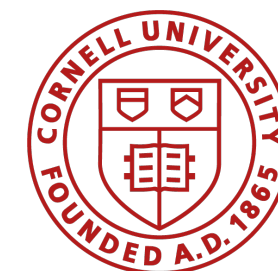


Effects and Coeffects in Call-By-Push-Value

Cassia Torczon, Emmanuel Suárez Acevedo, Shubh Agrawal,
Joey Velez-Ginorio, Stephanie Weirich

OOPSLA 2024



Cornell University



Dead code elimination

```
1 | let  $x$  = "hello" in  
2 | print "world"
```

Dead code elimination

```
1 | let x = "hello" in  
2 | print "world"
```

can the first line be removed?

Dead code elimination

```
1 | let x = print "hello" in  
2 | print "world"
```

Dead code elimination

```
1 | let  $x = e_1$  in  
2 |  $e_2$ 
```

Dead code elimination

```
1 | let  $x = e_1$  in  
2 |  $e_2$ 
```

Dead code elimination

```
1 | let  $x = e_1$  in  
2 |  $e_2$ 
```

is the code bound to x pure?

Dead code elimination

1 | let $x = e_1$ in
2 | e_2

is the code bound to x pure?

is x unused in the code that follows?

Effects

$x : \text{string}, y : \text{nat} \vdash \mathbf{print} \ x : \text{unit}$

Effects

$x : \text{string}, y : \text{nat} \vdash \text{print } x : \text{unit}$ ^{Output}

Effects

$$\Gamma \vdash e :^{\phi} \tau$$

Effects

IO effects

$$\Gamma \vdash e :^{\phi} \tau$$

Effects

IO effects

Running time

$$\Gamma \vdash e :^{\phi} \tau$$

Effects

IO effects

Running time

$$\Gamma \vdash e :^{\phi} \tau$$

Exceptions

... and **Coeffects**

$x : \text{string}, y : \text{nat} \vdash \text{print } x : \text{unit}$

... and **Coeffects**

$x :^1 \text{string}, y :^0 \text{nat} \vdash \text{print } x : \text{unit}$

... and Coeffects

$$x_1 :^{q_1} \tau_1, x_2 :^{q_2} \tau_2 \vdash e : \tau$$

... and **Coeffects**

$$\gamma \cdot \Gamma \vdash e : \tau$$

... and Coeffects

Relevance

$$\gamma \cdot \Gamma \vdash e : \tau$$

... and Coeffects

Relevance

Resources

$$\gamma \cdot \Gamma \vdash e : \tau$$

... and Coeffects

Relevance

Resources

$$\gamma \cdot \Gamma \vdash e : \tau$$

Differential privacy

... and Coeffects

Relevance

Resources

$$\gamma \cdot \Gamma \vdash e : \tau$$

Differential privacy

Information flow

Effects and Coeffects

```
1 | let  $x = e_1$  in  
2 |  $e_2$ 
```

Effect: is the code bound to x pure?

Coeffect: is x unused in the code that follows?

Statically tracking **effects**

```
1 | let  $x = \text{print "hello"}$  in  
2 |  $x; x$ 
```


Statically tracking **effects**

```
1 | let  $x = \text{print "hello"}$  in  
2 |  $x; x$ 
```

Expected **effect**?

Statically tracking **effects**

→ 1 | let $x = \text{print "hello"}$ in
2 | $x; x$

Expected **effect**?

Statically tracking **effects**

→ 1 | let $x = ()$ in
2 | $x; x$

Expected **effect**?

hello

Statically tracking **effects**

$x \mapsto ()$

1 | `let x = print "hello" in`

 2 | `x; x`


Expected **effect**?

hello

Statically tracking **effects**

$x \mapsto ()$

1 | let $x = \text{print "hello"}$ in

 2 | $() ; ()$

Expected **effect**?

hello

Statically tracking **effects**

```
1 | let  $x = \text{print "hello"}$  in  
2 |  $x; x$ 
```

Expected **effect**?

Statically tracking **effects**

→ 1 | let $x = \text{print "hello"}$ in
2 | $x; x$

Expected **effect**?

Statically tracking **effects**

$x \mapsto \text{print } \text{"hello"}$

1 | `let x = print "hello" in`


 2 | `x; x`

Expected **effect**?

Statically tracking **effects**

$x \mapsto \text{print } \text{"hello"}$

1 | `let x = print "hello" in`

 2 | `print "hello"; print "hello"`

Expected **effect**?

