

Notes on Formal Compiler Construction with the Π Framework

Christiano Braga

Instituto de Computação,
Universidade Federal Fluminense, Niterói, Brazil

August 4, 2021

<http://github.com/ChristianoBraga/PiFramework>



1. Introduction

Example

2. Π IR expressions

Grammar

Automaton

3. Π commands

Grammar

Automaton

4. Π IR declarations

Grammar

Automaton

5. Π IR abstractions

Grammar

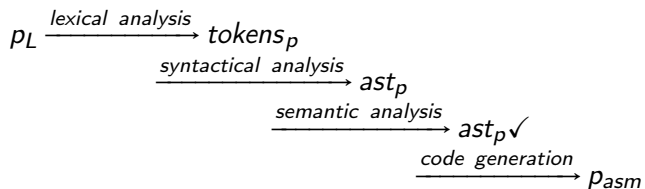
Automaton

6. Π IR recursive abstractions

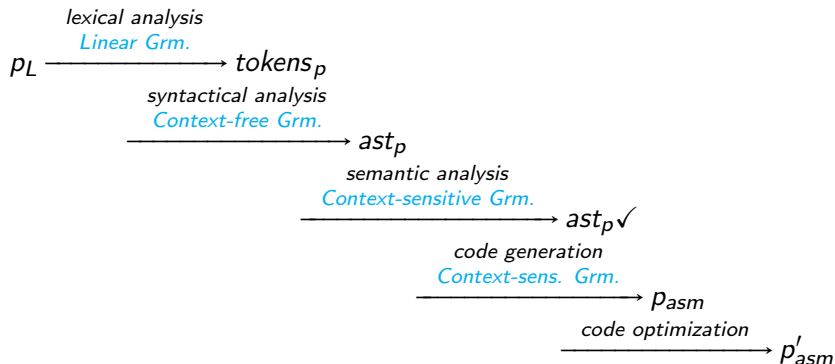
Grammar

Automaton

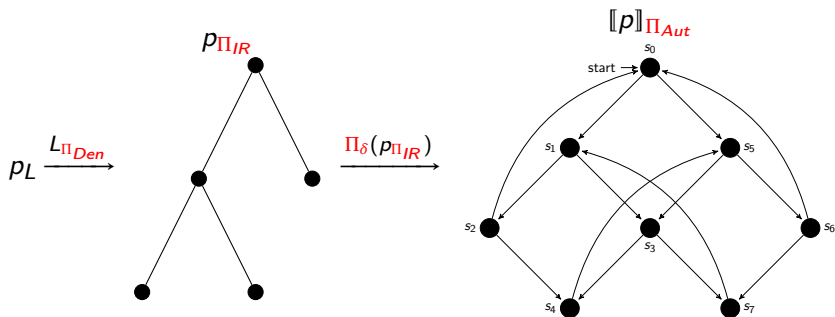
Compiler pipeline



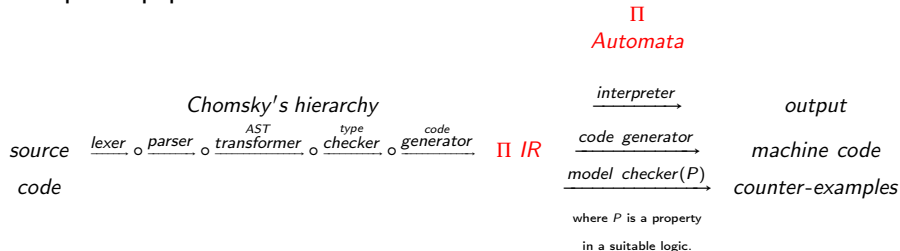
Compiler pipeline and formal languages



Compiler pipeline with the Π Framework I



Compiler pipeline with the Π Framework II



- Π IR defines a set of constructions common to many programming languages.
- Π IR constructions have a formal automata-based semantics in Π automata.
- One may execute (or validate) a program in a given language by running its associated Π IR program.

Compiler pipeline with the Π Framework III

- Π Framework:
<http://github.com/ChristianoBraga/PiFramework>
- Notes on Formal Compiler Construction with the Π Framework:
<https://github.com/ChristianoBraga/PiFramework/blob/master/notes/notes.pdf>.

