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

source program

specialized program

$$PE(p, s) = p_s$$

static input

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

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

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
  - ⇒ 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

dynamic expression
for residualization

type *a* ***symval*** = *a* ***option*** ´ ***exp***

static value
for reduction
(optional)

type *a* ***option*** = ***Some*** *a* | ***None***

Simple Online PE

[| *x* |] = *x*

[| l *x.e* |] = á*Some*(l*x*.[|*e*|]), l*x*´….ñ

[| *e_1e_2* |] = **case** [|*e_1*|]
of á*Some* *s*, _ñ Þ *s* [|*e_2*|]
| á*None*, *d*ñ Þ á*None*, *d* @ **snd** [|*e_2*|]ñ

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

let ***f*** **= l*x*.1+2 in (*f*, *f*)**

➡ **let** ***f*** **= l*x*.3 in (*f*, *f*)**

rather than **(l*x*.3, l*x*.3)**

Simple Online PE

[| *x* |]   = *x*

[| l *x.e* |] = á*Some*(l*x*.[|*e*|]), l*x*´….ñ

[| e_1 e_2 |] = **case** [|e_1|]
of á*Some* *s*, _ñ Þ *s* [|e_2|]
| á*None*, *d*ñ Þ á*None*, *d* @ **snd** [|e_2|]ñ

Simple Online PE
with Let-Insertion

[| *x* |] = *x*

[| l*x.e* |] = á*Some*(l*x.delimit-let*([|*e*|])),
insert-let(l*x*´….)ñ

[| e_1 e_2 |] = **case** [|e_1|]
of á*Some s*, _ñ Þ *s* [|e_2|]
| á*None*, *d*ñ Þ
á*None*, *insert-let*(*d* @ **snd** [|e_2|])ñ

Simple Online PE is Slow

because of:

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

[| (l *x*.*x*)3 |] =

case á*Some*(l *x*.*x*), *insert-let*(l *x*´....)ñ

of á*Some*(*s*), _ñ Þ *s* 3

| á*None*, *d*ñ Þ ...

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

[| (l *x*.*x*)3 |] =
case á***Some***(l ***x*.*x***), ***insert-let***(l*x*´….)ñ
of á***Some***(***s***), **_**ñ Þ ***s* 3**
| á***None***, ***d***ñ Þ …

Simple Online PE is Slow

because of:

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

[| (l *x*.*x*)3 |] =

case á l ***x*.*x***, ***let-insert***(l *x*´….)ñ

of á*s*, _ñ Þ *s* **3**

Simple Online PE is Slow

because of:

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

[| (l*x*.*x*)3 |] =

case ál*x*.*x*, ***let-insert***(l*x*´….)ñ

of á*s*, _ñ Þ *s* **3**

Simple Online PE is Slow

because of:

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

[| (l*x*.*x*)3 |] =

case l*x*.*x*

of *s* Þ *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

r (raw type) ::= $\boldsymbol{b_i}$ | $t_1 \to t_2$ | $t_1 \times t_2$

t (annotated type) ::= $r^{(\boldsymbol{s},\boldsymbol{d})}$

s (static use) ::= **0** (never) | w (always)

| **T** (sometimes)

"Whether a static value is available"

d (dynamic use) ::= **0** (never)

| **1** (at most once) | w (any number of times)

"How many times
a dynamic expression remains"

Types

r (raw type) $::=$ $\boldsymbol{b_i}$ | $t_1 \to t_2$ | $t_1 \times t_2$

t (annotated type) $::=$ $r^{(\boldsymbol{s},\boldsymbol{d})}$

Types

r (raw type) ::= $\boldsymbol{b_i}$ | $t_1 \rightarrow t_2$ | $t_1 \times t_2$

t (annotated type) ::= $r^{(\boldsymbol{s},\boldsymbol{d})}$

$\boldsymbol{s}$ (static use) ::= **0** (never) | w (always)
| **T** (sometimes)

"Whether a static value is available"