Statically tracking **effects**

$x \mapsto \text{print "hello"}$

1 | let $x = \text{print "hello"}$ in

 2 | (); ()

Expected **effect**?

hello
hello

Statically tracking effects

**Intermediate
representation**

Statically tracking effects

**Intermediate
representation**

**Explicit
evaluation
order**

... in Call-By-Push-Value (CBPV)

... in Call-By-Push-Value (CBPV)

A subsuming paradigm

... in Call-By-Push-Value (CBPV)

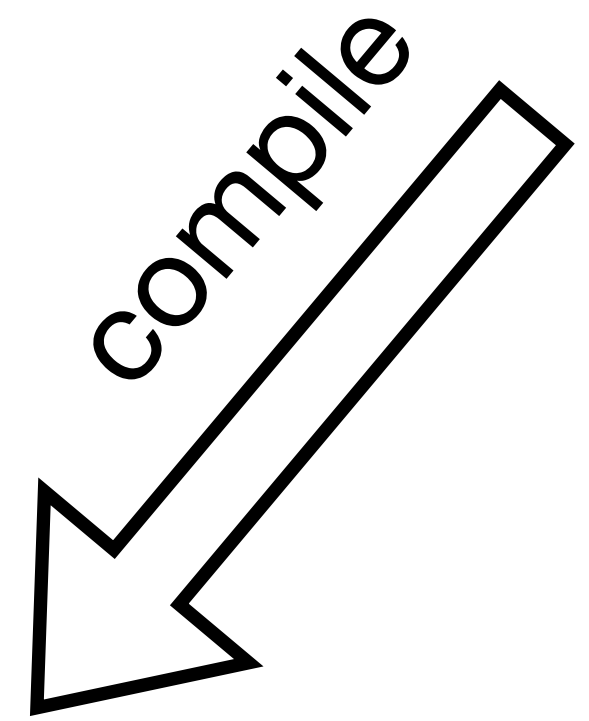
A subsuming paradigm

```
1 | let  $x = \text{print}$  "hello" in  
2 |  $x; x$ 
```

... in Call-By-Push-Value (CBPV)

A subsuming paradigm

```
1 | let x = print "hello" in  
2 | x; x
```

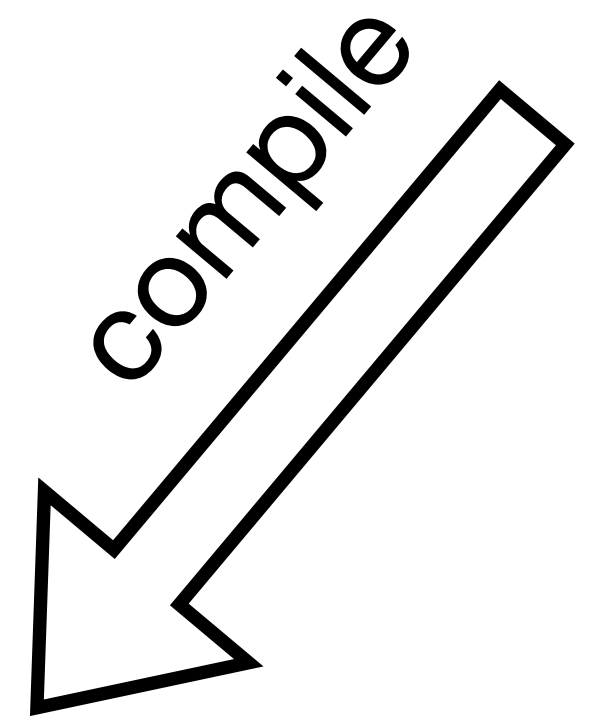


hello

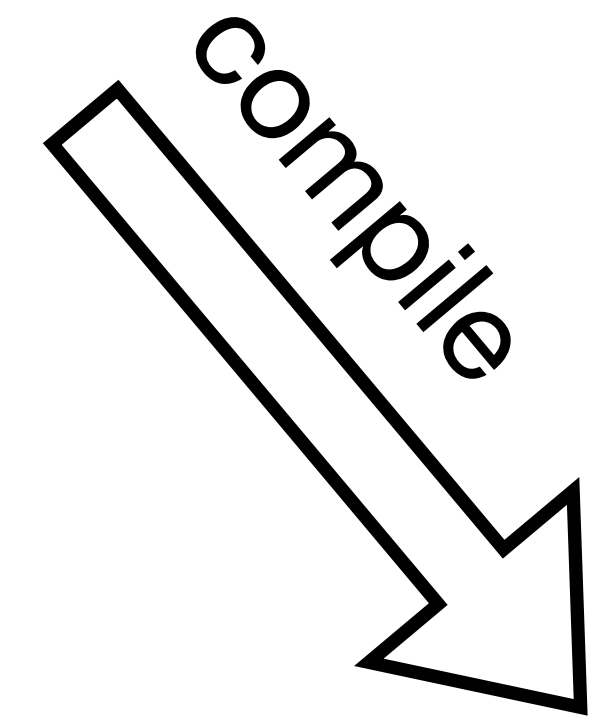
... in Call-By-Push-Value (CBPV)

A subsuming paradigm

```
1 | let x = print "hello" in  
2 | x; x
```



hello



hello
hello

... in Call-By-Push-Value (CBPV)

A subsuming paradigm

Expression

... in Call-By-Push-Value (CBPV)

A subsuming paradigm

Value

Computation

... in Call-By-Push-Value (CBPV)

