Recurrence Extraction for Functional Programs Through Call-by-Push-Value

Alex Kavvos
Department of Computer Science, Aarhus University

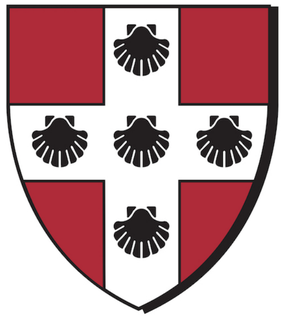
j.w.w. Edward Morehouse, Daniel Licata, and Norman Danner

47th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2020)

arXiv:1911.04588

Recurrence Extraction for Functional Programs Through Call-by-Push-Value

Alex Kavvos
Department of Mathematics and Computer Science,
Wesleyan University



j.w.w. Edward Morehouse, Daniel Licata, and Norman Danner

47th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2020)

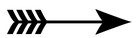
arXiv:1911.04588

Recurrence Extraction for FP

$$
\begin{aligned}
&\text{rec } \mathtt{sort}(xs) \quad = \\
&\text{case } xs \text{ of} \quad \text{nil} \mapsto \text{nil} \\
&\qquad\qquad \mid \text{cons}(y, ys) \mapsto \text{case } ys \text{ of} \quad \text{nil} \mapsto \text{cons}(y, \text{nil}) \\
&\qquad\qquad\qquad\qquad\qquad\qquad \mid \text{cons}(z, zs) \mapsto \text{let } q = \mathtt{divide}(\text{cons}(y, ys)) \text{ in} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathtt{merge}(\mathtt{sort}(\pi_1\, q), \mathtt{sort}(\pi_2\, q))
\end{aligned}
$$

Recurrence Extraction for FP

$$
\begin{array}{l}
\text{rec } \mathtt{sort}(xs) \quad = \\
\text{case } xs \text{ of} \quad \text{nil} \mapsto \text{nil} \\
\qquad\qquad \mid \text{cons}(y, ys) \mapsto \text{case } ys \text{ of} \quad \text{nil} \mapsto \text{cons}(y, \text{nil}) \\
\qquad\qquad\qquad\qquad\qquad\qquad \mid \text{cons}(z, zs) \mapsto \text{let } q = \mathtt{divide}(\text{cons}(y, ys)) \text{ in} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathtt{merge}(\mathtt{sort}(\pi_1\, q), \mathtt{sort}(\pi_2\, q))
\end{array}
$$



Recurrence Extraction for FP

```
rec sort(xs)  =
case xs of   nil ↦ nil
           | cons(y, ys) ↦ case ys of   nil ↦ cons(y, nil)
                                      | cons(z, zs) ↦ let q = divide(cons(y, ys)) in
                                                      merge(sort(π1 q), sort(π2 q))
```

$$T(1) = 0$$

$$T(n) = 7 + (c + d)n + 2T(n/2).$$

Recurrence Extraction for FP

```
rec sort(xs)  =
case xs of   nil ↦ nil
           | cons(y, ys) ↦ case ys of   nil ↦ cons(y, nil)
                                      | cons(z, zs) ↦ let q = divide(cons(y, ys)) in
                                                      merge(sort(π1 q), sort(π2 q))
```

$$T(1) = 0$$

$$T(n) = 7 + (c + d)n + 2T(n/2).$$

✤ Can we do this **automatically**, and establish **formal correctness**?

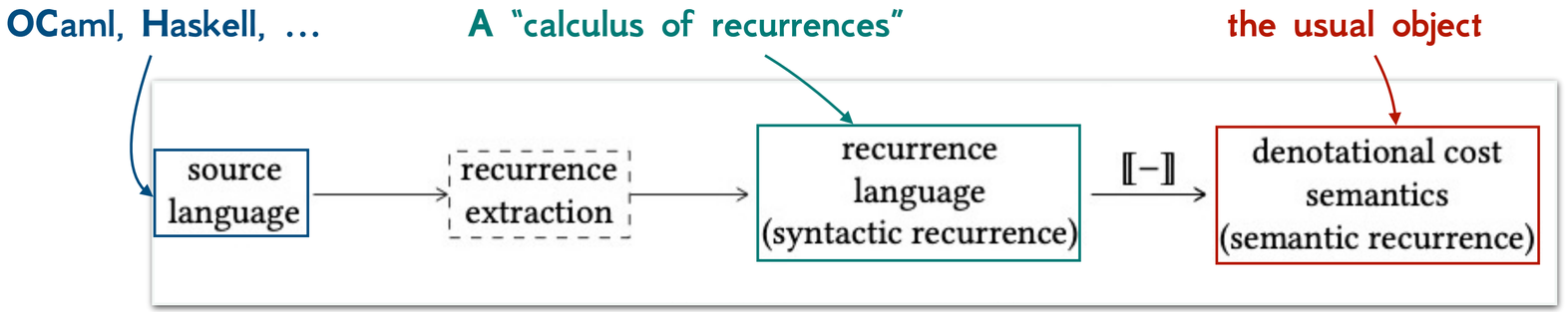
Recurrence Extraction for FP

```
rec sort(xs)  =
case xs of   nil ↦ nil
           | cons(y, ys) ↦ case ys of   nil ↦ cons(y, nil)
                                      | cons(z, zs) ↦ let q = divide(cons(y, ys)) in
                                                      merge(sort(π1 q), sort(π2 q))
```

