underline{\lambda x}.3\underline{+x}$

$\mathbf{let}\ f = \lambda x.1+2+x\ \mathbf{in}\ \langle f, f\rangle$

Examples of Typing (II): Dynamic Uses (**0**, **1**, and ω)

$\mathbf{let}\ f : \mathbf{int} \to^{(0,0)} \mathbf{int} = \lambda x.1+2+x\ \mathbf{in}\ 7$

⇨ $\mathbf{let}\ f = \langle\rangle\ \mathbf{in}\ 7$

➡ 7

$\mathbf{let}\ f : \mathbf{int} \to^{(0,1)} \mathbf{int} = \lambda x.1+2+x\ \mathbf{in}\ f$

⇨ $\mathbf{let}\ f = \underline{\lambda x}.1+2\underline{+x}\ \mathbf{in}\ f$

➡ $\underline{\lambda x}.3\underline{+x}$

$\mathbf{let}\ f : \mathbf{int} \to^{(0,\omega)} \mathbf{int} = \lambda x.1+2+x\ \mathbf{in}\ \langle f, f\rangle$

Examples of Typing (II): Dynamic Uses (0, 1, and ω)

$\mathbf{let}\ f : \mathbf{int} \to^{(0,0)} \mathbf{int} = \lambda x.1+2+x\ \mathbf{in}\ 7$

⇨ $\mathbf{let}\ f = \langle\rangle\ \mathbf{in}\ 7$

➡ 7

$\mathbf{let}\ f : \mathbf{int} \to^{(0,1)} \mathbf{int} = \lambda x.1+2+x\ \mathbf{in}\ f$

⇨ $\mathbf{let}\ f = \underline{\lambda x}.1+2\underline{+x}\ \mathbf{in}\ f$

➡ $\underline{\lambda x}.3\underline{+x}$

$\mathbf{let}\ f : \mathbf{int} \to^{(0,\omega)} \mathbf{int} = \lambda x.1+2+x\ \mathbf{in}\ \langle f, f\rangle$

⇨ $\mathbf{let}\ f = \mathit{insert\text{-}let}(\underline{\lambda x}.1+2\underline{+x})\ \mathbf{in}\ \langle f, f\rangle$

Examples of Typing (II): Dynamic Uses (0, 1, and ω)

$\mathbf{let}\ f : \mathbf{int} \to^{(0,0)} \mathbf{int} = \lambda x.1+2+x\ \mathbf{in}\ 7$

⇨ $\mathbf{let}\ f = \langle\rangle\ \mathbf{in}\ 7$

➡ 7

$\mathbf{let}\ f : \mathbf{int} \to^{(0,1)} \mathbf{int} = \lambda x.1+2+x\ \mathbf{in}\ f$

⇨ $\mathbf{let}\ f = \underline{\lambda x}.1+2\underline{+x}\ \mathbf{in}\ f$

➡ $\underline{\lambda x}.3\underline{+x}$

$\mathbf{let}\ f : \mathbf{int} \to^{(0,\omega)} \mathbf{int} = \lambda x.1+2+x\ \mathbf{in}\ \langle f, f\rangle$

⇨ $\mathbf{let}\ f = \textit{insert-let}(\underline{\lambda x}.1+2\underline{+x})\ \mathbf{in}\ \underline{(f, f)}$

➡ $\underline{\mathbf{let}\ f = \lambda x}.3\underline{+x\ \mathbf{in}\ (f, f)}$

Example of Typing Rules

For λ-abstractions:

$$\Gamma \succ (s,d) \cdot \Gamma_0$$

$$d \neq \mathbf{0} \Rightarrow s_1 \neq \omega$$

$$\frac{\Gamma_0, x : \rho_1^{(s_1,d_1)} \vdash e : \tau_2}{\Gamma \vdash \lambda x.e : \rho_1^{(s_1,d_1)} \rightarrow^{(s,d)} \tau_2} \textbf{(abs)}$$

Type Inference

- Construct a type derivation
  - assign use variables
  - generate constraints on them
- Solve the constraints

Type Inference

- Construct a type derivation
- Solve the constraints
  - approximate most conservatively
    (every $s = \mathbf{T}$ and every $d = \omega$)
  - refine by iterations
    ($\mathbf{0} \prec_{\mathbf{S}} \omega \prec_{\mathbf{S}} \mathbf{T}$ and $\mathbf{0} \prec_{\mathbf{D}} \mathbf{1} \prec_{\mathbf{D}} \omega$)
    - linear w.r.t. the # of use variables
    - can be stopped at any time

Type Inference

- Construct a type derivation
- Solve the constraints
  - approximate most conservatively
    (every s = $\mathbf{T}$ and every d = ω)
  - refine by iterations
    ($\mathbf{0} \prec_S \omega \prec_S \mathbf{T}$ and $\mathbf{0} \prec_D \mathbf{1} \prec_D \omega$)

