

# Fully Abstract Normal Form Bisimulation for Call-by-Value PCF

Vasileios Koutavas

Trinity College Dublin

Yu-Yang Lin

Trinity College Dublin

Nikos Tzevelekos

Queen Mary  
Uni. of London

LICS – June 2023

# (Contextual) Program Equivalence

$$P_1 \equiv P_2 : \forall C. C[P_1] \Downarrow \iff C[P_2] \Downarrow$$

i.e. no **context** C can **distinguish**  $P_1$  from  $P_2$

$P_0$  : **fun** f -> 0

$P_1$  : **fun** f -> **if** f() == f() **then** 0 **else** 42

$P_2$  : **fun** f -> **if** (f(); **true**) **then** 0 **else** 42

# (Contextual) Program Equivalence

$$P_1 \equiv P_2 : \forall C. C[P_1] \Downarrow \iff C[P_2] \Downarrow$$

i.e. no **context** C can **distinguish**  $P_1$  from  $P_2$

$P_0$  : **fun** f -> 0

$P_1$  : **fun** f -> **if** f() == f() **then** 0 **else** 42

$P_2$  : **fun** f -> **if** (f(); **true**) **then** 0 **else** 42

$C_{0,1}$ : **let rec** bot \_ = bot() **in** ([-] bot)

$C_{1,2}$ : **let** x = ref 0 **in**  
[-] (**fun** \_ -> **if** !x == 0 **then** x++; 1 **else** bot())

# (Contextual) Program Equivalence

$$P_1 \equiv P_2 : \forall C. C[P_1] \Downarrow \iff C[P_2] \Downarrow$$

i.e. no **context** C can **distinguish**  $P_1$  from  $P_2$

without effects?

$P_0$  : **fun** f -> 0

$P_0 \not\equiv P_1 \equiv P_2$

$P_1$  : **fun** f -> **if** f() == f() **then** 0 **else** 42

$P_2$  : **fun** f -> **if** (f(); **true**) **then** 0 **else** 42

$C_{0,1}$ : **let rec** bot \_ = bot() **in** ([-] bot)

~~$C_{1,2}$ : **let** x = ref 0 **in**~~

~~[-] (**fun** \_ -> **if** !x == 0 **then** x++; 1 **else** bot())~~

# Contribution

A fully abstract normal form bisimulation for PCFv

Turing complete  
call-by-value pure  
functional language

i.e. we devise a fully abstract model comprising:

- a game-semantics Labelled Transition System for open terms  
e.g:

$$\frac{(D, \vec{v}) = \text{ulpatt}(v) \quad K \neq \cdot \quad t' = t + \text{ret}(D)}{\langle A; M; K; t; v; \vec{u}, V \rangle \xrightarrow{\tau} \langle A; M[t']; K; t'; V; \vec{u}, \vec{v} \rangle}$$

- a Normal Form Bisimulation (over the LTS) so that

$$P_1 \equiv P_2 \iff \llbracket P_1 \rrbracket \approx \llbracket P_2 \rrbracket$$

+ PCFseq: bounded equivalence checker tool



# PCF(v) Full Abstraction / Normal Form Bisimulation

TCS: 1977

LCF CONSIDERED AS A PROGRAMMING LANGUAGE

G.D. PLOTKIN

*Department of Artificial Intelligence, University of Edinburgh, Hope Park Square, Meadow Lane,  
Edinburgh EH8 9NW, Scotland*

- full abstraction problem:

$$P_1 \equiv P_2 \iff \llbracket P_1 \rrbracket = \llbracket P_2 \rrbracket$$

- (continuous) functions too extensional

$$\llbracket P_1 \rrbracket = \llbracket P_2 \rrbracket \implies P_1 \equiv P_2$$

$$P_1 \equiv P_2 \not\implies \llbracket P_1 \rrbracket = \llbracket P_2 \rrbracket$$

# PCF(v) Full Abstraction / Normal Form Bisimulation

TCS: 1977

## LCF CONSIDERED AS A PROGRAMMING LANGUAGE

G.D. PLOTKIN

*Department of Artificial Intelligence, University of Edinburgh, Hope Park Square, Meadow Lane,  
Edinburgh EH8 9NW, Scotland*