Types

r (raw type) ::= $\boldsymbol{b_i}$ | $t_1 \rightarrow t_2$ | $t_1 \times t_2$

t (annotated type) ::= $r^{(\boldsymbol{s},\boldsymbol{d})}$

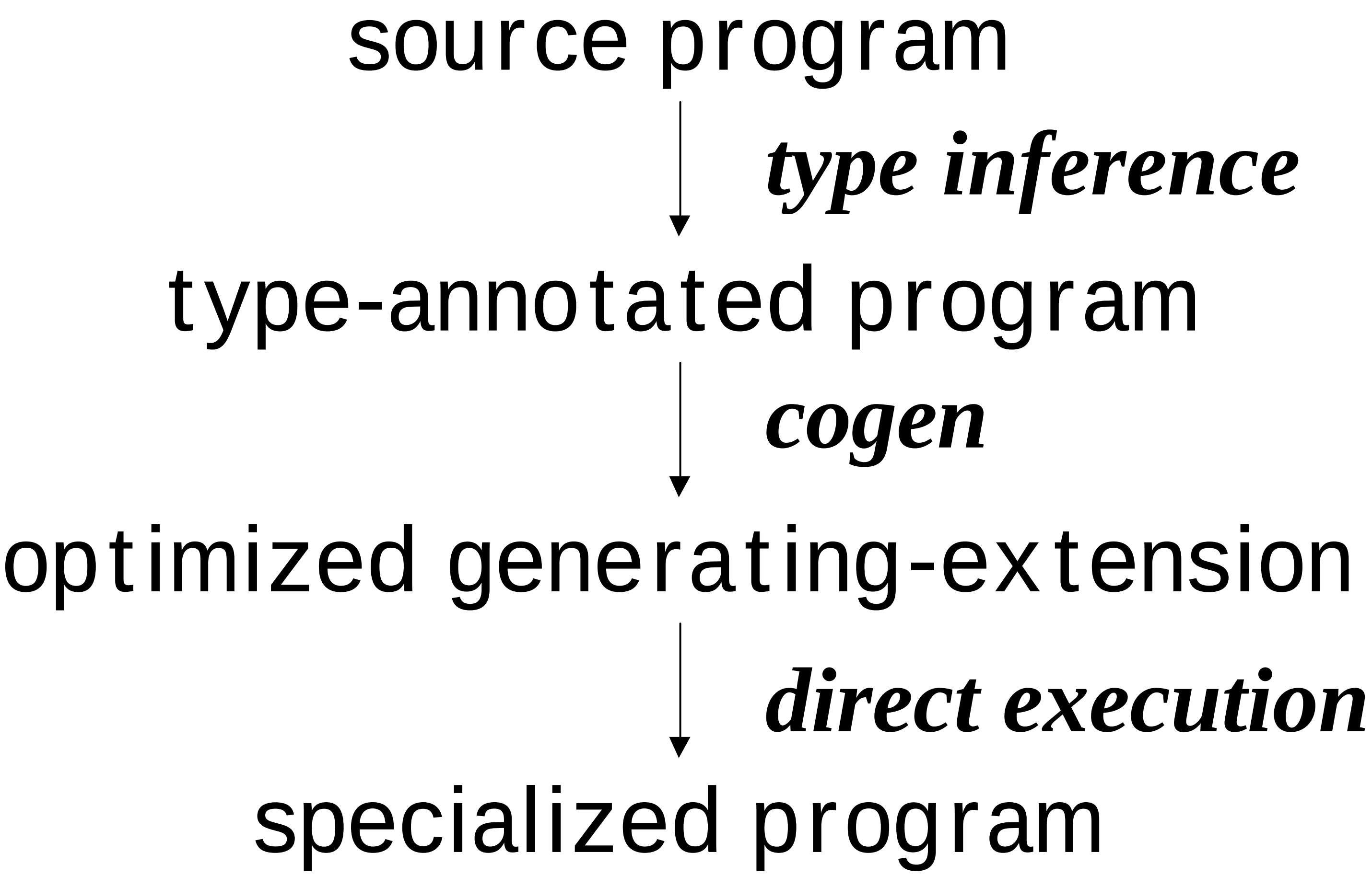
$\boldsymbol{s}$ (static use) ::= **0** (never) | w (always)
| **T** (sometimes)