$$T(1) = 0$$

$$T(n) = 7 + (c + d)n + 2T(n/2).$$

✤ Can we do this **automatically**, and establish **formal correctness**?



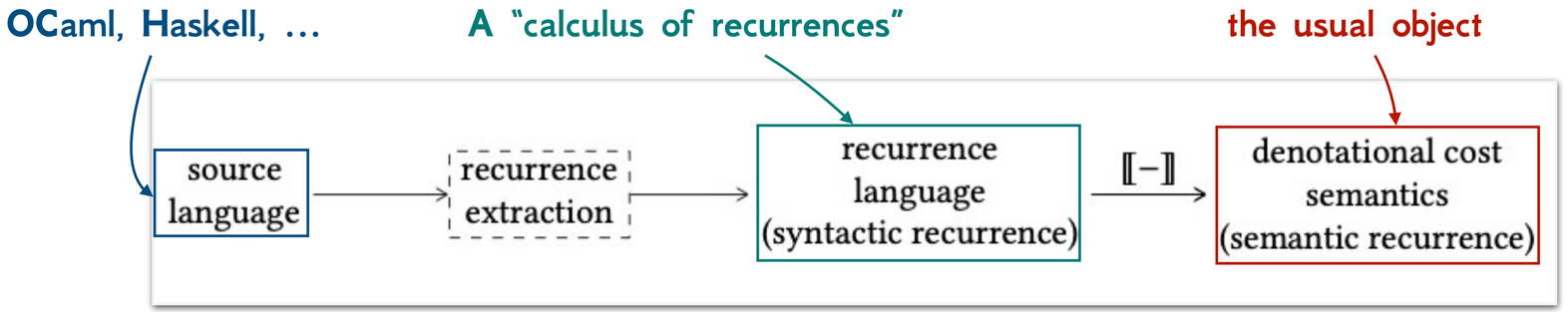
Recurrence Extraction for FP

```
rec sort(xs)  =
case xs of   nil ↦ nil
           | cons(y, ys) ↦ case ys of   nil ↦ cons(y, nil)
                                      | cons(z, zs) ↦ let q = divide(cons(y, ys)) in
                                                      merge(sort(π1 q), sort(π2 q))
```

$$T(1) = 0$$

$$T(n) = 7 + (c + d)n + 2T(n/2).$$

- Can we do this **automatically**, and establish **formal correctness**?
- Solved for **inductive types** in a **total CBV language** [Danner et al. @ ICFP 2015]



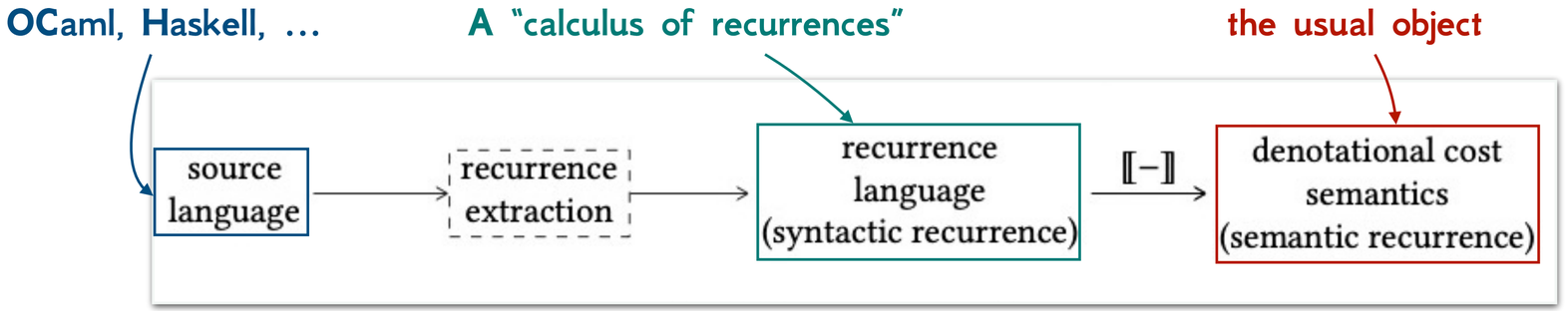
Recurrence Extraction for FP

```
rec sort(xs)   =
case xs of    nil ↦ nil
            | cons(y, ys) ↦ case ys of    nil ↦ cons(y, nil)
                                        | cons(z, zs) ↦ let q = divide(cons(y, ys)) in
                                                        merge(sort(π1 q), sort(π2 q))
```