A calculator

We wish to compute simple arithmetic expressions such as $5 * (3 + 2)$.

A calculator: Lexer

$\langle \textit{digit} \rangle ::= [0..9]$

$\langle \textit{digits} \rangle ::= \langle \textit{digit} \rangle^+$

$\langle \textit{boolean} \rangle ::= \text{'true'} \mid \text{'false'}$

A calculator: concrete syntax

$$\langle \text{exp} \rangle ::= \langle \text{aexp} \rangle \mid \langle \text{bexp} \rangle$$
$$\langle \text{aexp} \rangle ::= \langle \text{aexp} \rangle '+' \langle \text{term} \rangle \mid \langle \text{aexp} \rangle '-' \langle \text{term} \rangle \mid \langle \text{term} \rangle$$
$$\langle \text{term} \rangle ::= \langle \text{term} \rangle '*' \langle \text{factor} \rangle \mid \langle \text{term} \rangle '/' \langle \text{factor} \rangle \mid \langle \text{factor} \rangle$$
$$\langle \text{factor} \rangle ::= '(' \langle \text{aexp} \rangle ')' \mid \langle \text{digits} \rangle$$
$$\begin{aligned} \langle \text{bexp} \rangle ::= & \langle \text{boolean} \rangle \mid \sim \langle \text{bexp} \rangle \mid \langle \text{bexp} \rangle \langle \text{boolop} \rangle \langle \text{bexp} \rangle \\ & \mid \langle \text{aexp} \rangle \langle \text{iop} \rangle \langle \text{aexp} \rangle \end{aligned}$$
$$\langle \text{boolop} \rangle ::= '=' \mid '/\backslash' \mid '\backslash/'$$
$$\langle \text{iop} \rangle ::= '<' \mid '>' \mid '<=' \mid '>='$$

A calculator: abstract syntax

$$\langle exp \rangle ::= \langle digits \rangle \mid \langle boolean \rangle \mid \langle exp \rangle \langle bop \rangle \langle exp \rangle$$
$$\langle bop \rangle ::= '+' \mid '-' \mid '*' \mid '/' \mid '=' \mid '\backslash' \mid '\wedge' \mid '<' \mid '>' \mid '<=' \mid '>='$$

A calculator: Π denotations I

Let D in $\langle \text{digits} \rangle$, B in $\langle \text{boolean} \rangle$ and E_1, E_2 in $\langle \text{exp} \rangle$,

$$\llbracket D \rrbracket_{\Pi} = \text{Num}(D) \quad (1)$$

$$\llbracket B \rrbracket_{\Pi} = \text{Boo}(B) \quad (2)$$

$$\llbracket E_1 + E_2 \rrbracket_{\Pi} = \text{Sum}(\llbracket E_1 \rrbracket_{\Pi}, \llbracket E_2 \rrbracket_{\Pi}) \quad (3)$$

$$\llbracket E_1 - E_2 \rrbracket_{\Pi} = \text{Sub}(\llbracket E_1 \rrbracket_{\Pi}, \llbracket E_2 \rrbracket_{\Pi}) \quad (4)$$

$$\llbracket E_1 * E_2 \rrbracket_{\Pi} = \text{Mul}(\llbracket E_1 \rrbracket_{\Pi}, \llbracket E_2 \rrbracket_{\Pi}) \quad (5)$$

$$\llbracket E_1 / E_2 \rrbracket_{\Pi} = \text{Div}(\llbracket E_1 \rrbracket_{\Pi}, \llbracket E_2 \rrbracket_{\Pi}) \quad (6)$$

$$\llbracket E_1 < E_2 \rrbracket_{\Pi} = \text{Lt}(\llbracket E_1 \rrbracket_{\Pi}, \llbracket E_2 \rrbracket_{\Pi}) \quad (7)$$

$$\llbracket E_1 \leq E_2 \rrbracket_{\Pi} = \text{Le}(\llbracket E_1 \rrbracket_{\Pi}, \llbracket E_2 \rrbracket_{\Pi}) \quad (8)$$