N.B. The existence of $\mathbf{T}$
simplified the analysis with $\mathbf{1}$

Overview

- Introduction
- Simple Online PE
- Our Method
- Experiments
- Related Work
- Conclusion

Compared Methods

- Simple Online PE: only **(T,ω)**
  with post-processing for inlining
- Simple Offline PE: **(ω,0)** and **(0,ω)**
  with binding-time improvement *by hand*
- [Sperber-96]: **(ω,0)**, **(0,ω)**, and **(T,ω)**
- Our Method

All are implemented with:

- a cogen approach
- state-based let-insertion

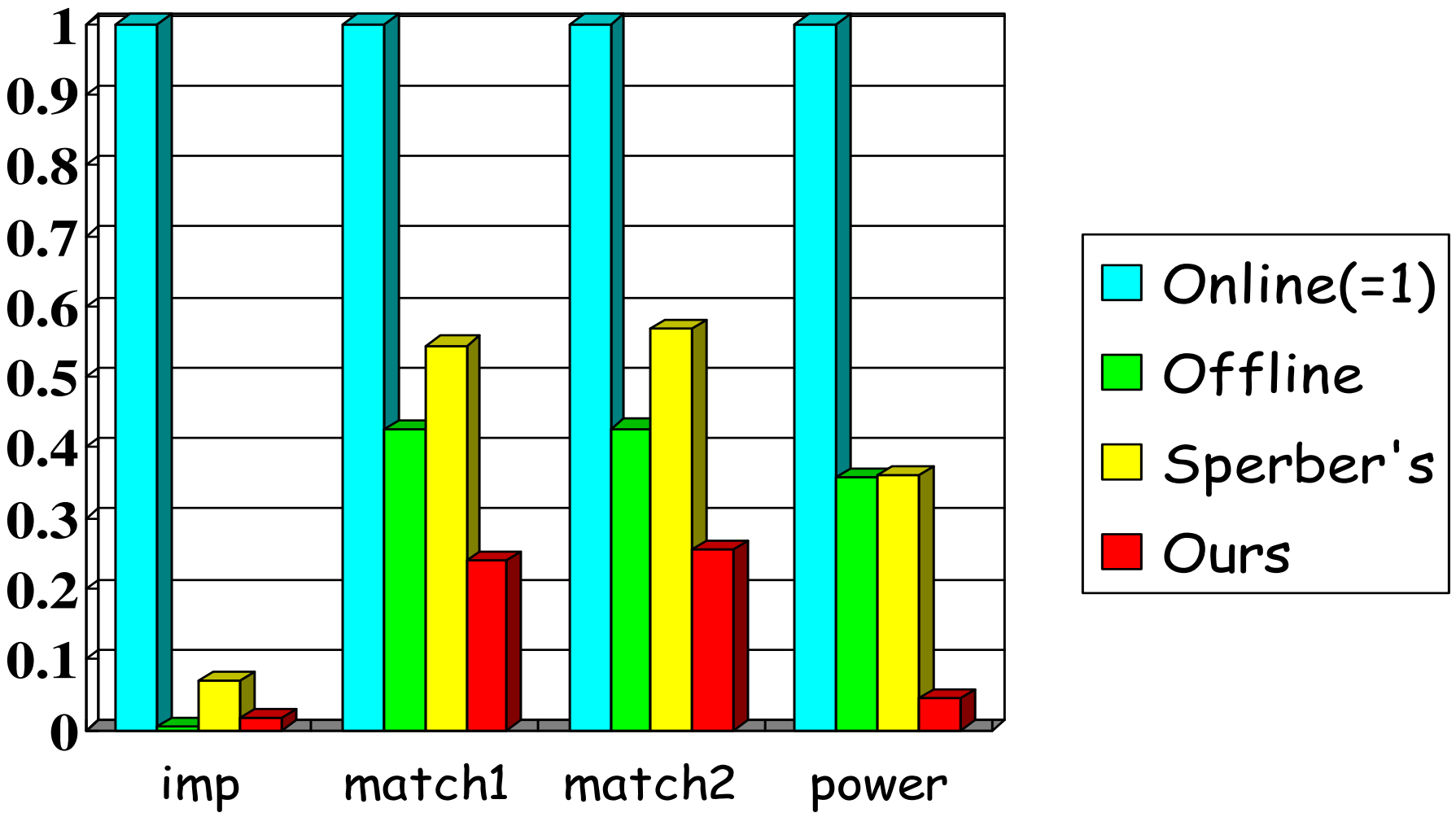
Applications

- imp: an interpreter
  for a simple imperative language
- match1: a pattern matcher
  with the pattern static
- match2: the same pattern matcher
  with the string static
- power