$$T(1) = 0$$

$$T(n) = 7 + (c + d)n + 2T(n/2).$$

- Can we do this **automatically**, and establish **formal correctness**?
- Solved for **inductive types** in a **total CBV language** [Danner et al. @ ICFP 2015]
- Can we also do it for **recursive** functional programs — CBN and CBV?



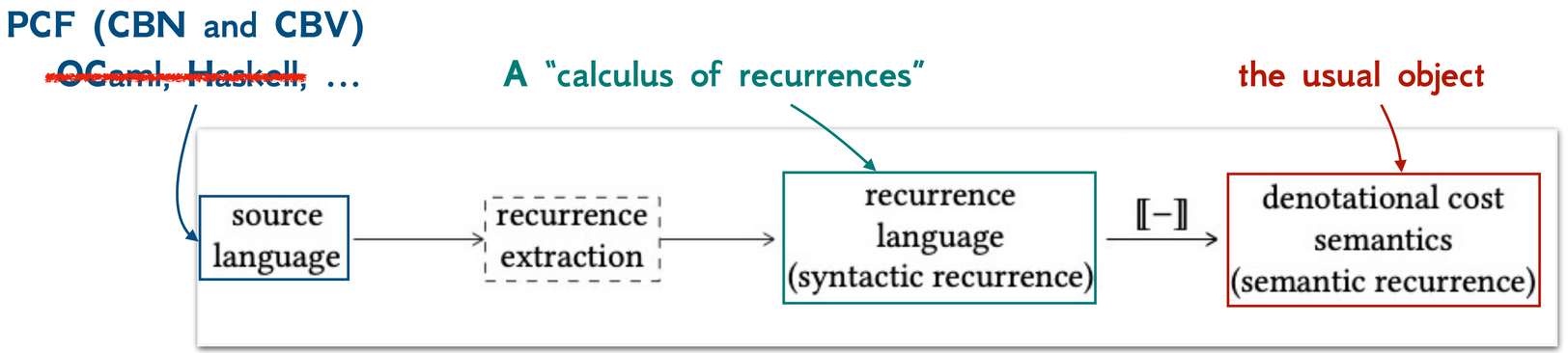
Recurrence Extraction for FP

```
rec sort(xs)  =
case xs of   nil ↦ nil
           | cons(y, ys) ↦ case ys of   nil ↦ cons(y, nil)
                                      | cons(z, zs) ↦ let q = divide(cons(y, ys)) in
                                                      merge(sort(π1 q), sort(π2 q))
```

$$T(1) = 0$$

$$T(n) = 7 + (c + d)n + 2T(n/2).$$

- Can we do this **automatically**, and establish **formal correctness**?
- Solved for **inductive types** in a **total CBV language** [Danner et al. @ ICFP 2015]
- Can we also do it for **recursive** functional programs — CBN and CBV?