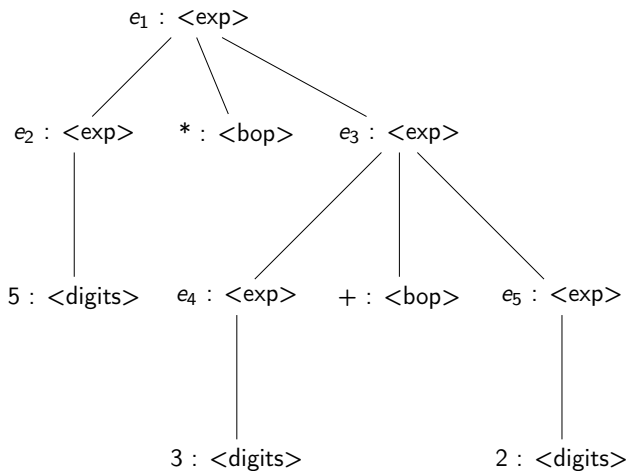
$$\llbracket E_1 > E_2 \rrbracket_{\Pi} = \text{Gt}(\llbracket E_1 \rrbracket_{\Pi}, \llbracket E_2 \rrbracket_{\Pi}) \quad (9)$$

$$\llbracket E_1 \geq E_2 \rrbracket_{\Pi} = \text{Ge}(\llbracket E_1 \rrbracket_{\Pi}, \llbracket E_2 \rrbracket_{\Pi}) \quad (10)$$

A calculator: Π denotations II

- Π denotations are *functions* $\llbracket \cdot \rrbracket_{\Pi} : AST \rightarrow \Pi \text{ IR}$, where *AST* denotes the *datatype* for the abstract syntax tree and $\Pi \text{ IR}$ denotes the datatype for $\Pi \text{ IR}$ programs.
- Note that $\llbracket \cdot \rrbracket_{\Pi}$ has *trees* as parameters, instances of *AST*. The example expression $5 * (3 + 2)$ becomes

A calculator: Π denotations III



A calculator: Π denotations IV

$$\llbracket 5 * (3 + 2) \rrbracket_{\Pi} = \text{Mul}(\llbracket 5 \rrbracket_{\Pi}, \llbracket (3 + 2) \rrbracket_{\Pi}) \quad \text{by Equation 5}$$

$$\text{Mul}(\llbracket 5 \rrbracket_{\Pi}, \llbracket (3 + 2) \rrbracket_{\Pi}) = \text{Mul}(\text{Num}(5), \llbracket (3 + 2) \rrbracket_{\Pi}) \quad \text{by Equation 1}$$

$$\text{Mul}(\text{num}(5), \llbracket (3 + 2) \rrbracket_{\Pi}) = \text{Mul}(\text{Num}(5), \text{Sum}(\llbracket 3 \rrbracket_{\Pi}, \llbracket 2 \rrbracket_{\Pi})) \quad \text{by Equation 3}$$

$$\text{Mul}(\text{Num}(5), \text{Sum}(\llbracket 3 \rrbracket_{\Pi}, \llbracket 2 \rrbracket_{\Pi})) = \text{Mul}(\text{Num}(5), \text{Sum}(\text{Num}(3), \llbracket 2 \rrbracket_{\Pi})) \quad \text{by Equation 1}$$

$$\text{Mul}(\text{Num}(5), \text{Sum}(\text{Num}(3), \llbracket 2 \rrbracket_{\Pi})) = \text{Mul}(\text{Num}(5), \text{Sum}(\text{Num}(3), \text{Num}(2))) \quad \text{by Equation 1}$$

A calculator: executing Π IR with Π automata I

A Π automaton is a 5-tuple $\mathcal{A} = (G, Q, \delta, q_0, F)$, where G is a context-free grammar, Q is the set of states, q_0 is the initial state, $F \subseteq Q$ is the set of final states and

$$\delta : L(G)^* \times L(G)^* \times Store \rightarrow Q,$$

where $L(G)$ is the language generated by G and $Store$ represents the memory. (Elements in a set S^* are represented by terms $[s_1, s_2, \dots, s_n]$.)

