Notes on Formal Compiler Construction with the π Framework

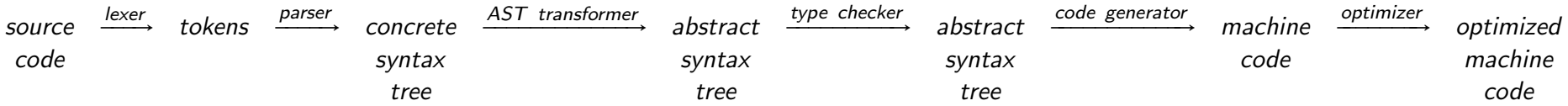
Christiano Braga

Instituto de Computação,
Universidade Federal Fluminense, Niterói, Brazil

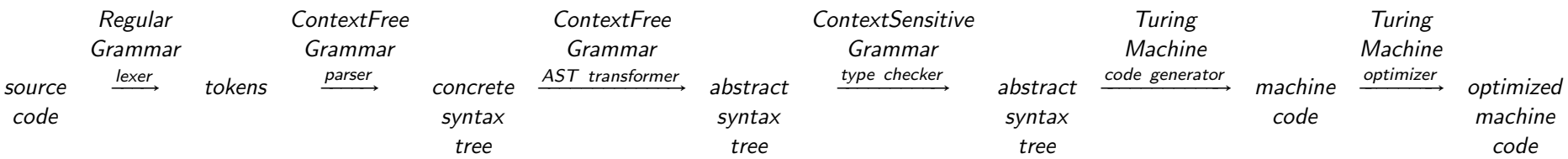
August 14, 2018

http://github.com/ChristianoBraga/BPLC

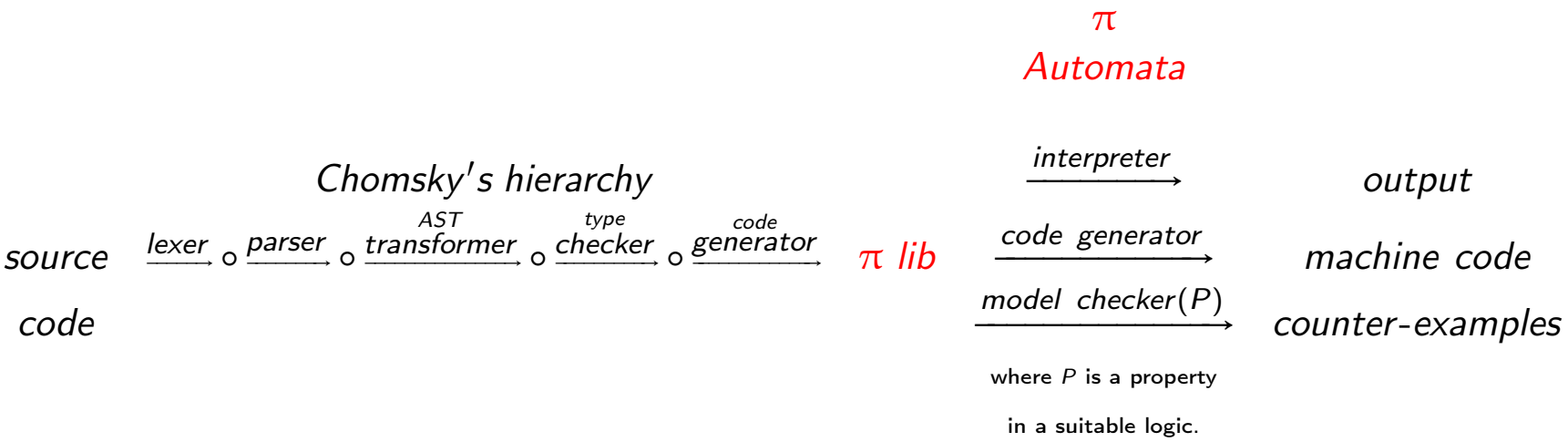
Compiler pipeline



Compiler pipeline and formal languages



Compiler pipeline with the π Framework



- π lib defines a set of constructions common to many programming languages.
- π lib 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 lib program.
- π Framework: `http://github.com/ChristianoBraga/BPLC`
- Notes on Formal Compiler Construction with the π Framework: `https://github.com/ChristianoBraga/BPLC/blob/master/notes/notes.pdf`.