A subsuming paradigm

Value

“is”

Computation

... in Call-By-Push-Value (CBPV)

A subsuming paradigm

Value

“is”

A string is a value



Computation

... in Call-By-Push-Value (CBPV)

A subsuming paradigm

Value

“is”

A string is a value



Computation

“does”

... in Call-By-Push-Value (CBPV)

A subsuming paradigm

Value

“is”

A string is a value

Computation

“does”

Printing a string is a
computation

```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 1


```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 1

hello



```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 1

hello



```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 1

hello

```
1 | x ← print "hello"
```



```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 1

hello

```
1 | x ← print "hello"
```

"sequence"



```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 1

hello

```
1 | x ← print "hello"
```

```
⊢ print "hello": F unit
```



```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 1

hello

```
1 | x ← print "hello"
```

```
⊢ print "hello": F unit
```

"Returner"



```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 1

hello

```
1 | x ← print "hello"
```

value
 $x : \text{unit}$

$\vdash \text{print "hello"} : \mathbf{F} \text{ unit}$



```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 1

hello

```
1 | x ← print "hello"
```

$$\frac{\vdash \text{print "hello"} : \mathbf{F} \text{ unit} \quad x : \text{unit} \vdash N : B}{\vdash x \leftarrow \text{print "hello"} ; N : B}$$
$$\vdash x \leftarrow \text{print "hello"} ; N : B$$



```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 1

hello

```
1 | x ← print "hello"
```

$\vdash \text{print "hello"} : \mathbf{F} \text{ unit} \quad x : \text{unit} \vdash N : B$

$\vdash x \leftarrow \text{print "hello"} : N : B$



```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 1

hello

```
1 | x ← print "hello"
```



```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 1

hello

```
1 | x ← print "hello"  
2 | _ ← x  
3 | x
```



```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 1

hello

```
1 | x ← print "hello"  
2 | _ ← x  
3 | x
```

"SEQUENCE"



```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 1

hello

```
1 | x ← print "hello"  
2 | _ ← x  
3 | x
```



```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 1

hello

```
1 | x ← print "hello"  
2 | _ ← x  
3 | x
```

$x : \text{unit}$ *value*



```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 1

hello

```
1 | x ← print "hello"  
2 | _ ← return x  
3 | return x
```

$x : \text{unit}$ *value*



```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 1

hello

```
1 | x ← print "hello"  
2 | _ ← return x  
3 | return x
```

$\vdash x : \text{unit}$

$\vdash \text{return } x : \mathbf{F} \text{ unit}$


```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 1

hello

```
1 | x ← print "hello"  
2 | _ ← return x  
3 | return x
```

Expected effect?

```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 1

hello

→

```
1 | x ← print "hello"  
2 | _ ← return x  
3 | return x
```

Expected effect?

```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 1

hello

→

```
1 | x ← return ()  
2 | _ ← return x  
3 | return x
```

Expected effect?

hello

```
1 | let x = print "hello" in
2 | x; x
```

Compilation 1

hello

$x \mapsto ()$

```
1 | x ← print "hello"
```

→ 2 | _ ← return x

```
3 | return x
```

Expected effect?

hello

```
1 | let x = print "hello" in
2 | x; x
```

Compilation 1

hello

$x \mapsto ()$



```
1 | x ← print "hello"
2 | _ ← return ()
3 | return x
```

Expected effect?

hello

```
1 | let x = print "hello" in
2 | x; x
```

Compilation 1

hello

$x \mapsto ()$

1 | $x \leftarrow$ print "hello"

2 | $_ \leftarrow$ return x

→ 3 | return x

Expected effect?

hello

```
1 | let x = print "hello" in
2 | x; x
```

Compilation 1

hello

$x \mapsto ()$

1 | $x \leftarrow$ print "hello"

2 | $_ \leftarrow$ return x

→ 3 | return ()

Expected effect?

hello

```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 1

hello

```
1 | x ← print "hello"  
2 | _ ← return x  
3 | return x
```

Expected **effect**?

hello


```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 2

```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 2

hello
hello



```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 2
hello
hello

```
1 | x ← print "hello"
```



```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 2

```
hello  
hello
```

```
1 | x ← {print "hello"}
```



```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 2

hello
hello

```
1 | x ← {print "hello"}
```

"suspend"



```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 2
hello
hello

```
1 | x ← {print "hello"}
```

"suspend"

\vdash print "hello": **F** unit



```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 2
hello
hello

```
1 | x ← {print "hello"}
```

$\vdash \text{print "hello"} : \mathbf{F} \text{ unit}$

"suspender"

$\vdash \{ \text{print "hello"} \} : \mathbf{U} (\mathbf{F} \text{ unit})$



```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 2
hello
hello

```
1 | x ← {print "hello"}
```

$\vdash \{ \text{print "hello"} \} : \mathbf{U} (\mathbf{F} \text{ unit})$ *value*