"Whether a static value is available"

$\boldsymbol{d}$ (dynamic use) ::= **0** (never)
| **1** (at most once) | w (any number of times)

"How many times a dynamic expression remains"

Framework of our Method



source program

type inference

type-annotated program

cogen

optimized generating-extension

direct execution

specialized program

Examples of Typing (1): Static Uses (**0**, w, and **T**)

l*g*. let *f* : int® $^{(0,1)}$*a* = *g* in *f* 3

⇨ l*g*. let *f* = *g* in *f*@3

➡ l*g*. g@3

l*g*. let *f* : int® $^{(w,0)}$int = l*x*.*x* in *f* 3

⇨ l*g*. let *f* = l*x*.*x* in *f* 3

➡ l*g*. 3

l*g*. let *f* : int® $^{(T,1)}$int = (if *true* then l*x*.*x* else *g*)

in f 3

⇨ l*g*. let *f* = (if *true* then á*Some*(l*x*.*x*), ¼ñ

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

l***g*. let *f* = *g* in *f* 3**

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

l$\boldsymbol{g}$. **let** $\boldsymbol{f}$ **: int**®$^{(\mathbf{0},\mathbf{1})}$$a$ **=** $\boldsymbol{g}$ **in** $\boldsymbol{f}$ **3**

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

$\mathrm{l}\, \boldsymbol{g}.\ \mathbf{let}\ \boldsymbol{f} : \mathbf{int}\, ®^{(\mathbf{0},1)} a\ = \boldsymbol{g}\ \mathbf{in}\ \boldsymbol{f}\ \mathbf{3}$

⇨ $\underline{\mathrm{l}\, \boldsymbol{g}}.\ \mathbf{let}\ \boldsymbol{f} = \underline{\boldsymbol{g}}\ \mathbf{in}\ \boldsymbol{f}\underline{\mathbf{@}}\mathbf{3}$

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

l*g*. **let** *f* **: int**®$^{(\mathbf{0},1)}$*a* **=** ***g*** **in** *f* **3**

⇨ $\underline{\text{l}\boldsymbol{g}}$**.** **let** *f* **=** *g* **in** ***f***$\underline{\textbf{@}}$**3**

➡ $\underline{\text{l}\boldsymbol{g}}$**.** $\underline{\textbf{g@}}$**3**

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

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

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

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

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

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

l*g*. let *f* : int®$^{(0,1)}$*a* = *g* in *f* 3

⇨ l*g*. let *f* = *g* in *f*@3

➡ l*g*. g@3

l*g*. let *f* : int®$^{(\mathrm{W},0)}$int = l*x*.*x* in *f* 3

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

l*g*. let *f* : int®$^{(0,1)}$*a* = *g* in *f* 3

⇨ $\underline{\text{l}g}$. let *f* = *g* in *f*$\underline{@}$3

➡ $\underline{\text{l}g}$. $\underline{g@}$3

l*g*. let *f* : int®$^{(\text{w},0)}$int = l*x*.*x* in *f* 3

⇨ $\underline{\text{l}g}$. let *f* = l*x*.*x* in *f* 3

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

l*g*. let *f* : int® $^{(0,1)}$*a* = *g* in *f* 3

⇨ $\underline{\text{l}g}$. let *f* = *g* in *f*$\underline{\text{@}}$3

➡ $\underline{\text{l}g}$. $\underline{\text{g@}}$3

l*g*. let *f* : int® $^{(W,0)}$int = l*x*.*x* in *f* 3

⇨ $\underline{\text{l}g}$. let *f* = l*x*.*x* in *f* 3

➡ $\underline{\text{l}g}$. 3

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

l*g*. let *f* : int®$^{(0,1)}$*a* = *g* in *f* 3 ➡ l*g*. g 3

l*g*. let *f* : int®$^{(w,0)}$int = l*x*.*x* in *f* 3 ➡ l*g*. 3

l*g*. let *f* = (if *true* then l*x*.*x* else *g*)

