Programming Language Concepts

This course teaches concepts in two ways:

- By implementing interpreters
  - new concept => extend interpreter
- By using Scheme
  - we assume that you don’t already know Scheme
Course Details

http://www.cs.utah.edu/classes/cs3520/
Bootstrapping Problem

• We'll learn about languages by writing interpreters in Scheme

• We'll learn about Scheme... by writing an interpreter...

  in Scheme set theory

• More specifically, we'll define Scheme as an extension of algebra

  Algebra is a programming language?
Algebra as a Programming Language

- Algebra has a grammar:
  - $(1 + 2)$ is a legal expression
  - $(1 + +)$ is not a legal expression

- Algebra has rules for evaluation:
  - $(1 + 2) = 3$
  - $f(17) = (17 + 3) = 20$ if $f(x) = (x + 3)$
A Grammar for Algebra Programs

The grammar in BNF (Backus-Naur Form; EoPL sec 1.1.2):

\[
\begin{align*}
<\text{prog}> & ::= <\text{defn}>^* <\text{expr}> \\
<\text{defn}> & ::= <\text{id}>(<\text{id}> ) = <\text{expr}> \\
<\text{expr}> & ::= ( <\text{expr}> + <\text{expr}> ) \\
& ::= ( <\text{expr}> - <\text{expr}> ) \\
& ::= <\text{id}>(<\text{expr}> ) \\
& ::= <\text{id}> | <\text{num}> \\
<\text{id}> & ::= \text{a variable name: } f, x, y, z, \ldots \\
<\text{num}> & ::= \text{a number: } 1, 42, 17, \ldots \\
\end{align*}
\]

• Each \textit{meta-variable}, such as \textit{<prog>}, defines a set
Using a BNF Grammar

\[
\begin{align*}
\text{id} & ::= \text{ a variable name: } f, x, y, z, \ldots \\
\text{num} & ::= \text{ a number: } 1, 42, 17, \ldots \\
\end{align*}
\]

• The set \( \text{id} \) is the set of all variable names

• The set \( \text{num} \) is the set of all numbers

• To make an example member of \( \text{num} \), simply pick an element from the set

\[
1 \in \text{num}
\]

\[
198 \in \text{num}
\]
Using a BNF Grammar

\[
\text{<expr>} ::= ( \text{<expr>} + \text{<expr>} ) \\
::= ( \text{<expr>} - \text{<expr>} ) \\
::= \text{id}(\text{<expr>}) \\
::= \text{id} | \text{<num>}
\]

- The set \text{<expr>} is defined in terms of other sets
Using a BNF Grammar

<expr> ::= ( <expr> + <expr> )
 ::= ( <expr> - <expr> )
 ::= <id>(<expr>)
 ::= <id> | <num>

• To make an example <expr>:
  ○ choose one case in the grammar
  ○ pick an example for each meta-variable
  ○ combine the examples with literal text
Using a BNF Grammar

\[
<\text{expr}> ::= ( <\text{expr}> + <\text{expr}> ) \\
 ::= ( <\text{expr}> - <\text{expr}> ) \\
 ::= \langle \text{id} \rangle ( <\text{expr}> ) \\
 ::= \langle \text{id} \rangle \mid \langle \text{num} \rangle
\]

- To make an example \(<\text{expr}>\):
  - choose one case in the grammar
  - pick an example for each meta-variable
    - \(7 \in \langle \text{num} \rangle\)
  - combine the examples with literal text
    - \(7 \in \langle \text{expr} \rangle\)
Using a BNF Grammar

\[ \text{<expr>} ::= (\text{<expr>} + \text{<expr>}) \]
\[ ::= (\text{<expr>} - \text{<expr>}) \]
\[ ::= \text{id} (\text{<expr>}) \]
\[ ::= \text{id} | \text{num} \]

To make an example \text{<expr>}:

- choose one case in the grammar
- pick an example for each meta-variable
  \[ \text{f} \in \text{id} \quad 7 \in \text{<expr>} \]
- combine the examples with literal text
  \[ \text{f}(7) \in \text{<expr>} \]
Using a BNF Grammar

<expr> ::= ( <expr> + <expr> )
 ::= ( <expr> - <expr> )
 ::= <id>(<expr>)
 ::= <id> | <num>

• To make an example <expr>:
  ○ choose one case in the grammar
  ○ pick an example for each meta-variable

    f ∈ <id>         f(7) ∈ <expr>

  ○ combine the examples with literal text

    f(f(7)) ∈ <expr>
Using a BNF Grammar

\[
\text{<prog>} ::= \text{<defn>}\star \text{<expr>}
\]

\[
\text{<defn>} ::= \text{id}\langle\text{id}\rangle = \text{<expr>}
\]

\[
f(x) = (x + 1) \in \text{<defn>}
\]

- To make a \text{<prog>} pick some number of \text{<defn>}s

\[
(x + y) \in \text{<prog>}
\]

\[
f(x) = (x + 1)
\]

\[
g(y) = f((y - 2)) \in \text{<prog>}
\]

\[
g(7)
\]
Demonstrating Set Membership

• We can run the element-generation process in reverse to prove that some item is a member of a set

• Such proofs have a standard tree format:

\[
\begin{align*}
&\text{sub-claim to prove} & \ldots & \text{sub-claim to prove} \\
&\text{claim to prove}
\end{align*}
\]

• Immediate membership claims serve as leaves on the tree:

\[7 \in <\text{num}>\]
Demonstrating Set Membership

- We can run the element-generation process in reverse to prove that some item is a member of a set.