A calculator: executing Π IR with Π automata II

$$\begin{aligned}
 \delta([Mul(Num(5), Sum(Num(3), Num(2))), \emptyset, \emptyset) &= \delta([Num(5), Sum(Num(3), Num(2)), \#MUL], \emptyset, \emptyset) \\
 \delta([Num(5), Sum(Num(3), Num(2)), \#MUL], \emptyset, \emptyset) &= \delta([Sum(Num(3), Num(2)), \#MUL], [Num(5)], \emptyset) \\
 \delta([Sum(Num(3), Num(2)), \#MUL], [Num(5)], \emptyset) &= \delta([Num(3), Num(2), \#SUM, \#MUL], [Num(5)], \emptyset) \\
 \delta([Num(3), Num(2), \#SUM, \#MUL], [Num(5)], \emptyset) &= \delta([Num(2), \#SUM, \#MUL], [Num(3), Num(5)], \emptyset) \\
 \delta([Num(2), \#SUM, \#MUL], [Num(3), Num(5)], \emptyset) &= \delta([\#SUM, \#MUL], [Num(2), Num(3), Num(5)], \emptyset) \\
 \delta([\#SUM, \#MUL], [Num(2), Num(3), Num(5)], \emptyset) &= \delta([\#MUL], [Num(5), Num(5)], \emptyset) \\
 \delta([\#MUL], [Num(5), Num(5)], \emptyset) &= \delta(\emptyset, [Num(25)], \emptyset) \\
 \delta(\emptyset, [Num(25)], \emptyset) &= Num(25)
 \end{aligned}$$

Excerpt of Π IR expressions
$$\langle \textit{Statement} \rangle ::= \langle \textit{Exp} \rangle$$
$$\langle \textit{Exp} \rangle ::= \langle \textit{ArithExp} \rangle \mid \langle \textit{BoolExp} \rangle \langle \textit{Exp} \rangle$$
$$\langle \textit{ArithExp} \rangle ::= \text{'Num'}(\langle \textit{digits} \rangle) \mid \text{'Sum'}(\langle \textit{Exp} \rangle, \langle \textit{Exp} \rangle) \mid \\ \text{'Sub'}(\langle \textit{Exp} \rangle, \langle \textit{Exp} \rangle) \mid \text{'Mul'}(\langle \textit{Exp} \rangle, \langle \textit{Exp} \rangle)$$
$$\langle \textit{BoolExp} \rangle ::= \text{'Eq'}(\langle \textit{Exp} \rangle, \langle \textit{Exp} \rangle) \mid \text{'Not'}(\langle \textit{Exp} \rangle)$$

Π automaton semantics for Π IR expressions I

- Recall that $\delta : L(G)^* \times L(G)^* \times Store \rightarrow Q$, and let $N, N_i \in \mathbb{N}$, $C, V \in L(G)^*$, $S \in Store$,

$$\delta(Num(N) :: C, V, S) = \delta(C, Num(N) :: V, S) \quad (11)$$

$$\delta(Sum(E_1, E_2) :: C, V, S) = \delta(E_1 :: E_2 :: \#SUM :: C, V, S) \quad (12)$$

$$\delta(\#SUM :: C, Num(N_1) :: Num(N_2) :: V, S) = \delta(C, (N_1 + N_2) :: V, S) \quad (13)$$

...

$$\delta(Not(E) :: C, V, S) = \delta(E :: \#NOT :: C, V, S) \quad (14)$$

$$\delta(\#NOT :: C, Boo(true) :: V, S) = \delta(C, Boo(false) :: V, S) \quad (15)$$

$$\delta(\#NOT :: C, Boo(false) :: V, S) = \delta(C, Boo(true) :: V, S) \quad (16)$$

- Notation $h :: ls$ denotes the concatenation of element h with the list ls .
- C represents the *control* stack. V represents the *value* stack. S denotes the memory store.
- $\delta(\emptyset, V, S)$ denotes an *accepting state*.

Π automaton semantics for Π IR expressions II

- On a particular implementation of the Π Framework, as in Python, `<digits>` denote built-in numbers in implementation language, such that all arithmetic operations, such as `+`, are defined. That's why N_i are in \mathbb{N} .