- full abstraction problem:

$$P_1 \equiv P_2 \iff \llbracket P_1 \rrbracket = \llbracket P_2 \rrbracket$$

- (continuous) functions too extensional

stable functions, logical relations, ...

$$\llbracket P_1 \rrbracket = \llbracket P_2 \rrbracket \implies P_1 \equiv P_2$$

$$P_1 \equiv P_2 \not\implies \llbracket P_1 \rrbracket = \llbracket P_2 \rrbracket$$

- game semantics: functions  $\rightarrow$  strategies

1. Abramsky, Jagadeesan & Malacaria;  
2. Hyland & Ong; 3. Nickau. All: ~1994

$$\forall \rho : \llbracket A \rrbracket. \exists P. \rho = \llbracket P \rrbracket$$

$$P_1 \equiv P_2 \iff \llbracket P_1 \rrbracket \equiv \llbracket P_2 \rrbracket$$

- direct model: strategies  $\rightarrow$  sets of O-views

Churchill, Laird & McCusker: 2010

# PCF(v) Full Abstraction / Normal Form Bisimulation

## LCF CONSIDERED AS A PROGRAMMING LANGUAGE

TCS: 1977

G.D. PLOTKIN

Department of Artificial Intelligence, University of Edinburgh, Hope Park Square, Meadow Lane, Edinburgh EH8 9NW, Scotland

- full abstraction problem:

$$P_1 \equiv P_2 \iff \llbracket P_1 \rrbracket = \llbracket P_2 \rrbracket$$

- (continuous) functions too extensional

$$\llbracket P_1 \rrbracket = \llbracket P_2 \rrbracket \implies P_1 \equiv P_2$$

$$P_1 \equiv P_2 \not\implies \llbracket P_1 \rrbracket = \llbracket P_2 \rrbracket$$

- game semantics: functions  $\rightarrow$  strategies

1. Abramsky, Jagadeesan & Malacaria;  
2. Hyland & Ong; 3. Nickau. All: ~1994

$$\forall \rho : \llbracket A \rrbracket. \exists P. \rho = \llbracket P \rrbracket$$

$$P_1 \equiv P_2 \iff \llbracket P_1 \rrbracket \equiv \llbracket P_2 \rrbracket$$

- direct model: strategies  $\rightarrow$  sets of O-views

Churchill, Laird & McCusker: 2010

LICS 1992

## The Lazy Lambda Calculus in a Concurrency Scenario (Extended Abstract)

Davide Sangiorgi  
LFCS - Department of Computer Science  
Edinburgh University

- devise LTS for open terms
- input functions abstracted away to *names*
- equivalence = bisimilarity

$$P_1 \equiv P_2 \iff \llbracket P_1 \rrbracket \approx \llbracket P_2 \rrbracket$$

- extended to languages with plethora of effects and data structures
- connections to (operational) game semantics

Lassen & Levy: LICS 2010

- LTS too discriminating for PCF

stable  
functions,  
logical  
relations, ...



# Call-by-Value PCF (PCFv)

Types  $T ::= \text{bool} \mid \text{int} \mid \text{unit} \mid T \rightarrow T \mid T \times T$

Exp's  $e ::= v \mid x \mid (e, e) \mid \pi_i(e) \mid \text{op}(\vec{e}) \mid e e \mid \text{if } e \text{ then } e \text{ else } e$

Values  $v ::= c \mid (v, v) \mid \text{fix}_T(x).e \mid \alpha$  ←

ECxt's  $E ::= [\cdot]_T \mid (E, e) \mid (v, E) \mid \pi_i(E) \mid \text{op}(\vec{v}, E, \vec{e})$   
 $\mid E e \mid v E \mid \text{if } E \text{ then } e \text{ else } e$

Note: a term may contain functions provided by its **context**

- AKA unknown / external / abstract functions

Abstract functions  
(*names*  $\alpha, \beta$ , etc.)