- Such proofs have a standard tree format:

  \[
  \text{sub-claim to prove} \quad \ldots \quad \text{sub-claim to prove}
  \]

  claim to prove

- Immediate membership claims serve as leaves on the tree:

  \[ f \in <\text{id}> \]
Demonstrating Set Membership

• We can run the element-generation process in reverse to prove that some item is a member of a set

• Such proofs have a standard tree format:

\[
\text{sub-claim to prove} \quad \ldots \quad \text{sub-claim to prove} \\
\hline
\text{claim to prove}
\]

• Other membership claims generate branches in the tree:

\[
7 \in \text{<num>} \\
\hline
7 \in \text{<expr>}
\]
Demonstrating Set Membership

- We can run the element-generation process in reverse to prove that some item is a member of a set.

- Such proofs have a standard tree format:

  \[
  \begin{array}{c}
  \text{sub-claim to prove} \\
  \text{...}
  \end{array}
  \]

  \[
  \begin{array}{c}
  \text{sub-claim to prove} \\
  \text{claim to prove}
  \end{array}
  \]

- Other membership claims generate branches in the tree:

  \[
  \begin{array}{c}
  f \in <id> \\
  7 \in <num>
  \end{array}
  \]

  \[
  \begin{array}{c}
  f(7) \in <expr>
  \end{array}
  \]

The proof tree's shape is driven entirely by the grammar.
Demonstrating Set Membership: Example

\[ f(7) \in <expr> \]

\[
<expr> ::= ( <expr> + <expr> )
\]
\[
 ::= ( <expr> - <expr> )
\]
\[
 ::= <id>(<expr>)
\]
\[
 ::= <id> | <num>
\]

- Two meta-variables on the left means two sub-trees:
  - One for \( f \in <id> \)
  - One for \( 7 \in <expr> \)
Demonstrating Set Membership: Example

\[ f \in \langle \text{id} \rangle \quad 7 \in \langle \text{expr} \rangle \]
\[ f(7) \in \langle \text{expr} \rangle \]

\[ \langle \text{id} \rangle \ ::= \text{a variable name: } f, x, y, z, \ldots \]
\[ \langle \text{expr} \rangle \ ::= (\langle \text{expr} \rangle + \langle \text{expr} \rangle) \]
\[ ::= (\langle \text{expr} \rangle - \langle \text{expr} \rangle) \]
\[ ::= \langle \text{id} \rangle (\langle \text{expr} \rangle) \]
\[ ::= \langle \text{id} \rangle \mid \langle \text{num} \rangle \]

- \( f \in \langle \text{id} \rangle \) is immediate
- \( 7 \in \langle \text{expr} \rangle \) has one meta-variable, so one subtree
Demonstrating Set Membership: Example

\[
\begin{align*}
  & f \in \langle \text{id} \rangle \\
  & 7 \in \langle \text{num} \rangle \\
  \hline
  & f \in \langle \text{id} \rangle \\
  & 7 \in \langle \text{expr} \rangle \\
  \hline
  & f(7) \in \langle \text{expr} \rangle
\end{align*}
\]

\[
\langle \text{num} \rangle \ ::= \text{a number: } 1, 42, 17, \ldots
\]

- \( 7 \in \langle \text{num} \rangle \) is immediate, so the proof is complete
Demonstrating Set Membership: Another Example

\[
\begin{align*}
  f(x) &= (x + 1) \\
  g(y) &= f((y - 2)) \in \langle\text{prog}\rangle \\
  g(7) &
\end{align*}
\]

\[
\text{\langle prog\rangle ::= \langle defn\rangle^* \langle expr\rangle}
\]

- Three meta-variables (after expanding *) means three sub-trees:
  - One for \( f(x) = (x + 1) \in \langle\text{defn}\rangle \)
  - One for \( g(y) = f((y - 2)) \in \langle\text{defn}\rangle \)
  - One for \( g(7) \in \langle\text{expr}\rangle \)
Demonstrating Set Membership: Example 2

| f(x) = (x + 1) ∈ <defn> | g(y) = f((y - 2)) ∈ <defn> | g(7) ∈ <expr> |

- f(x) = (x + 1)
- g(y) = f((y - 2)) ∈ <prog>
- g(7)

- Each sub-tree can be proved separately
- We'll prove only the first sub-tree for now
Demonstrating Set Membership: Example 2

\[ f(x) = (x + 1) \in \text{<defn>} \]

\[ \text{<defn> ::= <id>(<id>) = <expr>} \]

- Three meta-variables, three sub-trees
Demonstrating Set Membership: Example 2

\[ f \in <id> \quad x \in <id> \quad (x + 1) \in <expr> \]

\[ f(x) = (x + 1) \in <defn> \]