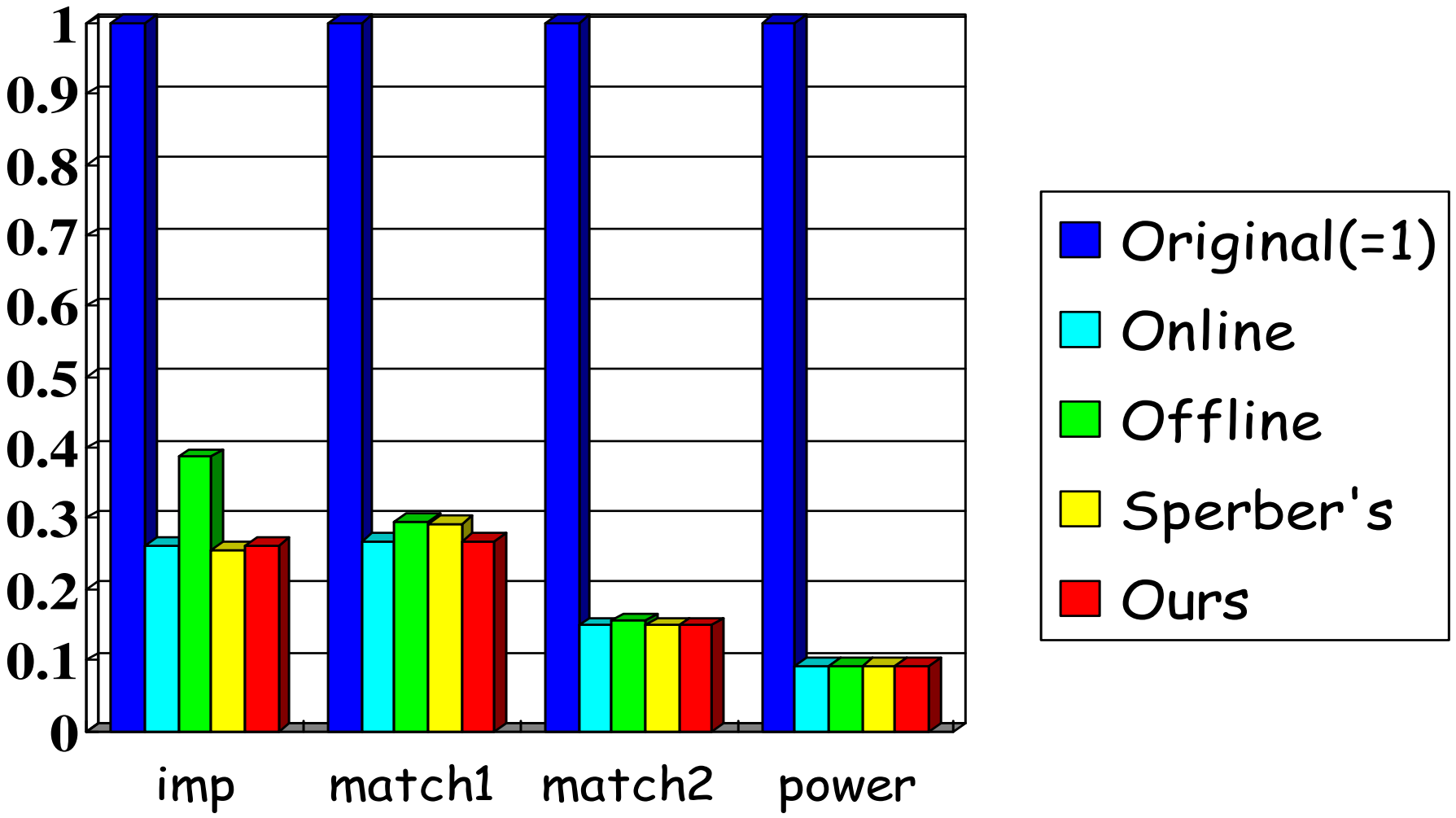
Environment

- Mobile Pentium II 400MHz
- 128MB Main Memory
- Linux 2.2.10
- SML/NJ 110.0.3

Time for Specialization



Time for Specialized Programs



Overview

- Introduction
- Simple Online PE
- Our Method
- Experiments
- Related Work
- Conclusion

Related Work (I)

- [Sperber-96]
  BTA with "unknown" $(\mathbf{T},\omega)$
- [Asai-99]
  BTA with "both" (ω,ω)
- [Bondorf-90]
  "Abstract occurrence counting analysis" to decrease unnecessary let-insertions

Our analysis subsumes all of these.

Related Work (II)

- [Ruf-93] [Sperber-96]
  (Quasi-)self-application for online PE
- [Thiemann-99]
  Systematic derivation of a cogen approach to offline PE

We adopted the cogen approach (which is simpler and faster) into online-and-offline PE.

Overview

- Introduction
- Simple Online PE
- Our Method
- Experiments
- Related Work
- Conclusion

Conclusion

- We presented "hybrid" PE combining:
  - the precision of online PE, and
  - the efficiency of offline PE
- Future work includes:
  - correctness proof
  - experiments with larger programs