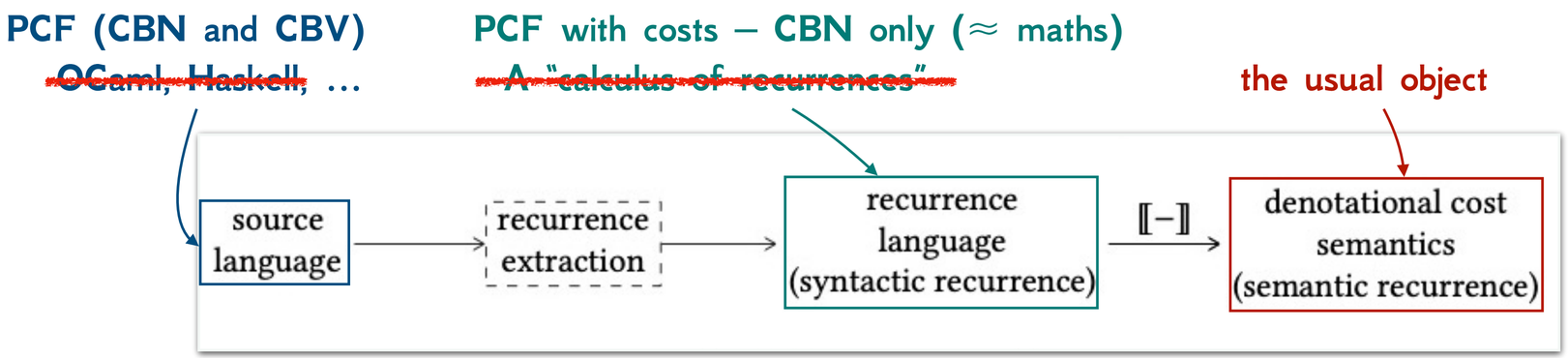
Recurrence Extraction for FP

```
rec sort(xs)   =
case xs of    nil ↦ nil
            | cons(y, ys) ↦ case ys of    nil ↦ cons(y, nil)
                                        | cons(z, zs) ↦ let q = divide(cons(y, ys)) in
                                                        merge(sort(π1 q), sort(π2 q))
```

$$T(1) = 0$$

$$T(n) = 7 + (c + d)n + 2T(n/2).$$

- Can we do this **automatically**, and establish **formal correctness**?
- Solved for **inductive types** in a **total CBV language** [Danner et al. @ ICFP 2015]
- Can we also do it for **recursive** functional programs — CBN and CBV?



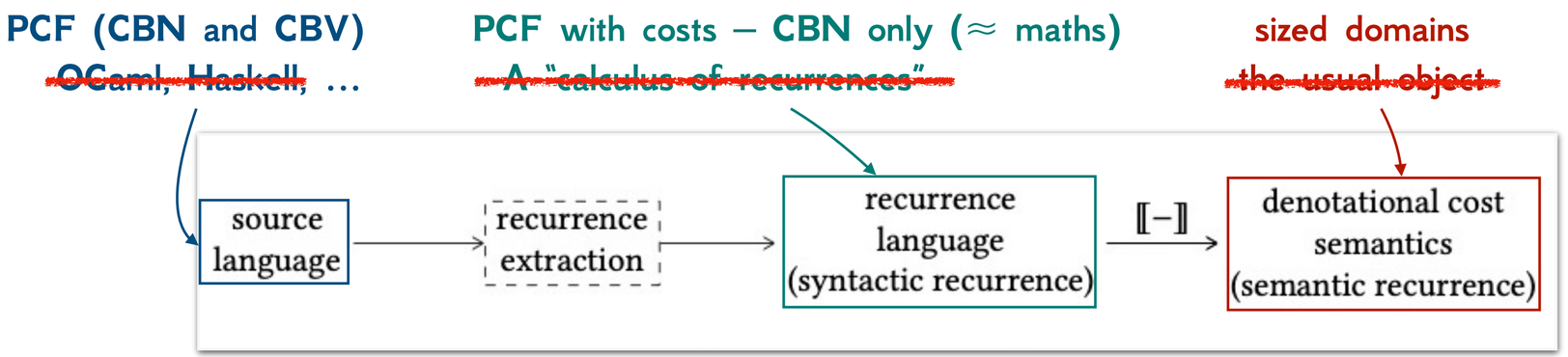
Recurrence Extraction for FP

```
rec sort(xs)   =
case xs of    nil ↦ nil
            | cons(y, ys) ↦ case ys of    nil ↦ cons(y, nil)
                                        | cons(z, zs) ↦ let q = divide(cons(y, ys)) in
                                                        merge(sort(π1 q), sort(π2 q))
```

$$T(1) = 0$$

$$T(n) = 7 + (c + d)n + 2T(n/2).$$

- Can we do this **automatically**, and establish **formal correctness**?
- Solved for **inductive types** in a **total CBV language** [Danner et al. @ ICFP 2015]
- Can we also do it for **recursive** functional programs — CBN and CBV?