```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 2

```
hello  
hello
```

```
1 | x ← return {print "hello"}
```



```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 2
hello
hello

```
1 | x ← return {print "hello"}
```



```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 2
hello
hello

```
1 | x ← return {print "hello"}  
2 | _ ← x  
3 | x
```



```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 2
hello
hello

```
1 | x ← return {print "hello"}  
2 | _ ← x  
3 | x
```

$x : \mathbf{U}$ (\mathbf{F} unit) *value*



```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 2
hello
hello

```
1 | x ← return {print "hello"}  
2 | _ ← x  
3 | x
```

$x : \mathbf{U} (\mathbf{F} \text{ unit})$

suspended computation



```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 2
hello
hello

```
1 | x ← return {print "hello"}  
2 | _ ← x!  
3 | x!
```

$x : \mathbf{U}$ (\mathbf{F} unit)



```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 2
hello
hello

```
1 | x ← return {print "hello"}  
2 | _ ← x ! "force"  
3 | x !
```

$x : \mathbf{U} (\mathbf{F} \text{ unit})$



```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 2
hello
hello

```
1 | x ← return {print "hello"}  
2 | _ ← x!  
3 | x!
```

$\vdash x : \mathbf{U} (\mathbf{F} \text{ unit})$



$\vdash x ! : \mathbf{F} \text{ unit}$


```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 2
hello
hello

```
1 | x ← return {print "hello"}  
2 | _ ← x!  
3 | x!
```

Expected effect?

```
1 | let x = print "hello" in
2 | x; x
```

Compilation 2
hello
hello

→

```
1 | x ← return {print "hello"}
2 | _ ← x!
3 | x!
```

Expected effect?

```
1 | let x = print "hello" in
2 | x; x
```

Compilation 2
hello
hello

$x \mapsto \{\text{print "hello"}\}$

1 | $x \leftarrow \text{return } \{\text{print "hello"}\}$

 2 | $_ \leftarrow x!$

3 | $\bar{x}!$


Expected **effect**?

```
1 | let x = print "hello" in
2 | x; x
```

Compilation 2
hello
hello

$x \mapsto \{\text{print "hello"}\}$

1 | $x \leftarrow \text{return } \{\text{print "hello"}\}$

 2 | $_ \leftarrow \{\text{print "hello"}\}!$

3 | $x!$

Expected **effect**?

```
1 | let x = print "hello" in
2 | x; x
```

Compilation 2
hello
hello

$x \mapsto \{\text{print "hello"}\}$

1 | $x \leftarrow \text{return } \{\text{print "hello"}\}$



2 | $_ \leftarrow \text{print "hello"}$

3 | $x!$

Expected effect?

```
1 | let x = print "hello" in
2 | x; x
```

Compilation 2
hello
hello

$x \mapsto \{\text{print "hello"}\}$

1 | $x \leftarrow \text{return } \{\text{print "hello"}\}$

→ 2 | $_ \leftarrow \text{return } ()$

3 | $\bar{x} !$

Expected effect?

hello

```
1 | let x = print "hello" in
2 | x; x
```

Compilation 2
hello
hello

$x \mapsto \{\text{print "hello"}\}$

1 | $x \leftarrow \text{return } \{\text{print "hello"}\}$

2 | $_ \leftarrow x!$



3 | $x!$

Expected effect?

hello

```
1 | let x = print "hello" in
2 | x; x
```

Compilation 2
hello
hello

$x \mapsto \{\text{print "hello"}\}$

```
1 | x ← return {print "hello"}
```

```
2 | _ ← x!
```

→ 3 | $\{\text{print "hello"}\}!$

Expected **effect**?

hello



```
1 | let x = print "hello" in
2 | x; x
```

Compilation 2
hello
hello

$x \mapsto \{\text{print "hello"}\}$

1 | $x \leftarrow \text{return } \{\text{print "hello"}\}$

2 | $_ \leftarrow x!$

 3 | print "hello"

Expected effect?

hello

```
1 | let x = print "hello" in
2 | x; x
```

Compilation 2
hello
hello

$x \mapsto \{\text{print "hello"}\}$

```
1 | x ← return {print "hello"}
2 | _ ← x!
3 | return ()
```



Expected effect?

hello
hello

```
1 | let x = print "hello" in  
2 | x; x
```

Compilation 2

```
hello  
hello
```

```
1 | x ← return {print "hello"}  
2 | _ ← x!  
3 | x!
```

Expected **effect**?

```
hello  
hello
```

Dead code elimination

```
1 | let  $x = e_1$  in  
2 |  $e_2$ 
```

Dead code elimination

1 | let e_1 in
2 | e_2



... in Call-By-Push-Value

1 | $x \leftarrow M$
2 | N

... in Call-By-Push-Value

1 | $x \leftarrow M$
2 | N

$\Gamma \vdash M : \mathbf{F} A$

$\Gamma, x : A \vdash N : B$

Effects ... in Call-By-Push-Value

1 | $x \leftarrow M$
2 | N

$\Gamma \vdash M :^{\varepsilon} \mathbf{F} A$

Effect: is the code bound to x (i.e. M) pure?