Instituto de Computação

A calculator

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

A calculator: Lexer

$\langle digit \rangle \quad ::= \quad [0..9]$

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

$\langle boolean \rangle \; ::= \; \text{`}\texttt{true}\text{'} \mid \text{`}\texttt{false}\text{'}$

Instituto de computação

A calculator: concrete syntax

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

Instituto de computação

A calculator: abstract syntax

$\langle exp \rangle$ ::= $\langle digits \rangle$ | $\langle boolean \rangle$ | $\langle exp \rangle$ $\langle bop \rangle$ $\langle exp \rangle$

$\langle bop \rangle$ ::= '+' | '-' | '*' | '|' | '/' | '=' | '<' | '>' | '<=' | '>='

Instituto de Computação

A calculator: π denotations I

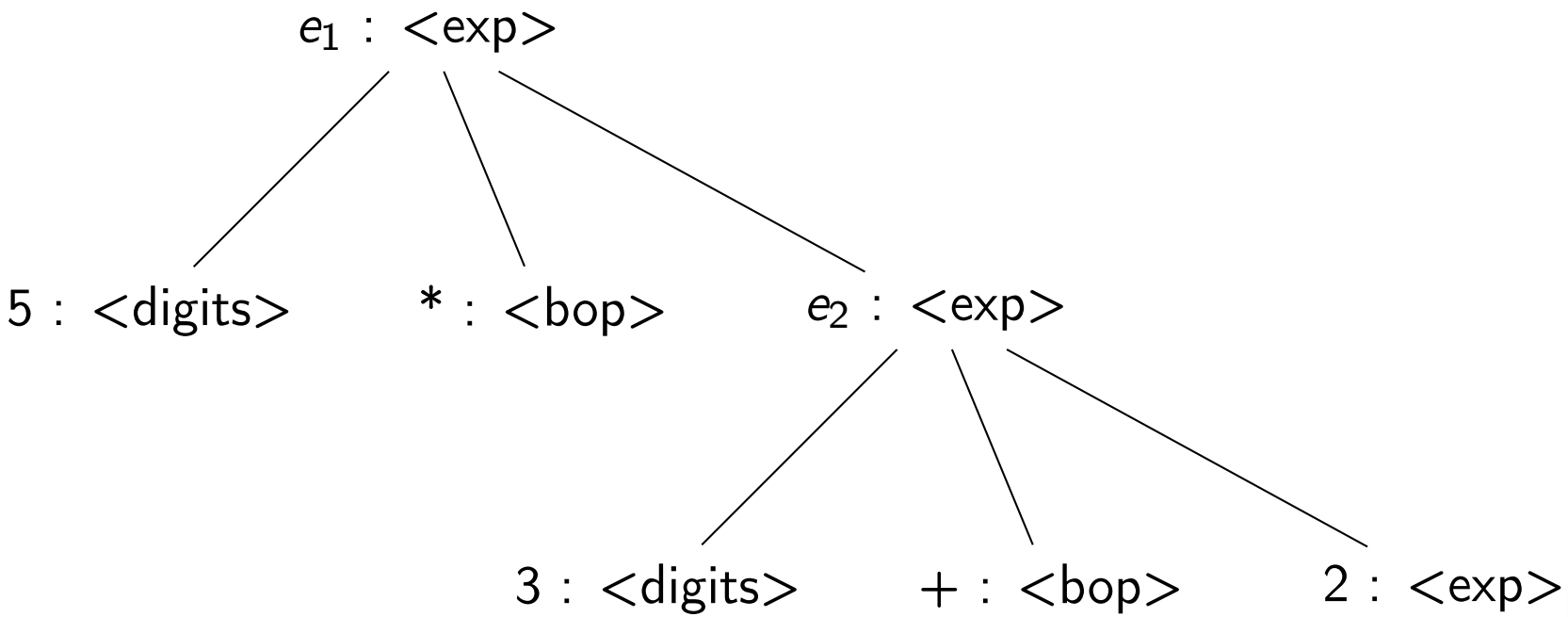
Let D in <digits>, B in <boolean> and E_1, E_2 in <exp>,