PCF

= simply-typed λ-calculus + fixpoints (CBN & CBV)

Constants

$$\frac{n \in \mathbb{N}}{\Gamma \vdash \underline{n} : \mathsf{nat}} \qquad \frac{\Gamma \vdash M, N : \mathsf{nat} \quad \mathbf{op} \in \{+, *, -, \div, \mathrm{mod}\}}{\Gamma \vdash M \ \mathbf{op}\ N : \mathsf{nat}}$$

Function types

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.\ M : A \to B} \qquad \frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash M\, N : B}$$

CBN recursion

$$\frac{\Gamma, x : A \vdash M : A}{\Gamma \vdash \mathsf{fix}\ x.\ M : A}$$

Big-step rules

$$\frac{M \downarrow^{a} \underline{m} \quad N \downarrow^{b} \underline{n}}{M \ \mathbf{op}\ N \downarrow^{a+b} \underline{m \ \mathbf{op}\ n}}$$

$$\frac{M \downarrow^{m} \lambda x.\ P \quad P[N/x] \downarrow^{n} V}{M N \downarrow^{m+n+1} V}$$

$$\frac{M[\mathsf{fix}\ x.\ M/x] \downarrow^{n} V}{\mathsf{fix}\ x.\ M \downarrow^{n+1} V}$$

PCFc

= simply-typed λ-calculus + fixpoints + costs (CBN only)

Constants

$$\frac{n \in \mathbb{N}}{\Gamma \vdash \underline{n} : \mathsf{nat}} \qquad \frac{\Gamma \vdash M, N : \mathsf{nat} \quad \mathbf{op} \in \{+, *, -, \div, \mathrm{mod}\}}{\Gamma \vdash M \ \mathbf{op}\ N : \mathsf{nat}}$$

Function types

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.\ M : A \to B} \qquad \frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash M\, N : B}$$

CBN recursion

$$\frac{\Gamma, x : A \vdash M : A}{\Gamma \vdash \mathsf{fix}\ x.\ M : A}$$

Costs

$$\frac{\widehat{n} \in \{\mathbf{0}, \mathbf{1}\}}{\Gamma \vdash \widehat{n} : \mathbb{C}} \qquad \frac{\Gamma \vdash M : \mathbb{C} \quad \Gamma \vdash N : \mathbb{C}}{\Gamma \vdash M \boxplus N : \mathbb{C}}$$

The Size Preorder

$$\Gamma \vdash M \leqslant N : A$$

"considered as recurrences, M is less than or equal to N at type A"

Extraction for CBV

In CBV:
- **Variables** carry **values**, which are fully evaluated: **only use-cost, no direct cost**
- **All types are observable**, e.g. evaluating a term at A -> B **has direct cost**

1. Translate each PCF type to **two PCFc types:**

Complexity of A
cost of evaluation + potential

Potential of A
size of output value, future use-cost

$$||A|| \stackrel{\text{def}}{=} \mathbb{C} \times \langle\!\langle A \rangle\!\rangle$$

$$\langle\!\langle \mathsf{nat} \rangle\!\rangle \stackrel{\text{def}}{=} \mathsf{nat}$$

$$\langle\!\langle A_1 \times A_2 \rangle\!\rangle \stackrel{\text{def}}{=} \langle\!\langle A_1 \rangle\!\rangle \times \langle\!\langle A_2 \rangle\!\rangle$$

$$\langle\!\langle A \to B \rangle\!\rangle \stackrel{\text{def}}{=} \langle\!\langle A \rangle\!\rangle \to ||B||$$

2. Extract a complexity from each term by induction on syntax.

Given **potentials** for the variables

this **recurrence**

gives the **complexity** of M

Theorem: $\Gamma \vdash M : A \Longrightarrow \langle\!\langle \Gamma \rangle\!\rangle \vdash ||M|| : ||A||$

Extraction for CBN

In CBN:

- **Variables** carry **thunks**, to be evaluated later.
- **Only base types are observable**, e.g. evaluating a term at A -> B has no cost but evaluating a nat might **trigger a cascade of thunk evaluations**.

1. Translate each PCF type to a **PCFc cost algebra**, i.e. a **type &** a **cost action** on

$$||\mathsf{nat}|| \stackrel{\text{def}}{=} \mathbb{C} \times \mathsf{nat}$$

$$||A_1 \times A_2|| \stackrel{\text{def}}{=} ||A_1|| \times ||A_2||$$

$$||A \to B|| \stackrel{\text{def}}{=} ||A|| \to ||B||$$

$$\left(||A||, \alpha_A(c, x)\right)$$

$$c : \mathbb{C}, x : ||A|| \vdash \alpha_A(c, x) : ||A||$$

Algebras are defined by induction on types; they **push costs down to observable types.**

2. Extract a complexity from each term by induction on syntax.

Given **complexities** for the inputs — this **recurrence** — gives the **complexity** of M