# Building LTS for open terms

## Game Semantics setup

Computation is a **game** between:

- the term – Proponent
- and its context – Opponent

A term (i.e. Proponent):

- can reduce ( $e \rightarrow e'$ )
- or hit a name application ( $\alpha v$ ) and make a **call** move
- or produce a value ( $v$ ) and make a **return** move

A context (i.e. Opponent):

- can make a **call** move
- or make a **return** move

Note: **abstract** functions ( $\alpha, \beta$ , etc.)

# Building LTS for open terms

## Game Semantics setup

Computation is a **game** between:

- the term – *Proponent*
- and its context – *Opponent*

A term (i.e. Proponent):

- can reduce ( $e \rightarrow e'$ )
- or hit a name application ( $\alpha v$ ) and make a **call** move
- or produce a value ( $v$ ) and make a **return** move

A context (i.e. Opponent):

- can make a **call** move
- or make a **return** move

Note: **abstract** functions ( $\alpha, \beta$ , etc.)

$$\langle e; \dots \rangle_P \xrightarrow{\tau} \langle e'; \dots \rangle_P$$

# Building LTS for open terms

## Game Semantics setup

Computation is a **game** between:

- the term – *Proponent*
- and its context – *Opponent*

A term (i.e. Proponent):

- can reduce ( $e \rightarrow e'$ )
- or hit a name application ( $\alpha v$ ) and make a **call** move
- or produce a value ( $v$ ) and make a **return** move

A context (i.e. Opponent):

- can make a **call** move
- or make a **return** move

Note: **abstract** functions ( $\alpha, \beta$ , etc.)

$$\langle e; \dots \rangle_P \xrightarrow{\tau} \langle e'; \dots \rangle_P$$

$$\langle E[\alpha v]; K; \dots \rangle_P \xrightarrow{\text{call}(a,v)} \langle \vec{v}; E::K; \dots \rangle_O$$

$$\langle v; K; \dots \rangle_P \xrightarrow{\text{ret}(a,v)} \langle \vec{v}; K; \dots \rangle_O$$

$\vec{v}$  : all the Proponent functions Opponent can call  
 $K$  : evaluation (continuation) stack

# Building LTS for open terms

## Game Semantics setup

Computation is a **game** between:

- the term – *Proponent*
- and its context – *Opponent*

A term (i.e. Proponent):

- can reduce ( $e \rightarrow e'$ )
- or hit a name application ( $\alpha v$ ) and make a **call** move
- or produce a value ( $v$ ) and make a **return** move

A context (i.e. Opponent):

- can make a **call** move
- or make a **return** move

Note: **abstract** functions ( $\alpha, \beta$ , etc.)

$$\langle e; \dots \rangle_P \xrightarrow{\tau} \langle e'; \dots \rangle_P$$

$$\langle E[\alpha v]; K; \dots \rangle_P \xrightarrow{\text{call}(a,v)} \langle \vec{v}; E::K; \dots \rangle_O$$

$$\langle v; K; \dots \rangle_P \xrightarrow{\text{ret}(a,v)} \langle \vec{v}; K; \dots \rangle_O$$

$\vec{v}$  : all the Proponent functions Opponent can call  
 $K$  : evaluation (continuation) stack

$$\langle \vec{v}; K; M; t; \dots \rangle_O \xrightarrow{\text{call}(i,v')} \langle v_i v'; K; M[t']; t'; \dots \rangle_P$$

$$\langle \vec{v}; (t', E)::K; M; t; \dots \rangle_O \xrightarrow{\text{ret}(v')} \langle E[v']; K; M[t'']; t'; \dots \rangle_P$$

$t$  : the current *Opponent view* (a part of the play)  
 $M$  : memory used for Opponent *innocence*  
 $t' = t + \text{call}(i,v)$ ,  $t'' = t + \text{ret}(v')$

# Normal Form Bisimulation

**Definition 15** (Weak (Bi-)Simulation). A binary relation  $\mathcal{R}$  on initial and final configurations is a *weak simulation* when for all  $C_1 \mathcal{R} C_2$ :