$\Gamma, x : A \vdash N :^{\phi} B$

Effects ... in Call-By-Push-Value

1 | $x \leftarrow_{\varepsilon} M$
2 | N

$\Gamma \vdash M :_{\varepsilon} \mathbf{F} A$

Effect: is the code bound to x (i.e. M) pure?

$\Gamma, x : A \vdash N :_{\phi} B$

Effects and Coeffects in Call-By-Push-Value

$$\begin{array}{l|l} 1 & x \leftarrow_{\varepsilon} M \\ 2 & N \end{array}$$

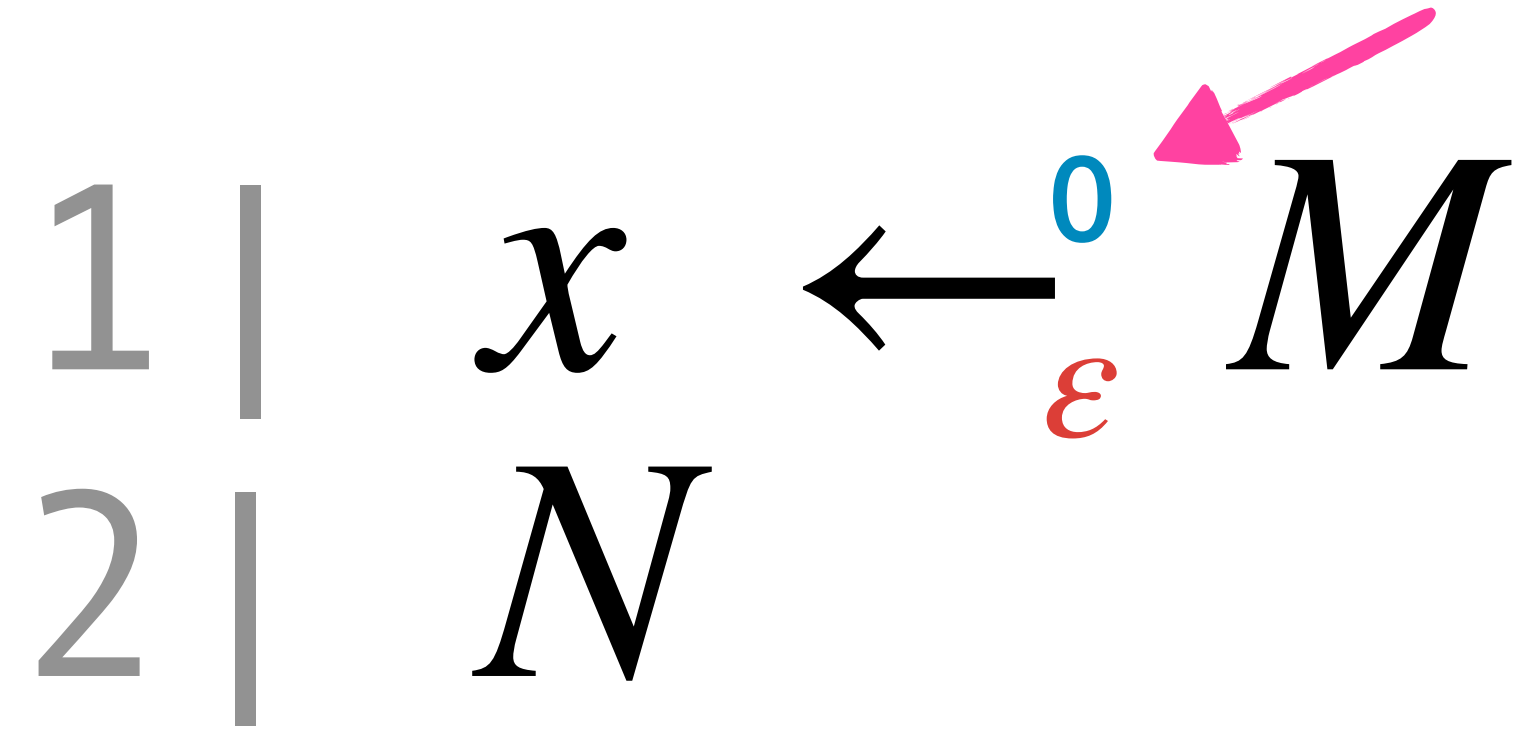
$$\gamma_1 \cdot \Gamma \vdash M :^{\varepsilon} \mathbf{F} A$$

Effect: is the code bound to x (i.e. M) pure?

$$\gamma_2 \cdot \Gamma, x :^0 A \vdash N :^{\phi} B$$

Coeffect: is x unused in N ?

Effects and Coeffects in Call-By-Push-Value



$$\gamma_1 \cdot \Gamma \vdash M :^{\varepsilon} \mathbf{F} A$$

Effect: is the code bound to x (i.e. M) pure?

$$\gamma_2 \cdot \Gamma, x :^0 A \vdash N :^{\phi} B$$

Coeffect: is x unused in N ?

