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

$$\textit{source code} \xrightarrow{lexer} \textit{tokens} \xrightarrow{parser} \textit{concrete syntax tree} \xrightarrow{AST\ transformer} \textit{abstract syntax tree} \xrightarrow{type\ checker} \textit{abstract syntax tree} \xrightarrow{code\ generator} \textit{machine code} \xrightarrow{optimizer} \textit{optimized machine code}$$

Compiler pipeline and formal languages

$$\text{source code} \overset{\text{Regular Grammar}}{\xrightarrow{\text{lexer}}} \text{tokens} \overset{\text{ContextFree Grammar}}{\xrightarrow{\text{parser}}} \text{concrete syntax tree} \overset{\text{ContextFree Grammar}}{\xrightarrow{\text{AST transformer}}} \text{abstract syntax tree} \overset{\text{ContextSensitive Grammar}}{\xrightarrow{\text{type checker}}} \text{abstract syntax tree} \overset{\text{Turing Machine}}{\xrightarrow{\text{code generator}}} \text{machine code} \overset{\text{Turing Machine}}{\xrightarrow{\text{optimizer}}} \text{optimized machine code}$$

Compiler pipeline with the π Framework

π *Automata*

Chomsky's hierarchy

$$\textit{source code} \xrightarrow{\textit{lexer}} \circ \xrightarrow{\textit{parser}} \circ \xrightarrow{\textit{AST transformer}} \circ \xrightarrow{\textit{type checker}} \circ \xrightarrow{\textit{code generator}} \pi\ \textit{lib}$$

$$\pi\ \textit{lib} \xrightarrow{\textit{interpreter}} \textit{output}$$

$$\pi\ \textit{lib} \xrightarrow{\textit{code generator}} \textit{machine code}$$

$$\pi\ \textit{lib} \xrightarrow{\textit{model checker}(P)} \textit{counter-examples}$$

where P is a property in a suitable logic.

- 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`.

A calculator

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

A calculator: Lexer

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

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

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

A calculator: concrete syntax

$$
\begin{array}{lcl}
\langle exp \rangle & ::= & \langle aexp \rangle \mid \langle bexp \rangle \\[1em]
\langle aexp \rangle & ::= & \langle aexp \rangle \text{ `+' } \langle term \rangle \mid \langle aexp \rangle \text{ `-' } \langle term \rangle \mid \langle term \rangle \\[1em]
\langle term \rangle & ::= & \langle term \rangle \text{ `*' } \langle factor \rangle \mid \langle term \rangle \text{ `/' } \langle factor \rangle \mid \langle factor \rangle \\[1em]
\langle factor \rangle & ::= & \text{`('} \ \langle aexp \rangle \ \text{`)'} \mid \langle digits \rangle \\[1em]
\langle bexp \rangle & ::= & \langle boolean \rangle \mid \text{`\textasciitilde'} \ \langle bexp \rangle \mid \langle bexp \rangle \ \langle boolop \rangle \ \langle bexp \rangle \\
& \mid & \langle aexp \rangle \ \langle iop \rangle \ \langle aexp \rangle \\[1em]
\langle iop \rangle & ::= & \text{`='} \mid \text{`<'} \mid \text{`>'} \mid \text{`<='} \mid \text{`>='}
\end{array}
$$

A calculator: abstract syntax

$$
\begin{aligned}
\langle \mathit{exp} \rangle \quad & ::= \; \langle \mathit{digits} \rangle \mid \langle \mathit{boolean} \rangle \mid \langle \mathit{exp} \rangle \; \langle \mathit{bop} \rangle \; \langle \mathit{exp} \rangle \\
\langle \mathit{bop} \rangle \quad & ::= \; \text{`+'} \mid \text{`-'} \mid \text{`}*\text{'} \mid \text{`|'} \mid \text{`/'} \mid \text{`='} \mid \text{`<'} \mid \text{`>'} \mid \text{`<='} \mid \text{`>='}
\end{aligned}
$$

A calculator: π denotations I

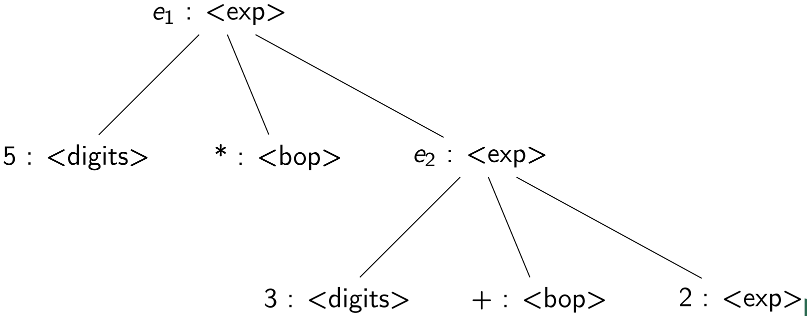
Let D in <digits>, B in <boolean> and E_1, E_2 in <exp>,

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

Instituto de computação

A calculator: π denotations II

- π denotations are *functions* $[\![\cdot]\!]_\pi : AST \to \pi$ *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

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

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

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

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

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