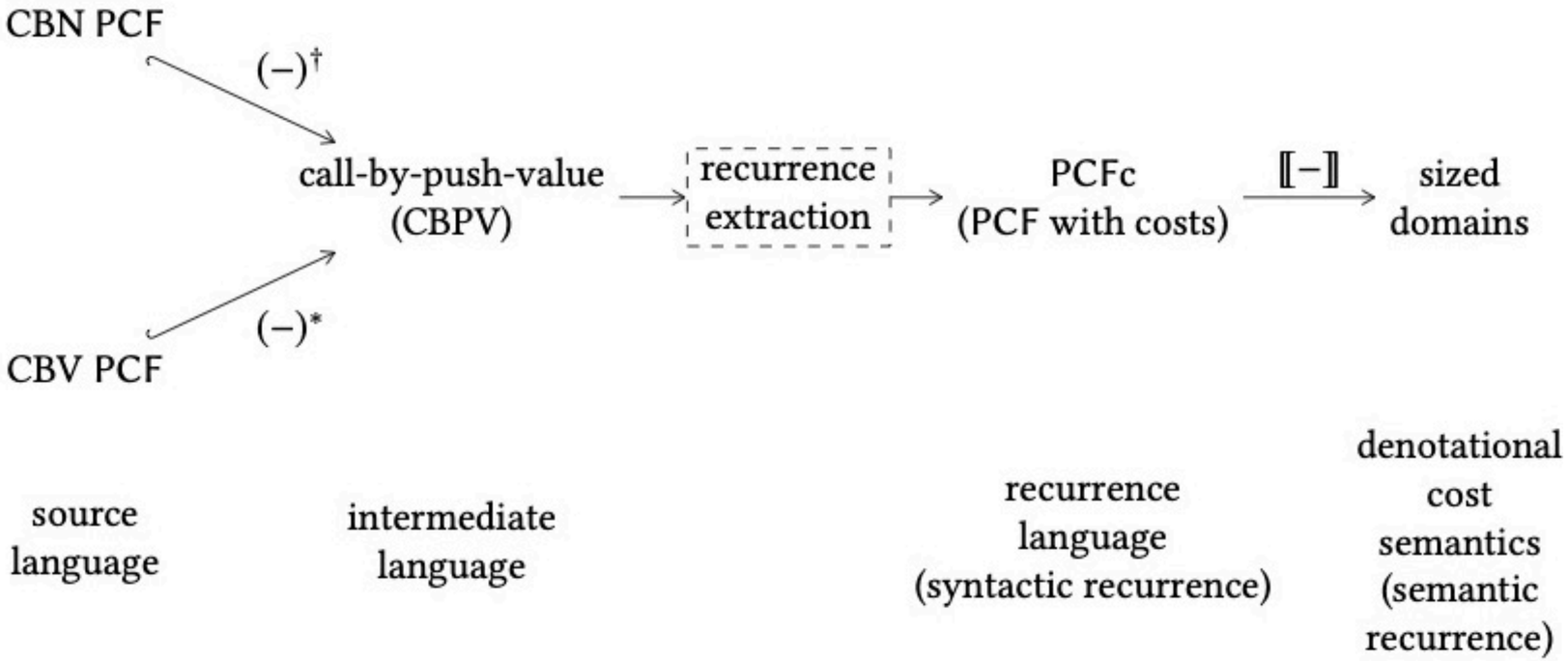
Theorem: $\Gamma \vdash M : A \Longrightarrow ||\Gamma|| \vdash ||M|| : ||A||$

Proving this correct is nontrivial.

- Some difficult issues:
  - Handling **nontermination** is nontrivial.
  - Translating **CBV to CBN.** Continuations?
  - Function types -> need **logical relation** to state correctness. The straightforward definition fails due to fixpoints.
- How did we even come up with the extraction in the first place?

Two for the price of one: Call-by-Push-Value

- CBPV features **modalities** that control the evaluation of terms.
- We can embed both CBV and CBN in it.