- *Initial configurations*: if  $C_1 \xrightarrow{\text{ret}(D')} \gg C'_1$ , there exists  $C'_2$  such that  $C_2 \xrightarrow{\text{ret}(D')} \gg C'_2$  and  $C'_1 \mathcal{R} C'_2$ ;
- *Final configurations*: if  $C_1 \xrightarrow{\text{call}(i, D[\vec{\alpha}]) \text{ret}(D')} \gg C'_1$  with  $\vec{\alpha}$  fresh for  $C_2$ , there exists  $C'_2$  such that  $C_2 \xrightarrow{\text{call}(i, D[\vec{\alpha}]) \text{ret}(D')} \gg C'_2$  and  $C'_1 \mathcal{R} C'_2$ ;
- $M_{C_1} \subseteq M_{C_2}$  (where  $M_{C_i}$  is the  $M$ -component of  $C_i$ ).

If  $\mathcal{R}, \mathcal{R}^{-1}$  are weak simulations then  $\mathcal{R}$  is a *weak bisimulation*. Similarity ( $\sqsubseteq$ ) and bisimilarity ( $\approx$ ) are the largest weak simulation and bisimulation, respectively.  $\square$

We focus on *top-level* moves: opponent calls and returns at the outermost context

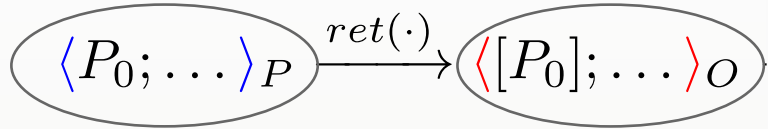
Inner moves are hidden!

They are not observable, unless they lead to different opponent behaviours, which are recorded in  $M$

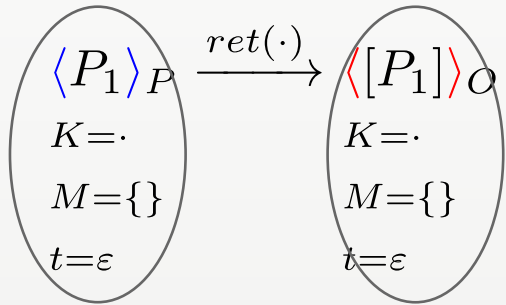
**Theorem 19** (Full abstraction).  $e_1 \approx e_2$  iff  $e_1 \equiv e_2$ .

# Examples

$P_0$  : **fun**  $f$   $\rightarrow$  0

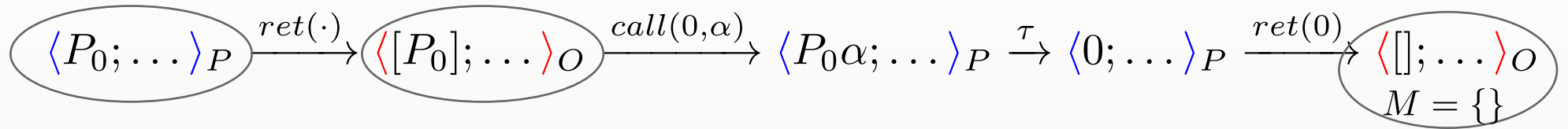


$P_1$  : **fun**  $f$   $\rightarrow$  **if**  $f()$  **==**  $f()$  **then** 0 **else** 42

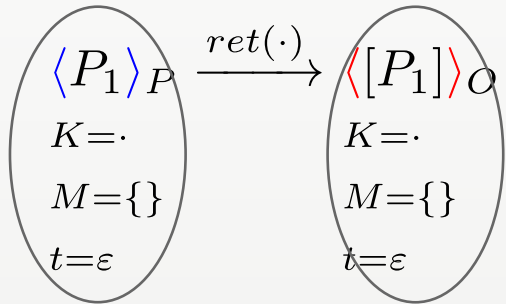


# Examples

$P_0$  : **fun**  $f$   $\rightarrow$  0

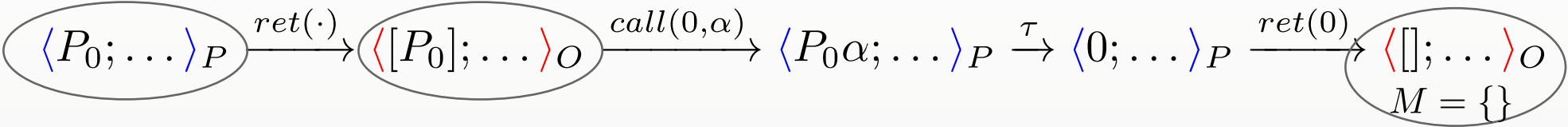


$P_1$  : **fun**  $f$   $\rightarrow$  **if**  $f()$   $==$   $f()$  **then** 0 **else** 42