Effects and Coeffects in Call-By-Push-Value

$$\begin{array}{l} 1 \mid x \xleftarrow[\varepsilon]{0} M \\ 2 \mid N \end{array}$$

$$\gamma_1 \cdot \Gamma \vdash M : \varepsilon \mathbf{F} A$$

$$\gamma_2 \cdot \Gamma, x : {}^0 A \vdash N : \phi B$$

Effects and Coeffects in Call-By-Push-Value

$$\begin{array}{l} 1 \mid x \xleftarrow[\varepsilon]{0} M \\ 2 \mid N \end{array}$$

$$\gamma_1 \cdot \Gamma \vdash M : \varepsilon \mathbf{F} A$$

$$\gamma_2 \cdot \Gamma, x : {}^0 A \vdash N : \phi B$$

Effects and Coeffects in Call-By-Push-Value

$$\begin{array}{l} 1 \mid x \xleftarrow[\varepsilon]{0} M \\ 2 \mid N \end{array}$$

$$\gamma_1 \cdot \Gamma \vdash M : \varepsilon \mathbf{F} A \quad \gamma_2 \cdot \Gamma, x : {}^0 A \vdash N : \phi B$$

$$\gamma_2 \cdot \Gamma \vdash x \xleftarrow[\varepsilon]{0} M ; N : \phi B$$

Effects and Coeffects in Call-By-Push-Value

$$\begin{array}{l|l} 1 & x \leftarrow_{\varepsilon}^0 M \\ 2 & N \end{array}$$

demand
ignored

$$\gamma_1 \cdot \Gamma \vdash M :^{\varepsilon} F A \qquad \gamma_2 \cdot \Gamma, x :^0 A \vdash N :^{\phi} B$$

$$\gamma_2 \cdot \Gamma \vdash x \leftarrow_{\varepsilon}^0 M ; N :^{\phi} B$$

Effects and Coeffects in Call-By-Push-Value

$$\begin{array}{l|l} 1 & x \xleftarrow[\varepsilon]{0} M \\ 2 & N \end{array}$$

Effects and Coeffects in Call-By-Push-Value

$$\begin{array}{l|l} 1 & x \leftarrow_{\varepsilon}^0 M \\ 2 & N \end{array}$$

Effects and Coeffects in Call-By-Push-Value

$$\begin{array}{c} x \\ N \end{array} \leftarrow_{\varepsilon}^0 M \equiv N$$

Effects and Coeffects in Call-By-Push-Value

$$\begin{array}{c} x \\ N \end{array} \leftarrow_{\varepsilon}^0 M \equiv N$$



mechanized in



Effects and Coeffects in Call-By-Push-Value Semantics

$$\rho \vdash M \Downarrow T$$

Effects and Coeffects in Call-By-Push-Value Semantics

environment

$$\rho \vdash M \Downarrow T$$

Effects and Coeffects in Call-By-Push-Value Semantics

$$\rho \vdash M \Downarrow T$$

environment *terminal*

Effects and Coeffects in Call-By-Push-Value

Effect-tracking semantics

$$\rho \vdash M \Downarrow T \# \phi$$

Effects and Coeffects in Call-By-Push-Value

Effect-tracking semantics

$$\rho \vdash M \Downarrow T \# \phi$$

resulting effect

Effects and Coeffects in Call-By-Push-Value

Effect-tracking and coeffect-tracking semantics

$$\gamma \cdot \rho \vdash M \Downarrow T \# \phi \quad \text{resulting effect}$$

Effects and Coeffects in Call-By-Push-Value

Effect-tracking and coeffect-tracking semantics

consumed
coeffects

$$\gamma \cdot \rho \vdash M \Downarrow T \# \phi$$

resulting
effect

Effects and Coeffects in Call-By-Push-Value

Effect-tracking and coeffect-tracking semantics

$$\gamma \cdot \rho \vdash M \Downarrow T \# \phi$$

Effects and Coeffects in Call-By-Push-Value

$$\Gamma \vdash M : B$$

$$\gamma \cdot \rho \vdash M \Downarrow T \# \phi$$

Effects and Coeffects in Call-By-Push-Value

$$\Gamma \vdash M : \phi' B$$

$$\gamma \cdot \rho \vdash M \Downarrow T \# \phi$$

Effects and Coeffects in Call-By-Push-Value

Type-and-effect soundness

$$\Gamma \vdash M :^{\phi'} B \qquad \phi \leq \phi'$$

$$\gamma \cdot \rho \vdash M \Downarrow T \# \phi$$

mechanized in



Effects and Coeffects in Call-By-Push-Value

Type-and-effect soundness

$$\gamma' \cdot \Gamma \vdash M :^{\phi'} B \quad \phi \leq \phi'$$

$$\gamma \cdot \rho \vdash M \Downarrow T \# \phi$$

mechanized in



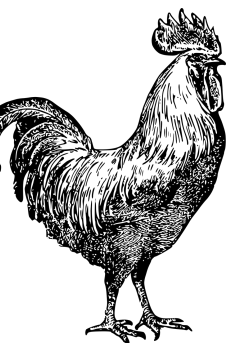
Effects and Coeffects in Call-By-Push-Value