Call-by-Push-Value (CBPV)

"A value is, a computation does." — Paul Blain Levy

- Two kinds of types:
  - Terms of **value types** are**...** values.
  - Terms of **computation type** can be **effectful**. Equip with one effect: **charging a unit cost**.
- Extraction: mix'n'match of CBV & CBN styles
  - **Value types -> potentials**
  - **Computation types -> PCFc cost algebras**

thunks

$$A ::= \mathsf{nat} \mid A_1 \times A_2 \mid U\underline{B}$$

$$\underline{B} ::= FA \mid \underline{B}_1 \,\&\, \underline{B}_2 \mid A \to \underline{B}$$

returners

$$\frac{\Gamma \vdash_c M : \underline{B}}{\Gamma \vdash_c \mathsf{charge}.\, M : \underline{B}}$$

$$\frac{M \Downarrow^n T}{\mathsf{charge}.\, M \Downarrow^{n+1} T}$$

Proving correctness

$$\widetilde{n} \lesssim^{\mathsf{val}}_{\mathsf{nat}} E \stackrel{\mathsf{def}}{\equiv} \underline{n} \leqslant E$$

$$(V_1, V_2) \lesssim^{\mathsf{val}}_{A_1 \times A_2} E \stackrel{\mathsf{def}}{\equiv} \begin{cases} V_1 \lesssim^{\mathsf{val}}_{A_1} \pi_1(E) \\ V_2 \lesssim^{\mathsf{val}}_{A_2} \pi_2(E) \end{cases}$$

$$\mathsf{thunk}\ M \lesssim^{\mathsf{val}}_{U\underline{B}} E \stackrel{\mathsf{def}}{\equiv} M \lesssim^{\mathsf{c}}_{\underline{B}} E$$

$$M \lesssim^{\mathsf{c}}_{FA} E \stackrel{\mathsf{def}}{\equiv} E_c{\downarrow} \implies \exists n, V. \begin{cases} M \Downarrow^n \mathsf{return}\ V \\ \widehat{n} \leqslant E_c \\ V \lesssim^{\mathsf{val}}_{A} E_p \end{cases}$$

$$M \lesssim^{\mathsf{c}}_{A \to \underline{B}} E \stackrel{\mathsf{def}}{\equiv} \forall (N \lesssim^{\mathsf{val}}_{A} X).\ M\,N \lesssim^{\mathsf{c}}_{\underline{B}} E\,X$$

$$M \lesssim^{\mathsf{c}}_{\underline{B}_1 \& \underline{B}_2} E \stackrel{\mathsf{def}}{\equiv} \begin{cases} \pi_1(M) \lesssim^{\mathsf{c}}_{\underline{B}_1} \pi_1(E) \\ \pi_2(M) \lesssim^{\mathsf{c}}_{\underline{B}_2} \pi_2(E) \end{cases}$$

Theorem (Bounding Theorem):

$$\cdot \vdash V : A \implies V \lesssim^{\mathsf{val}}_{A} \langle\!\langle V \rangle\!\rangle$$

$$\cdot \vdash M : \underline{B} \implies M \lesssim^{\mathsf{c}}_{\underline{B}} \|M\|$$

Proving correctness

$$\widetilde{n} \lesssim^{\text{val}}_{\text{nat}} E \stackrel{\text{def}}{\equiv} \underline{n} \leqslant E$$

$$(V_1, V_2) \lesssim^{\text{val}}_{A_1 \times A_2} E \stackrel{\text{def}}{\equiv} \begin{cases} V_1 \lesssim^{\text{val}}_{A_1} \pi_1(E) \\ V_2 \lesssim^{\text{val}}_{A_2} \pi_2(E) \end{cases}$$

$$\text{thunk } M \lesssim^{\text{val}}_{U\underline{B}} E \stackrel{\text{def}}{\equiv} M \lesssim^{\text{c}}_{\underline{B}} E$$

$$M \lesssim^{\text{c}}_{FA} E \stackrel{\text{def}}{\equiv} E_c{\downarrow} \implies \exists n, V. \begin{cases} M \Downarrow^n \text{return } V \\ \widehat{n} \leqslant E_c \\ V \lesssim^{\text{val}}_{A} E_p \end{cases}$$

$$M \lesssim^{\text{c}}_{A \to \underline{B}} E \stackrel{\text{def}}{\equiv} \forall (N \lesssim^{\text{val}}_{A} X).\ M\,N \lesssim^{\text{c}}_{\underline{B}} E\,X$$

$$M \lesssim^{\text{c}}_{\underline{B}_1 \& \underline{B}_2} E \stackrel{\text{def}}{\equiv} \begin{cases} \pi_1(M) \lesssim^{\text{c}}_{\underline{B}_1} \pi_1(E) \\ \pi_2(M) \lesssim^{\text{c}}_{\underline{B}_2} \pi_2(E) \end{cases}$$