- The first two are immediate, the last requires work:

\[
\begin{align*}
<\text{expr}> & \ ::= \ (<\text{expr}> + <\text{expr}> )  \\
& \ ::= \ (<\text{expr}> - <\text{expr}> )  \\
& \ ::= \ <\text{id}>(<\text{expr}> )  \\
& \ ::= \ <\text{id}> \mid <\text{num}>  \\
\end{align*}
\]
Demonstrating Set Membership: Example 2

Final tree:

\[
f \in \langle \text{id} \rangle \quad x \in \langle \text{id} \rangle \quad \frac{x \in \langle \text{id} \rangle}{x \in \langle \text{expr} \rangle} \quad \frac{1 \in \langle \text{num} \rangle}{1 \in \langle \text{expr} \rangle} \quad \frac{x \in \langle \text{expr} \rangle}{(x + 1) \in \langle \text{expr} \rangle} \quad f(x) = (x + 1) \in \langle \text{defn} \rangle
\]

• This was just one of three sub-trees for the original \(\in\langle \text{prog} \rangle\) proof...
Algebra as a Programming Language

- Algebra has a grammar:
  - (1 + 2) is a legal expression
  - (1 + +) is not a legal expression

- Algebra has rules for evaluation:
  - (1 + 2) = 3
  - f(17) = (17 + 3) = 20  if  f(x) = (x + 3)
Evaluation Function

- An *evaluation function*, \(\rightarrow\), takes a single evaluation step.
- It maps programs to programs:

\[
(2 + (7 - 4)) \rightarrow (2 + 3)
\]
Evaluation Function

- An **evaluation function**, \( \rightarrow \), takes a single evaluation step.
- It maps programs to programs:

\[
f(x) = (x + 1) \quad \rightarrow \quad f(x) = (x + 1)\\
(2 + (7 - 4)) \quad \rightarrow \quad (2 + 3)
\]
Evaluation Function

• An evaluation function, \(\rightarrow\), takes a single evaluation step

• It maps programs to programs:

\[
\begin{align*}
  f(x) &= (x + 1) & \rightarrow & & f(x) &= (x + 1) \\
  g(y) &= (y - 1) & & g(y) &= (y - 1) \\
  h(z) &= f(z) & & h(z) &= f(z) \\
  (2 + f(13)) & & (2 + (13 + 1))
\end{align*}
\]
Evaluation Function

- Apply $\rightarrow$ repeatedly to obtain a result:

\[
\begin{align*}
  \mathbf{f}(\mathbf{x}) &= (\mathbf{x} + 1) \
  (2 + (7 - 4)) &\rightarrow (2 + 3) \\
  \mathbf{f}(\mathbf{x}) &= (\mathbf{x} + 1) \
  (2 + 3) &\rightarrow 5
\end{align*}
\]
Evaluation Function

• The function is defined by a set of pattern-matching rules:

\[ f(x) = (x + 1) \rightarrow f(x) = (x + 1) \]
\[ (2 + (7 - 4)) \rightarrow (2 + 3) \]

due to the pattern rule

\[ ... (7 - 4) ... \rightarrow ... 3 ... \]
Evaluation Function

- The $\rightarrow$ function is defined by a set of pattern-matching rules:

$$f(x) = (x + 1) \rightarrow f(x) = (x + 1)$$

$$(2 + f(13)) \rightarrow (2 + (13 + 1))$$

due to the pattern rule

$$\ldots<id>1(<id>2) = <expr>1 \ldots \rightarrow \ldots<id>1(<id>2) = <expr>1 \ldots$$

$$\ldots<id>1(<expr>2) \ldots \rightarrow \ldots<expr>3 \ldots$$

where $<expr>3$ is $<expr>1$ with $<id>2$ replaced by $<expr>2$
Pattern-Matching Rules for Evaluation

• Rule 1

... <id>_1(<id>_2) = <expr>_1 ... \rightarrow ... <id>_1(<id>_2) = <expr>_1 ...

... <id>_1(<expr>_2) ... ...

... <expr>_3 ...

where <expr>_3 is <expr>_1 with <id>_2 replaced by <expr>_2

• Rules 2 - \infty

... (0 + 0) ... \rightarrow ... 0 ...

... (0 - 0) ... \rightarrow ... 0 ...

... (1 + 0) ... \rightarrow ... 1 ...

... (1 - 0) ... \rightarrow ... 1 ...

... (0 + 1) ... \rightarrow ... 1 ...

... (0 - 1) ... \rightarrow ... -1 ...

... (2 + 0) ... \rightarrow ... 2 ...

... (2 - 0) ... \rightarrow ... 2 ...

\textit{etc.} \hspace{1cm} \textit{etc.}
Homework

• Some evaluations
• Some membership proofs
• See the web page for details
• Due next Tuesday, August 27, 11:59 PM
Where is This Going?

Next time:

• Shift syntax slightly to match that of Scheme
• Add new clauses to the expression grammar
• Add new evaluation rules

Current goal is to learn Scheme, but we'll use algebraic techniques all semester