Type-and-effect-and-coeffect soundness

$$\gamma \leq \gamma' \quad \gamma' \cdot \Gamma \vdash M :^{\phi'} B \quad \phi \leq \phi'$$

$$\gamma \cdot \rho \vdash M \Downarrow T \# \phi$$

mechanized in



Effects and Coeffects in Call-By-Push-Value

$$\Gamma \vdash M : B$$

$$\Gamma \vdash \{M\} : \mathbf{U} B$$
$$\Gamma \vdash V : A$$

$$\Gamma \vdash \text{return } V : \mathbf{F} A$$

Effects and Coeffects in Call-By-Push-Value

$$\Gamma \vdash M : B^{\phi}$$

$$\Gamma \vdash \{M\} : \mathbf{U}_{\phi} B$$

$$\Gamma \vdash V : A$$

$$\Gamma \vdash \text{return } V : \mathbf{F} A$$

Effects and Coeffects in Call-By-Push-Value

$$\Gamma \vdash M : B^{\phi}$$

$$\Gamma \vdash \{M\} : \mathbf{U}_{\phi} B$$

suspended
effect



$$\Gamma \vdash V : A$$

$$\Gamma \vdash \text{return } V : \mathbf{F} A$$

Effects and Coeffects in Call-By-Push-Value

$$\gamma \cdot \Gamma \vdash M : B^{\phi}$$

$$\gamma \cdot \Gamma \vdash \{M\} : \mathbf{U}_{\phi} B$$

suspended
effect



$$\gamma \cdot \Gamma \vdash V : A$$

$$(q \cdot \gamma) \cdot \Gamma \vdash \text{return}_q V : \mathbf{F}_q A$$

Effects and Coeffects in Call-By-Push-Value

$$\gamma \cdot \Gamma \vdash M : B^\phi$$

$$\gamma \cdot \Gamma \vdash \{M\} : \mathbf{U}_\phi B$$

suspended
effect

$$\gamma \cdot \Gamma \vdash V : A$$

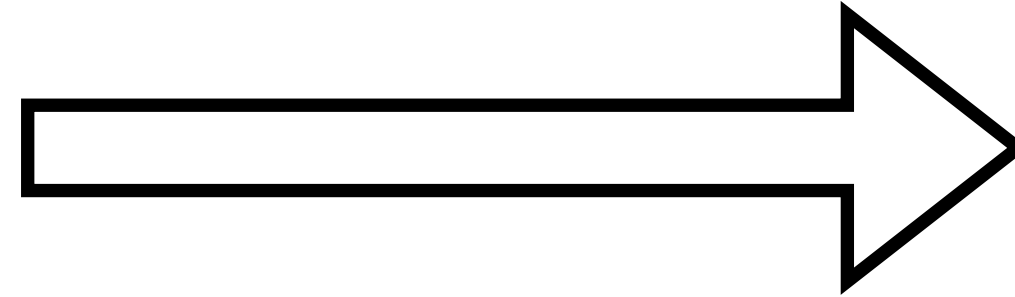
$$(q \cdot \gamma) \cdot \Gamma \vdash \text{return}_q V : \mathbf{F}_q A$$

multiplied
coeffect

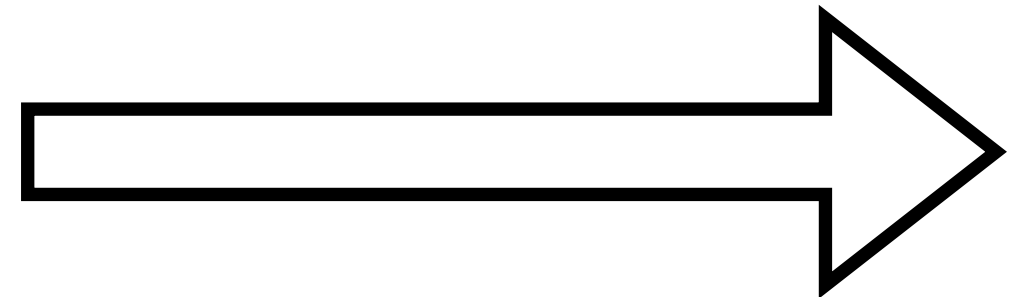
Effects and Coeffects in Call-By-Push-Value