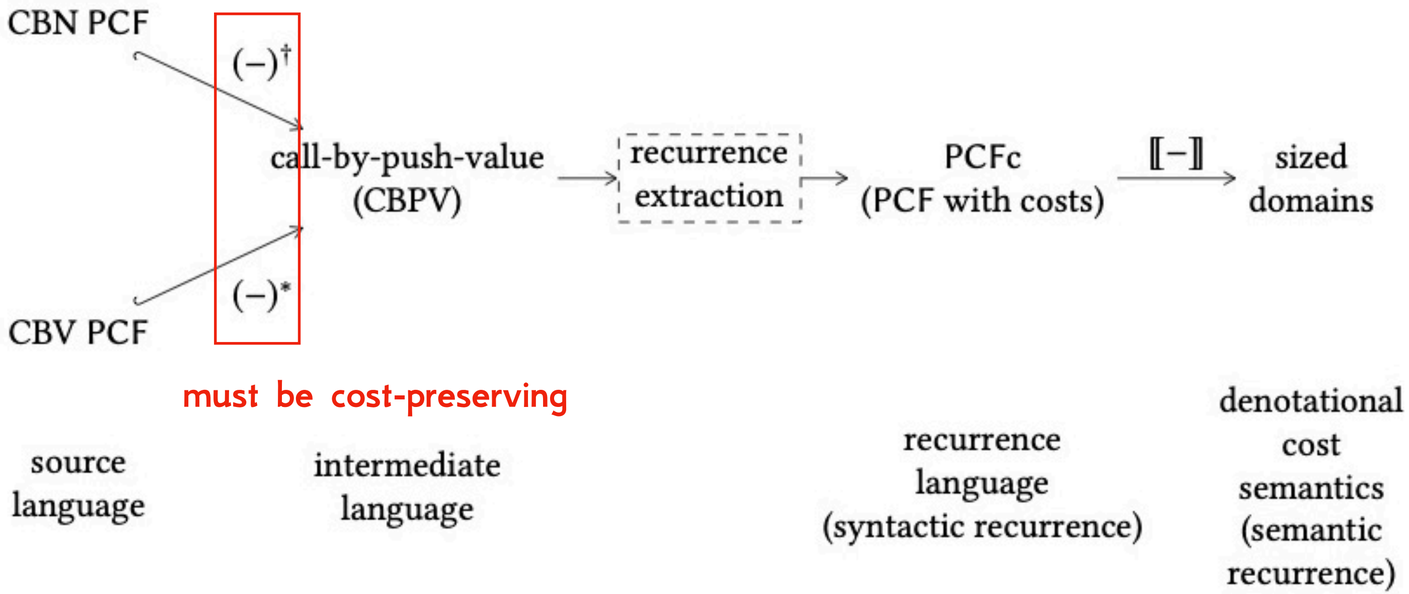
Theorem (Bounding Theorem):

$$\cdot \vdash V : A \implies V \lesssim^{\text{val}}_{A} \langle\!\langle V \rangle\!\rangle$$

$$\cdot \vdash M : \underline{B} \implies M \lesssim^{\text{c}}_{\underline{B}} \|M\|$$

Proving the original theorems for CBN and CBV

- Use the **Levy embedding** of CBN and CBV into CBPV.
- Sprinkle occurrences of "**charge. (-)**" wherever cost should be incurred.
- Prove that the embedding is cost-preserving (a bisimulation-like result).



Denotational semantics of PCFc

- A **sized domain** consists of a set D and
  - An **information order**, i.e. a pointed ω-cpo $(D, \sqsubseteq, \bot)$
  - A **size preorder** $(D, \leqslant, \mathbf{0}, \vee)$

such that

least element

chosen least upper bounds

> **Why preorder?** E.g. the lists [1, 2, 3] and [4, 5, 6] are "size-equal" but not identical.

  - The chosen lubs are continuous w.r.t. the information order.
  - **A better-defined bound is a smaller bound**: $x \sqsubseteq y \Longrightarrow y \leqslant x$
  - **A recursively defined upper bound is an upper bound**:
    for a chain $(x_i)_{i \in \omega}$ we have $(\forall i.\ z \leqslant x_i) \Longrightarrow z \leqslant \sqcup_{i \in \omega} x_i$

Conclusions

- Recurrence extraction for **both CBN and CBV with recursion**: obtained in a uniform way, and **shown to be correct**.
- The extracted recurrences are the expected ones.
- CBPV illuminates the basic concepts of **cost**, **potential** and **complexity:**
  - **cost** is an effect
  - **values** have **potential** (use-cost)
  - **computations** have **complexity** (direct, or deferred cost)
- This is the beginning of a theory of **higher-order recurrences**.
- The analysis can easily be extended to **CBV inductive types**.
- **Recursive types** are harder—but we'll get there!