Π IR commands

- Commands are language constructions that require a *memory* store to be evaluated.

$\langle \text{Statement} \rangle ::= \langle \text{Cmd} \rangle$

$\langle \text{Exp} \rangle ::= \text{'Id'}(\langle \text{String} \rangle)$

$\langle \text{Cmd} \rangle ::= \begin{array}{l} \text{'Assign'}(\langle \text{Id} \rangle, \langle \text{Exp} \rangle) \\ | \text{'Loop'}(\langle \text{BoolExp} \rangle, \langle \text{Cmd} \rangle) \\ | \text{'CSeq'}(\langle \text{Cmd} \rangle, \langle \text{Cmd} \rangle) \end{array}$

Π automaton semantics for Π IR commands I

- A location $l \in Loc$ denotes a memory cell.
- Storable and Bindable sets denote the data that may be mapped to by identifiers and locations on the memory and environment respectively.
- $Store = Loc \mapsto Storable$, $Env = Id \mapsto Bindable$, $Loc \subseteq Storable$, $\mathbb{N} \subseteq Loc, Bindable$.
- Now the transition function is $\delta : L(G)^* \times L(G)^* \times Env \times Store \rightarrow Q$, and let $W \in String$, $C, V \in L(G)^*$, $S \in Store$, $E \in Env$, $B \in Bindable$, $l \in Loc$, $T \in Storable$, $X \in \langle Exp \rangle$, $M, M_1, M_2 \in \langle Cmd \rangle$, and

Π automaton semantics for Π IR commands II

expression $S' = S/[I \mapsto N]$ means that S' equals to S in all indices but I that is bound to N ,

$$\delta(Id(W) :: C, V, E, S) = \delta(C, B :: V, E, S), \quad (17)$$

$$\text{where } E[W] = I \text{ and } S[I] = B,$$

$$\delta(Assign(W, X) :: C, V, E, S) = \delta(X :: \#ASSIGN :: C, W :: V, E, S), \quad (18)$$

$$\delta(\#ASSIGN :: C, T :: W :: V, E, S) = \delta(C, V, E, S'), \quad (19)$$

$$\text{where } E[W] = I \text{ and } S' = S/[I \mapsto T],$$

$$\delta(Loop(X, M) :: C, V, E, S) = \delta(X :: \#LOOP :: C, Loop(X, M) :: V, E, S), \quad (20)$$

$$\delta(\#LOOP :: C, Boo(true) :: Loop(X, M) :: V, E, S) = \delta(M :: Loop(X, M) :: C, V, E, S), \quad (21)$$

$$\delta(\#LOOP :: C, Boo(false) :: Loop(X, M) :: V, E, S) = \delta(C, V, E, S), \quad (22)$$

$$\delta(CSeq(M_1, M_2) :: C, V, E, S) = \delta(M_1 :: M_2 :: C, V, E, S). \quad (23)$$

Π IR declarations

- Declarations are statements that create an environment, binding identifiers to (bindable) values.
- In Π IR, a bindable value is either a Boolean value, an integer or a location.
- From a syntactic standpoint, all classes are monotonically extended.

$\langle \text{Statement} \rangle ::= \langle \text{Dec} \rangle$

$\langle \text{Exp} \rangle ::= \text{'Ref'}(\langle \text{Exp} \rangle) \mid \text{'DeRef'}(\langle \text{Id} \rangle)$

$\langle \text{Dec} \rangle ::= \text{'Bind'}(\langle \text{Id} \rangle, \langle \text{Exp} \rangle) \mid \text{'DSeq'}(\langle \text{Dec} \rangle, \langle \text{Dec} \rangle)$

$\langle \text{Cmd} \rangle ::= \text{'Blk'}(\langle \text{Dec} \rangle, \langle \text{Cmd} \rangle)$