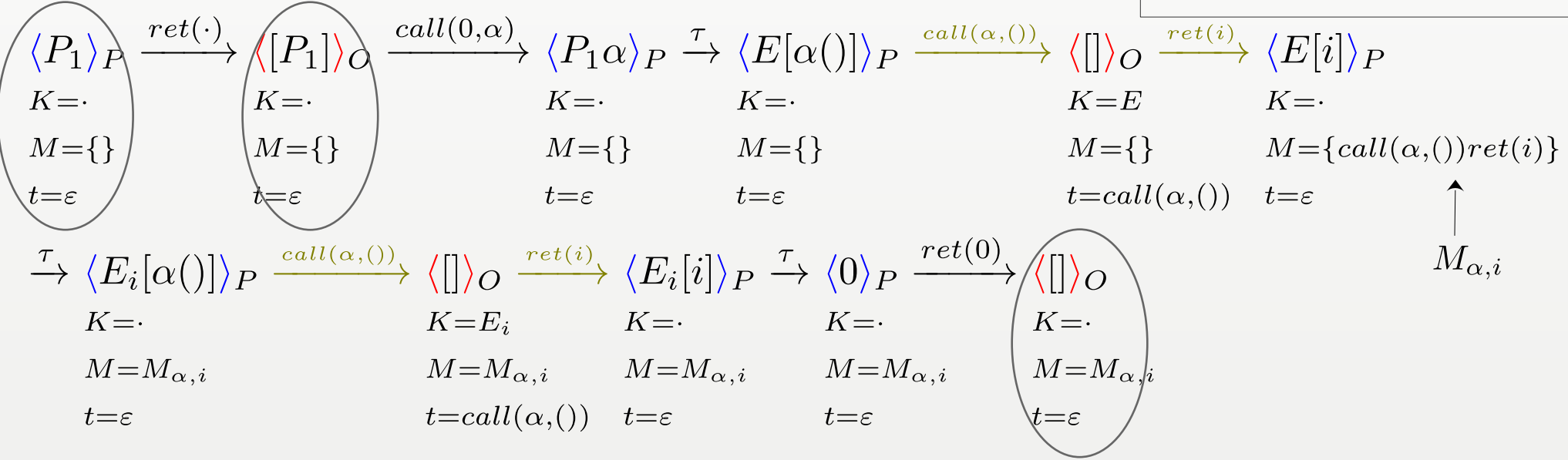
# Examples

$P_0 : \mathbf{fun} \ f \ -> \ 0$



$P_1 : \mathbf{fun} \ f \ -> \ \mathbf{if} \ f() == f() \ \mathbf{then} \ 0 \ \mathbf{else} \ 42$

$E : \mathbf{if} \ [.] == \alpha() \ \mathbf{then} \ 0 \ \mathbf{else} \ 42$   
 $E_i : \mathbf{if} \ i == [.] \ \mathbf{then} \ 0 \ \mathbf{else} \ 42$

