

Notes on Formal Compiler Construction with the π Framework

Christiano Braga

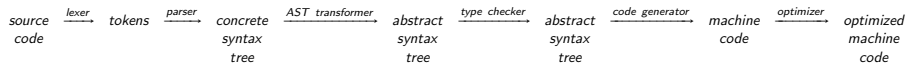
Instituto de Computação,
Universidade Federal Fluminense, Niterói, Brazil

August 14, 2018

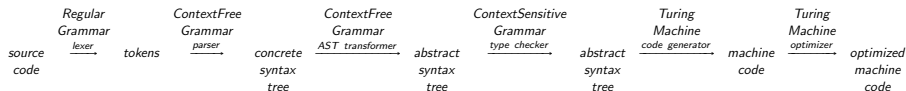
<http://github.com/ChristianoBraga/BPLC>



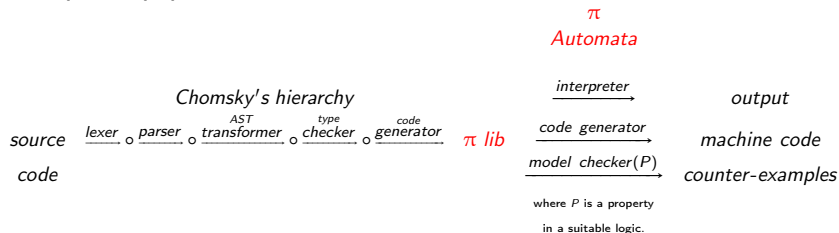
Compiler pipeline



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/ChristianBraga/BPLC>
- Notes on Formal Compiler Construction with the π Framework: <https://github.com/ChristianBraga/BPLC/blob/master/notes/notes.pdf>.

A calculator

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

A calculator: Lexer

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

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

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

A calculator: concrete syntax

$$\langle exp \rangle ::= \langle aexp \rangle \mid \langle bexp \rangle$$
$$\langle aexp \rangle ::= \langle aexp \rangle '+' \langle term \rangle \mid \langle aexp \rangle '-' \langle term \rangle \mid \langle term \rangle$$
$$\langle term \rangle ::= \langle term \rangle '*' \langle factor \rangle \mid \langle term \rangle '/' \langle factor \rangle \mid \langle factor \rangle$$
$$\langle factor \rangle ::= '(' \langle aexp \rangle ')' \mid \langle digits \rangle$$
$$\langle bexp \rangle ::= \langle boolean \rangle \mid \sim \langle bexp \rangle \mid \langle bexp \rangle \langle boolop \rangle \langle bexp \rangle \\ \mid \langle aexp \rangle \langle iop \rangle \langle aexp \rangle$$
$$\langle iop \rangle ::= '=' \mid '<' \mid '>' \mid '<=' \mid '>='$$

A calculator: abstract syntax

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

A calculator: π denotations I

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

$$\llbracket D \rrbracket_{\pi} = \text{num}(D) \quad (1)$$

$$\llbracket B \rrbracket_{\pi} = \text{boo}(B) \quad (2)$$

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

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

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

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

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

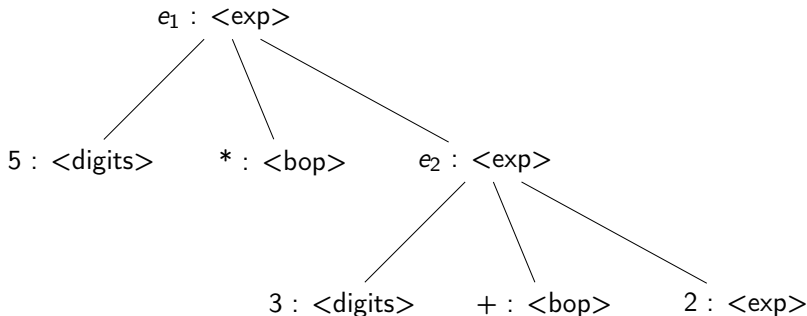
$$\llbracket E_1 \leq E_2 \rrbracket_{\pi} = \text{le}(\llbracket E_1 \rrbracket_{\pi}, \llbracket E_2 \rrbracket_{\pi}) \quad (8)$$

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

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

A calculator: π denotations II

- π denotations are *functions* $\llbracket \cdot \rrbracket_{\pi} : AST \rightarrow \pi \text{ lib}$, where *AST* denotes the *datatype* for the abstract syntax tree and $\pi \text{ lib}$ denotes the datatype for $\pi \text{ lib}$ 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

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

$$\text{mul}(\llbracket 5 \rrbracket_{\pi}, \llbracket (3 + 2) \rrbracket_{\pi}) = \text{mul}(\text{num}(5), \llbracket (3 + 2) \rrbracket_{\pi}) \quad \text{by Equations 1}$$

$$\text{mul}(\text{num}(5), \llbracket (3 + 2) \rrbracket_{\pi}) = \text{mul}(\text{num}(5), \text{add}(\llbracket 3 \rrbracket_{\pi}, \llbracket 2 \rrbracket_{\pi})) \quad \text{by Equation 3}$$

$$\text{mul}(\text{num}(5), \text{add}(\llbracket 3 \rrbracket_{\pi}, \llbracket 2 \rrbracket_{\pi})) = \text{mul}(\text{num}(5), \text{add}(\text{num}(3), \llbracket 2 \rrbracket_{\pi})) \quad \text{by Equation 1}$$

$$\text{mul}(\text{num}(5), \text{add}(\text{num}(3), \llbracket 2 \rrbracket_{\pi})) = \text{mul}(\text{num}(5), \text{add}(\text{num}(3), \text{num}(2))) \quad \text{by Equation 1}$$

A calculator: executing π lib with π automata

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 $[s_1, s_2, \dots, s_n]$.)

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