Π automaton semantics for Π IR declarations I

Let $BlockLocs = \mathcal{P}(Loc)$, now the transition function is
 $\delta : L(G)^* \times L(G)^* \times Env \times Store \times BlockLocs \rightarrow Q$, and let $L, L' \in BlockLocs$,
 $Loc \subseteq Storable$, and S/L means the store S without the locations in L ,

$$\delta(Ref(X) :: C, V, E, S, L) = \delta(X :: \#REF :: C, V, E, S, L), \quad (24)$$

$$\delta(\#REF :: C, T :: V, E, S, L) = \delta(C, I :: V, E, S', L'), \text{ where } S' = S \cup [I \mapsto T], I \notin S, L' = L \cup \{I\}, \quad (25)$$

$$\delta(DeRef(Id(W)) :: C, V, E, S, L) = \delta(C, T :: V, E, S, L), \text{ where } I = E[W], T = S[I] \quad (26)$$

$$\delta(Bind(Id(W), X) :: C, V, E, S, L) = \delta(X :: \#BIND :: C, W :: V, E, S, L), \quad (27)$$

$$\delta(\#BIND :: C, B :: W :: V, E, S, L) = \delta(C, [W \mapsto B] :: V, E, S, L), \quad (28)$$

$$\delta(DSeq(D_1, D_2), X) :: C, V, E, S, L) = \delta(D_1 :: D_2 :: \#DSEQ :: C, V, E, S, L), \quad (29)$$

$$\delta(\#DSEQ :: C, E :: E' :: H :: V, E, S, L) = \delta(C, (E \cup E') :: H :: V, E, S, L), \text{ where } E' \in Env, H \notin Env, \quad (30)$$

$$\delta(Blk(D, M) :: C, V, E, S, L) = \delta(D :: \#BLKDEC :: C, M :: L :: V, E, S, \emptyset), \quad (31)$$

$$\delta(\#BLKDEC :: C, E' :: M :: V, E, S, L) = \delta(M :: \#BLKCMD :: C, E :: V, E \setminus E', S, L), \quad (32)$$

$$\delta(\#BLKCMD :: C, E :: L :: V, E', S, L') = \delta(C, V, E, S', L), \text{ where } S' = S/L'.$$

Π automaton semantics for Π IR declarations II

Π IR abstractions

- Abstractions extend Bindables by allowing a name to be bound to a list of formal parameters, a list of identifiers, and a block in the environment.
- Such names can be called and applied to actual parameters, a list of expressions.

$\langle Dec \rangle \quad ::= \text{'Bind'}(\langle Id \rangle, \langle Abs \rangle)$

$\langle Abs \rangle \quad ::= \text{'Abs'}(\langle Formals \rangle, \langle Blk \rangle)$

$\langle Formals \rangle \quad ::= \langle Id \rangle^*$

$\langle Cmd \rangle \quad ::= \text{'Call'}(\langle Id \rangle, \langle Actuals \rangle)$

$\langle Actuals \rangle \quad ::= \langle Exp \rangle^*$

Π automaton semantics for Π IR abstractions — Closures I

We chose a *static binding* semantics for abstractions. Therefore, we interpret abstractions as *closures* formed by an abstraction together with its declaration environment which defines the context in which the abstraction will be evaluated.

$$\text{Closure} : \text{Formals} \times \text{Blk} \times \text{Env} \rightarrow \text{Bindable}$$

Π automaton semantics for Π IR abstractions — Example I

```
1 # Iterative factorial
2 def fat (x) {
3     var z = x, y = 1 ;
4     while (z > 0)
5     {
6         y := y * z ;
7         z := z - 1 ;
8         print(y)
9     }
10 }
11 fat(2)
```

Π automaton semantics for Π IR abstractions — Example II

1 Π IR syntax tree:

```
2 Blk(  
3   BindAbs(Id(fat), Abs([Id(x)],  
4     Blk(Bind(Id(z), Ref(Id(x))),  
5       Blk(Bind(Id(y), Ref(1)),  
6         Loop(Gt(Id(z), 0),  
7           Blk(CSeq(CSeq(Assign(Id(y), Mul(Id(y), Id(z))), Assign(Id(z), Sub(Id(z), 1))),  
8             Print(Id(y)))))))))  
9   Call(Id(fat), [2]))
```