$$[\![D]\!]_\pi = num(D) \tag{1}$$
$$[\![B]\!]_\pi = boo(B) \tag{2}$$
$$[\![E_1 + E_2]\!]_\pi = add([\![E_1]\!]_\pi, [\![E_2]\!]_\pi) \tag{3}$$
$$[\![E_1 - E_2]\!]_\pi = sub([\![E_1]\!]_\pi, [\![E_2]\!]_\pi) \tag{4}$$
$$[\![E_1 * E_2]\!]_\pi = mul([\![E_1]\!]_\pi, [\![E_2]\!]_\pi) \tag{5}$$
$$[\![E_1 / E_2]\!]_\pi = div([\![E_1]\!]_\pi, [\![E_2]\!]_\pi) \tag{6}$$
$$[\![E_1 < E_2]\!]_\pi = lt([\![E_1]\!]_\pi, [\![E_2]\!]_\pi) \tag{7}$$
$$[\![E_1 <= E_2]\!]_\pi = le([\![E_1]\!]_\pi, [\![E_2]\!]_\pi) \tag{8}$$
$$[\![E_1 > E_2]\!]_\pi = gt([\![E_1]\!]_\pi, [\![E_2]\!]_\pi) \tag{9}$$
$$[\![E_1 >= E_2]\!]_\pi = ge([\![E_1]\!]_\pi, [\![E_2]\!]_\pi) \tag{10}$$

Instituto de Computação

A calculator: π denotations II

- π denotations are *functions* $[\![\cdot]\!]_\pi : AST \rightarrow \pi\ lib$, where AST denotes the *datatype* for the abstract syntax tree and π *lib* denotes the datatype for π *lib* programs.
- Note that $[\![\cdot]\!]_\pi$ has *trees* as parameters, instances of AST. The example expression $5 * (3 + 2)$ becomes



A calculator: π denotations III

$$[\![5*(3+2)]\!]_\pi = mul([\![5]\!]_\pi, [\![(3+2)]\!]_\pi) \quad \textit{by Equation } 5$$
$$mul([\![5]\!]_\pi, [\![(3+2)]\!]_\pi) = mul(num(5), [\![(3+2)]\!]_\pi) \quad \textit{by Equations } 1$$
$$mul(num(5), [\![(3+2)]\!]_\pi) = mul(num(5), add([\![3]\!]_\pi, [\![2]\!]_\pi) \quad \textit{by Equation } 3$$
$$mul(num(5), add([\![3]\!]_\pi, [\![2]\!]_\pi) = mul(num(5), add(num(3), [\![2]\!]_\pi) \quad \textit{by Equation } 1$$
$$mul(num(5), add(num(3), [\![2]\!]_\pi) = mul(num(5), add(num(3), num(2)) \quad \textit{by Equation } 1$$

Instituto de Computação

A calculator: executing π lib with π automata

A π automaton is a 5-tuple $\mathscr{A} = (G, Q, \delta, q_0, F)$, 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 $[s_1, s_2, \ldots, s_n]$.)

$$\begin{aligned}
\delta([mul(num(5), add(num(3), num(2)]), \emptyset, \emptyset) &= \delta([num(5), add(num(3), num(2)), \#MUL], \emptyset, \emptyset) \\
\delta([num(5), add(num(3), num(2)), \#MUL], \emptyset, \emptyset) &= \delta([add(num(3), num(2)), \#MUL], [num(5)], \emptyset) \\
\delta([add(num(3), num(2)), \#MUL], [num(5)], \emptyset) &= \delta([num(3), num(2), \#SUM, \#MUL], [num(5)], \emptyset) \\
\delta([num(3), num(2), \#SUM, \#MUL], [num(5)], \emptyset) &= \delta([num(2), \#SUM, \#MUL], [num(3), num(5)], \emptyset) \\
\delta([num(2), \#SUM, \#MUL], [num(3), num(5)], \emptyset) &= \delta([\#SUM, \#MUL], [num(2), num(3), num(5)], \emptyset) \\
\delta([\#SUM, \#MUL], [num(2), num(3), num(5)], \emptyset) &= \delta([\#MUL], [num(5), num(5)], \emptyset) \\
\delta([\#MUL], [num(5), num(5)], \emptyset) &= \delta(\emptyset, [num(25)], \emptyset) \\
\delta(\emptyset, [num(25)], \emptyset) &= num(25)
\end{aligned}$$