in f 3

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

l*g*. let *f* : int®$^{(0,1)}$*a* = *g* in *f* 3 ➡ **l*g*. g 3**

l*g*. let *f* : int®$^{(w,0)}$int = l*x*.*x* in *f* 3 ➡ **l*g*. 3**

l*g*. let *f* : int®$^{(T,1)}$int = (if *true* then l*x*.*x* else *g*)

in f 3

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

l*g*. let *f* : int® $^{(0,1)}$*a* = *g* in *f* 3 ➡ l*g*. g 3

l*g*. let *f* : int® $^{(w,0)}$int = l*x*.*x* in *f* 3 ➡ l*g*. 3

l*g*. let *f* : int® $^{(T,1)}$int = (if *true* then l*x*.*x* else *g*)

in f 3

⇨ **l*g*. let *f* = (if *true* then á*Some*(l*x*.*x*), ¼ñ**

else á*None*, gñ)

in (case *f* of á*Some* *s*, _ñ Þ *s* 3

| á*None*, *d*ñ Þ *d*@3)

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

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

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

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

$\quad \mathbf{in\ f\ 3}$

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

$\qquad \mathbf{else}\ \langle \mathit{None},\ \underline{g}\rangle)$

$\quad \mathbf{in}\ (\mathbf{case}\ f\ \mathbf{of}\ \langle \mathit{Some}\ s,\ \_\rangle \Rightarrow s\ 3$