Π automaton semantics for Π IR abstractions I

Let $F \in \text{Formals}$, $B \in \text{Blk}$, $I \in \text{Id}$, $A \in \text{Actuals}$, $V_i \in \text{Value}$, $1 \leq i \leq n$, $n \in \mathbb{N}$,

$$\delta(\text{Abs}(F, B) :: C, V, E, S, L) = \delta(C, \text{Closure}(F, B, E) :: V, E, S, L) \quad (34)$$

$$\begin{aligned} \delta(\text{Call}(I, [X_1, X_2, \dots, X_n])) :: C, V, E, S, L) = \\ \delta(X_n :: X_{n-1} :: \dots :: X_1 :: \# \text{CALL}(I, n) :: C, V, E, S, L) \end{aligned} \quad (35)$$

$$\begin{aligned} \delta(\# \text{CALL}(I, n) :: C, (V_1 :: V_2 :: \dots V_n :: V), E, S, L) = \\ \delta(B :: \# \text{BLKCMD} :: C, E :: V, E', S, L) \end{aligned} \quad (36)$$

where $E = \{I \mapsto \text{Closure}(F, B, E_1)\} \cup E_2$,
 $E' = E_1 / \text{match}(F, [V_1, V_2, \dots, V_n])$

Π automaton semantics for Π IR abstractions II

$$match : Id^* \times Values^* \rightarrow Env$$

$$match(fl, al) = \text{if } |fl| \neq |al| \text{ then } \{\} \text{ else } _match(fl, al, \{\})$$

$$_match : Id^* \times Values^* \times Env \rightarrow Env$$

$$_match([], [], E) = E$$

$$_match(f, a, E) = \{f \mapsto a\} \cup E$$

$$_match(f :: fl, a :: al, E) = _match(fl, al, \{f \mapsto a\} \cup E)$$

Π IR recursive abstractions

- Abstractions can be recursive to allow for the declaration of recursive functions.

$\langle Dec \rangle \quad ::= \text{'Rbnd'}(\langle Id \rangle, \langle Abs \rangle)$

Π automaton semantics for Π IR recursive abstractions — Recursive closures I

In the context of *static binding* semantics for abstractions, in a call to a recursive function, the evaluation of identifiers needs to be reminded about the binding of the function name to a closure.

$$Rec : Formals \times Blk \times Env \times Env \rightarrow Bindable$$
$$unfold : Env \rightarrow Env$$
$$reclose_E : Env \rightarrow Env$$

Π automaton semantics for Π IR recursive abstractions — Recursive closures II

$$\text{unfold}(E) = \text{reclose}_E(E) \quad (37)$$

$$\text{reclose}_E(I \mapsto \text{Closure}(F, B, E')) = (I \mapsto \text{Rec}(F, B, E', E)) \quad (38)$$

$$\text{reclose}_E(I \mapsto \text{Rec}(F, B, E', E'')) = (I \mapsto \text{Rec}(F, B, E', E)) \quad (39)$$

$$\text{reclose}_E(I \mapsto v) = (I \mapsto v) \text{ if } v \neq \text{Closure}(F, B, E) \quad (40)$$

$$\text{reclose}_E(E_1 \cup E_2) = \text{reclose}_E(E_1) \cup \text{reclose}_E(E_2) \quad (41)$$

$$\text{reclose}_E(\emptyset) = \emptyset \quad (42)$$

Π automaton semantics for Π IR recursive abstractions — Recursive closures I

$$\begin{aligned} \delta(Rbnd(I, Abs(F, B)) :: C, V, E, S, L) = \\ \delta(C, unfold(I \mapsto Closure(F, B, E)) :: V, E, S, L) \end{aligned} \quad (43)$$

$$\begin{aligned} \delta(\#CALL(I, n) :: C, V_1 :: V_2 :: \dots :: V_n :: V, E, S, L) = \\ \delta(B :: \#BLKCMD :: C, E :: L :: V, E', S, \emptyset) \end{aligned} \quad (44)$$

$$\begin{aligned} \text{where } E &= \{I \mapsto Rec(F, B, E_1, E_2)\} \cup E_3, \\ E' &= E_1 / unfold(E_2) / match(F, [V_1, V_2, \dots, V_n]) \end{aligned}$$