Instituto de Computação

π lib expressions in Python I

```
1  class Statement:
2      def __init__(self, *args):
3          self.opr =args
4      def __str__(self):
5          ret =str(self.__class__.__name__)+"("
6          for o in self.opr:
7              ret +=str(o)
8          ret +=")"
9          return ret
10 class Exp(Statement): pass
11 class ArithExp(Exp): pass
```

Instituto de Computação

π lib expressions in Python II

```
1 class Num(ArithExp):
2     def __init__(self, f):
3         assert(isinstance(f, int))
4         ArithExp.__init__(self,f)
5 class Sum(ArithExp):
6     def __init__(self, e1, e2):
7         assert(isinstance(e1, Exp) and isinstance(e2, Exp))
8         ArithExp.__init__(self, e1, e2)
9 ...
```

Instituto de Computação

π lib expressions in Python III

```
1 class BoolExp(Exp): pass
2 class Eq(BoolExp):
3     def __init__(self, e1, e2):
4         assert(isinstance(e1, Exp) and isinstance(e2, Exp))
5         BoolExp.__init__(self, e1, e2)
6 ...
```

Instituto de Computação

π lib expressions in Python IV

```
1 exp =Sum(Num(1), Mul(Num(2), Num(4)))
2 print(exp)
3
4 Sum(Num(1)Mul(Num(2)Num(4)))
```

π lib expressions in Python V

```
exp2 =Mul(2, 1)
---------------------------------------------------------------------------
                                          ---
AssertionError Traceback (most recent call last)
<ipython-input-7-00fd40a79a54> in <module>()
---->1 exp2 =Mul(2, 1)

<ipython-input-5-42a82e58862f> in __init__(self, e1, e2)
     28 class Mul(ArithExp):
     29 def __init__(self, e1, e2):
--->30 assert(isinstance(e1, Exp) and isinstance(e2, Exp))
     31 ArithExp.__init__(self, e1, e2)
     32 class BoolExp(Exp): pass

AssertionError:
```

Instituto de computação

π automaton for π lib expressions I

```
1 ## Expressions
2 class ValueStack(list): pass
3 class ControlStack(list): pass
4 class ExpKW:
5     SUM ="#SUM"
6     SUB ="#SUB"
7     MUL ="#MUL"
8     EQ ="#EQ"
9     NOT ="#NOT"
```

Instituto de Computação

π automaton for π lib expressions II

```
1  class ExpPiAut(dict):
2      def __init__(self):
3          self["val"] =ValueStack()
4          self["cnt"] =ControlStack()
5      def __evalSum(self, e):
6          e1 =e.opr[0]
7          e2 =e.opr[1]
8          self.pushCnt(ExpKW.SUM)
9          self.pushCnt(e1)
10         self.pushCnt(e2)
11     def pushCnt(self, e):
12         cnt =self.cnt()
13         cnt.append(e)
14 ...
```

Instituto de Computação

π automaton for π lib expressions III

```
1 ea =ExpPiAut()
2 print(exp)
3 ea.pushCnt(exp)
4 while not ea.emptyCnt():
5     ea.eval()
6     print(ea)
```

Instituto de Computação

π automaton for π lib expressions IV

```
1 Sum(Num(1)Mul(Num(2)Num(4)))
2 {'val': [], 'cnt': ['#SUM', <__main__.Num object at 0x11851470>, <
                                __main__.Mul object at 0x1118516d8>]
                                }
3 {'val': [], 'cnt': ['#SUM', <__main__.Num object at 0x11851470>, '#MUL'
                                , <__main__.Num object at
                                0x11851630>, <__main__.Num object
                                at 0x1118516a0>]}
4 {'val': [4], 'cnt': ['#SUM', <__main__.Num object at 0x11851470>, '#MUL
                                ', <__main__.Num object at
                                0x11851630>]}
5 {'val': [4, 2], 'cnt': ['#SUM', <__main__.Num object at 0x11851470>, '#
                                MUL']}
6 {'val': [8], 'cnt': ['#SUM', <__main__.Num object at 0x11851470>]}
7 {'val': [8, 1], 'cnt': ['#SUM']}
8 {'val': [9], 'cnt': []}
```

Instituto de Computação