$\qquad |\ \langle \mathit{None},\ d\rangle \Rightarrow d\underline{@}3)$

➡ $\underline{\lambda g}.\ 3$

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

let ***f*** **: int® $^{(0,0)}$int =** l***x*.1+2+*x* in 7**

⇨ **let** ***f*** **=** áñ **in 7**

➡ **7**

let ***f*** **: int® $^{(0,1)}$int =** l***x*.1+2+*x* in** ***f***

⇨ **let** ***f*** **=** l***x*.1+2+*x* in** ***f***

➡ l***x*.3+*x***

let ***f*** **: int® $^{(0,w)}$int =** l***x*.1+2+*x* in** á***f*, *f***ñ

⇨ **let** ***f*** **=** ***insert-let*(**l***x*.1+2+*x*) in (*f*, *f*)**

➡ **let** ***f*´** **=** l***x*.3+x in (*f*´, *f*´)**

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

let *f* = l***x*.1+2+*x* in 7**

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

let ***f*** **: int®$^{(0,0)}$int =** l***x*****.1+2+*****x*** **in 7**

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

let ***f*** **: int®**$^{(0,0)}$**int** = l***x*****.1+2+*x* in 7**

⇨ **let *f*** = áñ **in 7**

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

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

⇨ **let** ***f*** **=** áñ **in 7**

➡ **7**

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

let ***f*** **: int®**$^{(0,0)}$**int =** l***x*****.1+2+*x* in 7**

⇨ **let** ***f*** **=** áñ **in 7**

➡ **7**

let ***f*** **=** l***x*****.1+2+*x* in** ***f***

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

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

⇨ **let** f = ⟨⟩ **in** 7

➡ 7

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

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

let ***f*** **: int®$^{(0,0)}$int =** l***x*****.1+2+*****x*** **in 7**

⇨ **let** ***f*** = áñ **in 7**

➡ 7

let ***f*** **: int®$^{(0,1)}$int =** l***x*****.1+2+*****x*** **in** ***f***

⇨ **let** ***f*** **=** l***x*****.1+2+*****x*** **in** ***f***

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

let *f* **: int**® $^{(0,0)}$**int =** l*x*.1+2+*x* **in 7**

⇨ **let** *f* = áñ **in 7**

➡ **7**

let *f* **: int**® $^{(0,1)}$**int =** l*x*.1+2+*x* **in** *f*

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

➡ $\underline{\text{l}x}$.3$\underline{+x}$

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

let *f* : int® $^{(0,0)}$int = l***x*.1+2+*x* in 7**

⇨ **let *f* =** áñ **in 7**

➡ **7**

let *f* : int® $^{(0,1)}$int = l***x*.1+2+*x* in *f***

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

➡ $\underline{\text{l}x}$.3$\underline{+x}$

let *f* = l***x*.1+2+*x* in** á***f*, *f***ñ

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

let *f* : int® $^{(0,0)}$int = l*x*.1+2+*x* in 7

⇨ **let *f* = áñ in 7**

➡ **7**

let *f* : int® $^{(0,1)}$int = l*x*.1+2+*x* in *f*

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

➡ $\underline{\text{l}x}$**.3**$\underline{+x}$

let *f* : int® $^{(0,\text{W})}$int = l*x*.1+2+*x* in á*f*, *f*ñ

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

let *f* **: int® $^{(0,0)}$int =** l*x*.1+2+*x* **in 7**

⇨ **let** *f* = áñ **in 7**

➡ **7**

let *f* **: int® $^{(0,1)}$int =** l*x*.1+2+*x* **in** *f*

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

➡ $\underline{\text{l}x}$.3$\underline{+x}$

let *f* **: int® $^{(0,\text{w})}$int =** l*x***.1+2+***x* **in** á*f*, *f*ñ

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

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

let *f* : **int**®$^{(0,0)}$**int** = l*x*.1+2+*x* **in 7**

⇨ **let** *f* = áñ **in 7**

➡ **7**

let *f* : **int**®$^{(0,1)}$**int** = l*x*.1+2+*x* **in** *f*

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

➡ $\underline{\mathrm{l}x}$.3$\underline{+x}$

let *f* : **int**®$^{(0,\mathrm{w})}$**int** = l*x*.1+2+*x* **in** á*f*, *f*ñ

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

➡ $\underline{\textbf{let}}$ $\underline{f}$ = $\underline{\mathrm{l}x}$.3$\underline{+x}$ $\underline{\textbf{in}}$ $\underline{(f, f)}$

Example of Typing Rules

For λ-abstractions:

$$G \succ (s,d) \between G_0$$

$$d \;{}^{1}\; 0 \;\text{Þ}\; s_1 \;{}^{1}\; w$$

$$\frac{G_0, x : r_1^{(s_1,d_1)} \vdash e : t_2}{G \vdash l\, x.e : r_1^{(s_1,d_1)} \;\text{®}^{(s,d)}\; t_2}\;\textbf{(abs)}$$

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 $\boldsymbol{s} = \mathbf{T}$ and every $\boldsymbol{d} = \mathrm{w}$)
  - refine by iterations ($\mathbf{0} \prec_{\mathbf{S}} \mathrm{w} \prec_{\mathbf{S}} \mathbf{T}$ and $\mathbf{0} \prec_{\mathbf{D}} \mathbf{1} \prec_{\mathbf{D}} \mathrm{w}$)
    - 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 $\boldsymbol{s} = \mathbf{T}$ and every $\boldsymbol{d} = \mathrm{w}$)
  - refine by iterations ($\mathbf{0} \prec_{\mathbf{S}} \mathrm{w} \prec_{\mathbf{S}} \mathbf{T}$ and $\mathbf{0} \prec_{\mathbf{D}} \mathbf{1} \prec_{\mathbf{D}} \mathrm{w}$)

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

All are implemented with:

- a cogen approach
- state-based let-insertion

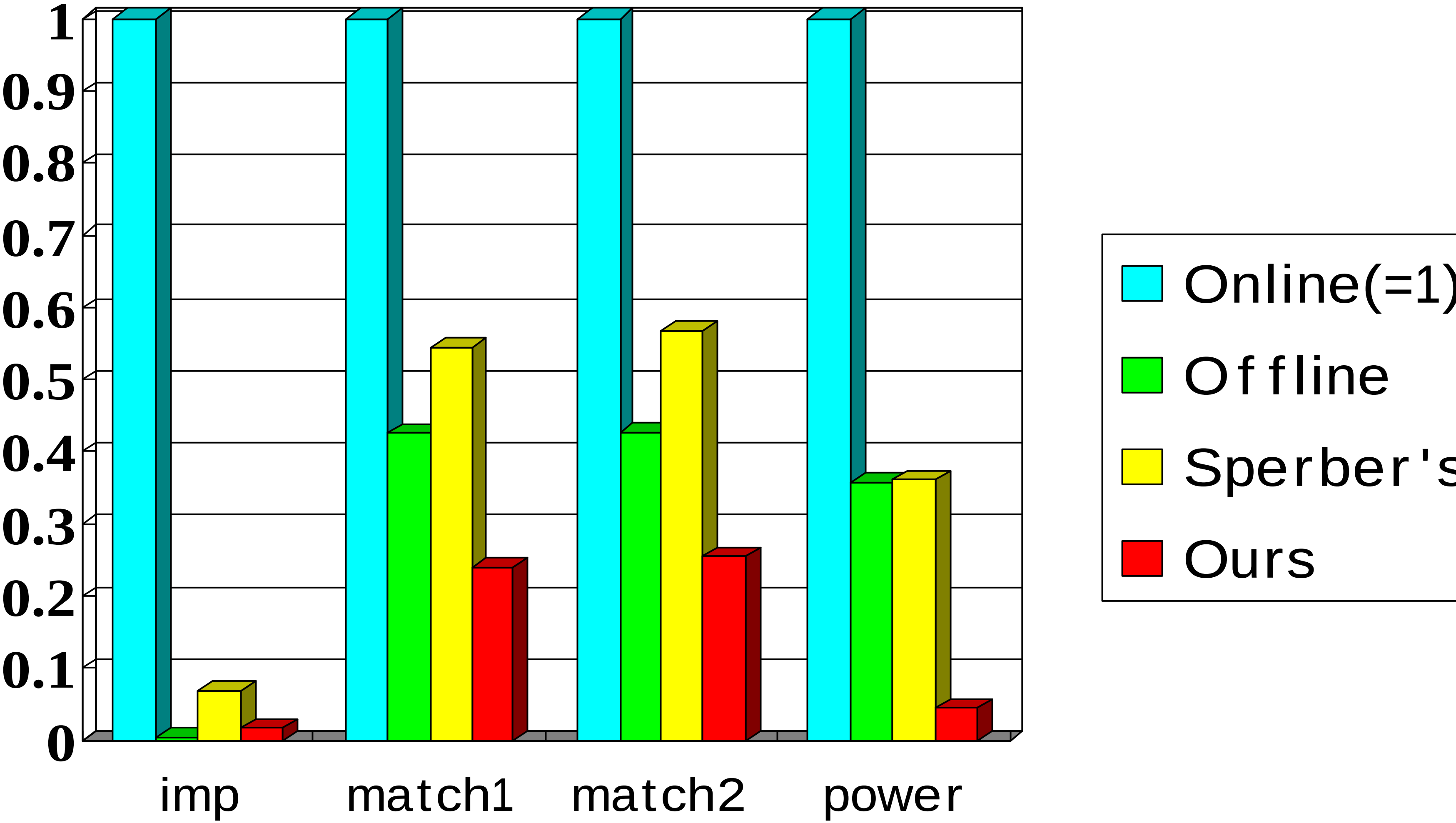
Applications

- 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

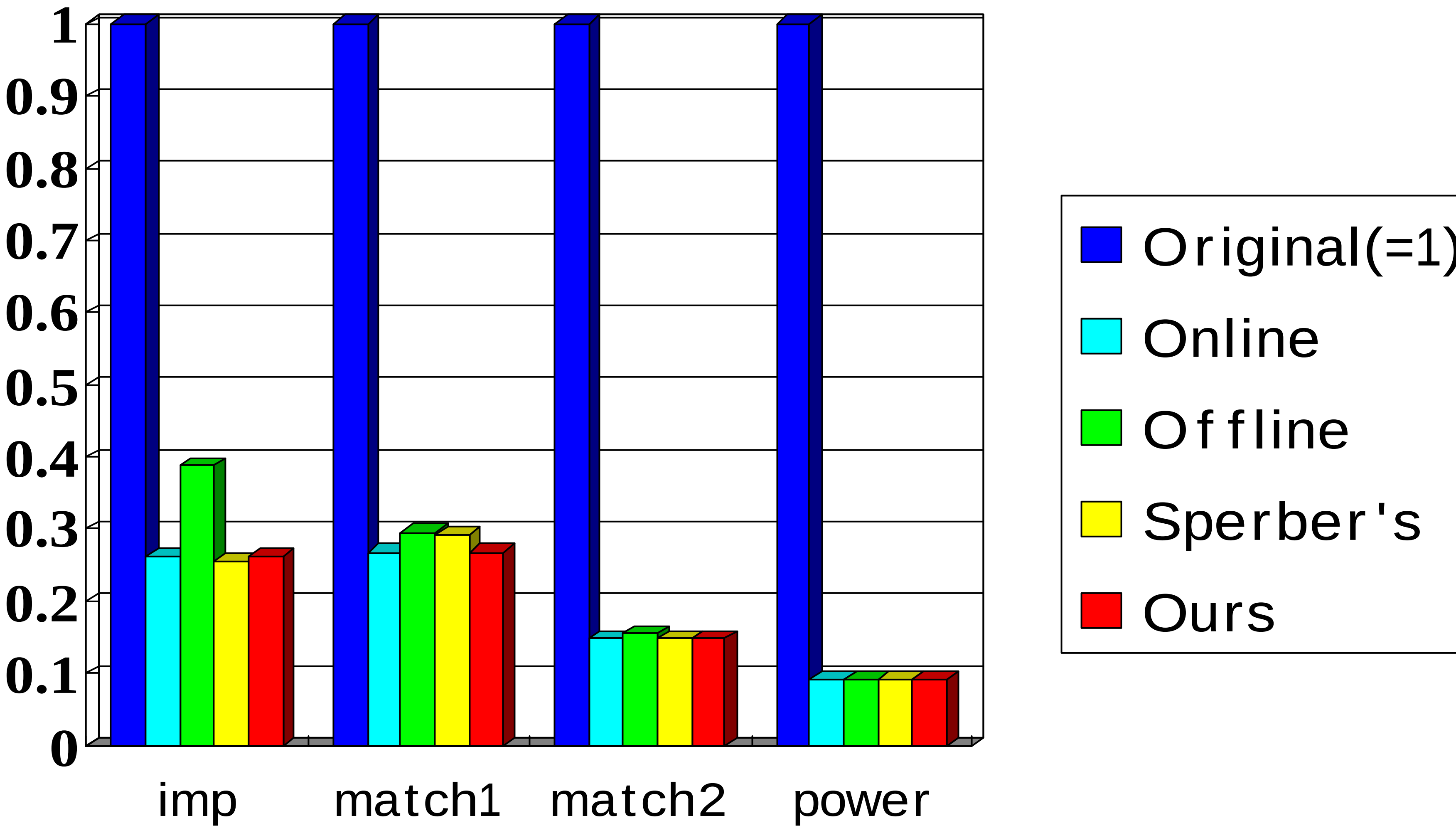
Environment

- 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" **(T,**w**)**

- [Asai-99]

  BTA with "both" **(**w**,**w**)**

- [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