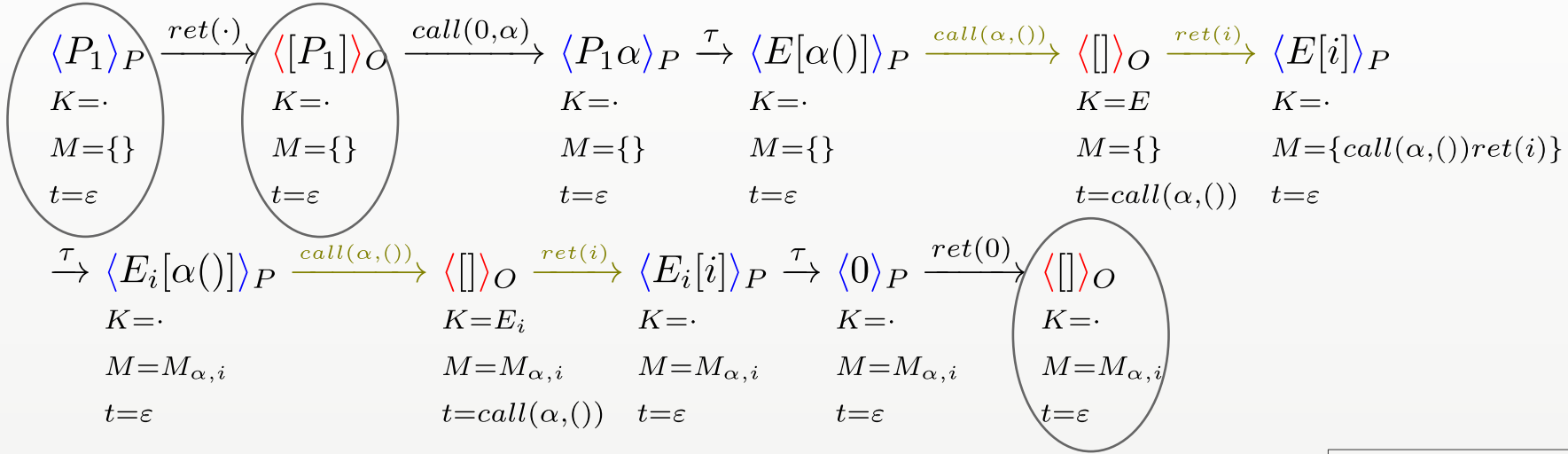


# Examples

$P_1$  : **fun**  $f$   $\rightarrow$  **if**  $f()$  **==**  $f()$  **then** 0 **else** 42

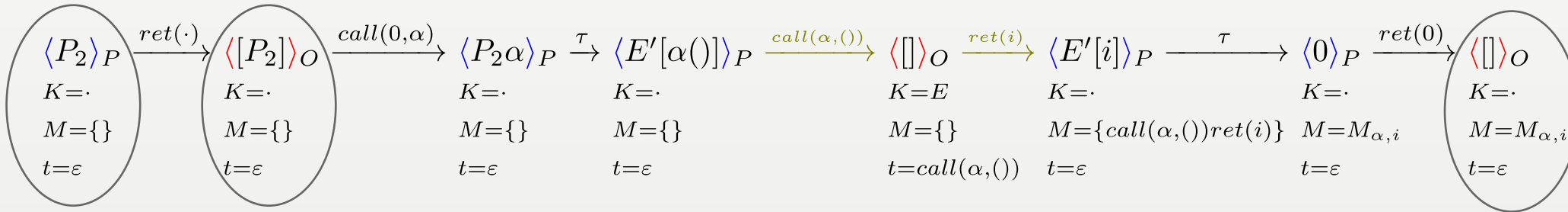
$E$  : **if**  $[.]$  **==**  $\alpha()$  **then** 0 **else** 42

$E_i$  : **if**  $i$  **==**  $[.]$  **then** 0 **else** 42



$P_2$  : **fun**  $f$   $\rightarrow$  **if** ( $f()$ ; **true**) **then** 0 **else** 42

$E'$  : **if**  $[-]$ ; **true** **then** 0 **else** 42



# Summary

A fully abstract normal form bisimulation for PCFv

i.e. we devise a fully abstract model comprising:

- a Labelled Transition System for open terms akin to operational game semantics, e.g:

$$\frac{(D, \vec{v}) = \text{ulpatt}(v) \quad K \neq \cdot \quad t' = t + \text{ret}(D)}{\langle A; M; K; t; v; \vec{u}, V \rangle \xrightarrow{\tau} \langle A; M[t']; K; t'; V; \vec{u}, \vec{v} \rangle}$$

- a Normal Form Bisimulation (over the LTS) so that

$$P_1 \equiv P_2 \iff \llbracket P_1 \rrbracket \approx \llbracket P_2 \rrbracket$$

+ implementation in a bounded equivalence checker



Further on:

- equivalences still hard in practice!
- Up-to techniques and abstractions, cf. K.L.T. TACAS 2022