$\Gamma \vdash e : \tau$

CBN Compilation



CBV Compilation

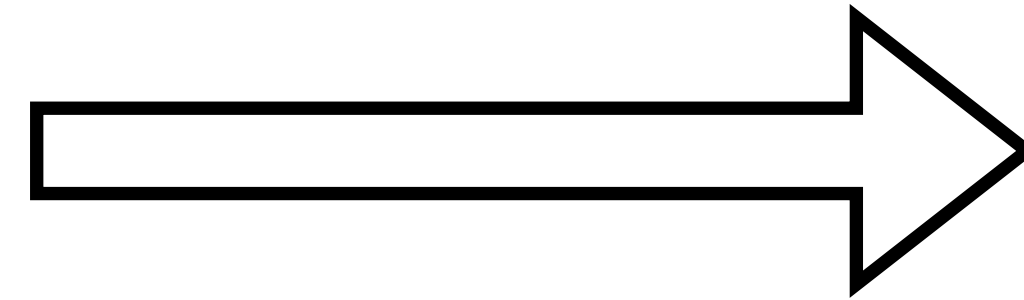


Call-By-Push-Value

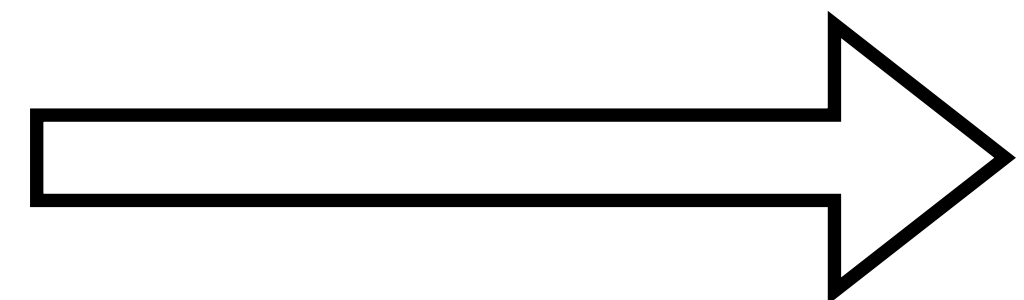
Effects and Coeffects in Call-By-Push-Value

$\Gamma \vdash e : \tau$

CBN Compilation



CBV Compilation



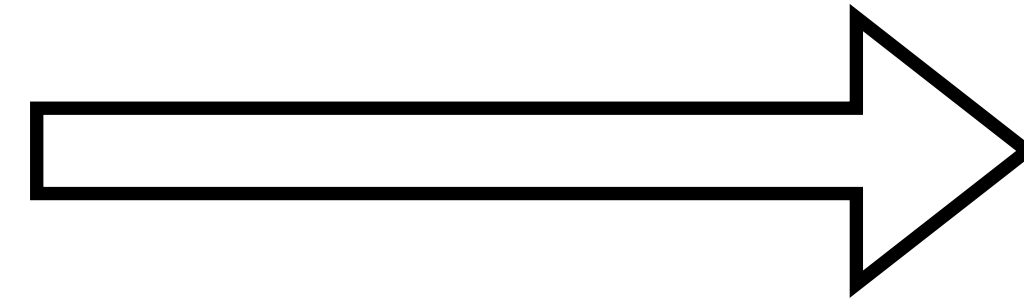
Call-By-Push-Value

preserve
effects

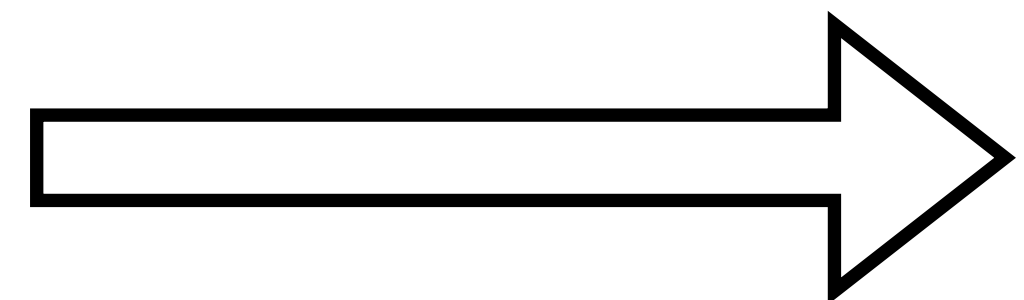
Effects and Coeffects in Call-By-Push-Value

$\gamma \cdot \Gamma \vdash e : \tau$

CBN Compilation



CBV Compilation

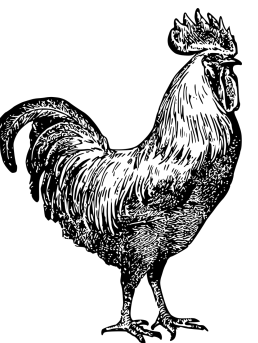


Call-By-Push-Value

preserve
coeffects

preserve
effects

mechanized in



Effects and Coeffects in Call-By-Push-Value

$$\gamma \cdot \Gamma \vdash M :^{\phi} B$$

Thank you!

$$\gamma \cdot \rho \vdash M \Downarrow T \# \phi$$

$$x \leftarrow_{\varepsilon}^0 M ; N \equiv N$$



mechanized in

