CS3520
Programming Language Concepts

Instructor: **Matthew Flatt**

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):

<prog>	::=	<defn>* <expr>
<defn>	::=	<id>(<id>) = <expr>
<expr>	::=	(<expr> + <expr>)
	::=	(<expr> - <expr>)
	::=	<id>(<expr>)
	::=	<id> **|** <num>
<id>	::=	a variable name: **f**, **x**, **y**, **z**, ...
<num>	::=	a number: 1, 42, 17, ...

- Each ***meta-variable***, such as <prog>, defines a set

Using a BNF Grammar

<id>	::=	a variable name: **f**, **x**, **y**, **z**, ...
<num>	::=	a number: 1, 42, 17, ...

- The set <id> is the set of all variable names
- The set <num> is the set of all numbers

- To make an example member of <num>, simply pick an element from the set

1 ∈ <num>

198 ∈ <num>

Using a BNF Grammar

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

- The set <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

<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

    7 ∈ <num>

  - combine the examples with literal text

    7 ∈ <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>   7 ∈ <expr>

  - combine the examples with literal text

    f(7) ∈ <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

<prog> ::= <defn>* <expr>
<defn> ::= <id>(<id>) = <expr>

f(**x**) = (**x** + 1) ∈ <defn>

- To make a <prog> pick some number of <defn>s

(**x** + **y**) ∈ <prog>

f(**x**) = (**x** + 1)
g(**y**) = **f**((**y** - 2)) ∈ <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:

$$\frac{\textit{sub-claim to prove} \qquad \ldots \qquad \textit{sub-claim to prove}}{\textit{claim to prove}}$$

- Immediate membership claims serve as leaves on the tree:

7 ∈ <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:

$$\frac{\textit{sub-claim to prove} \qquad \ldots \qquad \textit{sub-claim to prove}}{\textit{claim to prove}}$$

- Immediate membership claims serve as leaves on the tree:

f ∈ <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:

$$\frac{\textit{sub-claim to prove} \qquad \ldots \qquad \textit{sub-claim to prove}}{\textit{claim to prove}}$$

- Other membership claims generate branches in the tree:

$$\frac{7 \in \texttt{<num>}}{7 \in \texttt{<expr>}}$$

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:

$$\frac{\textit{sub-claim to prove} \qquad \ldots \qquad \textit{sub-claim to prove}}{\textit{claim to prove}}$$

- Other membership claims generate branches in the tree:

$$\frac{\mathbf{f} \in \text{<id>} \qquad \dfrac{7 \in \text{<num>}}{7 \in \text{<expr>}}}{\mathbf{f}(7) \in \text{<expr>}}$$

The proof tree's shape is driven entirely by the grammar

Demonstrating Set Membership: Example

f(7) ∈ <expr>

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

- Two meta-variables on the left means two sub-trees:
  - One for **f** ∈ <id>
  - One for 7 ∈ <expr>

Demonstrating Set Membership: Example

$$\frac{\mathbf{f} \in \text{<id>} \qquad\qquad 7 \in \text{<expr>}}{\mathbf{f}(7) \in \text{<expr>}}$$

<id> ::= a variable name: **f**, **x**, **y**, **z**, ...

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

- **f** ∈ <id> is immediate
- 7 ∈ <expr> has one meta-variable, so one subtree

Demonstrating Set Membership: Example

$$\frac{\mathbf{f} \in \langle\text{id}\rangle \qquad \dfrac{\boxed{7 \in \langle\text{num}\rangle}}{7 \in \langle\text{expr}\rangle}}{\mathbf{f}(7) \in \langle\text{expr}\rangle}$$

<num> ::= a number: 1, 42, 17, ...

- 7 ∈ <num> is immediate, so the proof is complete

Demonstrating Set Membership: Another Example

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

<prog>   ::=   <defn>* <expr>

- Three meta-variables (after expanding *) means three sub-trees:
  - One for **f**(**x**) = (**x** + 1) ∈ <defn>
  - One for **g**(**y**) = **f**((**y** - 2)) ∈ <defn>
  - One for **g**(7) ∈ <expr>

Demonstrating Set Membership: Example 2

$$\frac{\mathbf{f}(\mathbf{x}) = (\mathbf{x} + 1) \in \text{<defn>} \qquad \mathbf{g}(\mathbf{y}) = \mathbf{f}((\mathbf{y} - 2)) \in \text{<defn>} \qquad \mathbf{g}(7) \in \text{<expr>}}{\begin{array}{l} \mathbf{f}(\mathbf{x}) = (\mathbf{x} + 1) \\ \mathbf{g}(\mathbf{y}) = \mathbf{f}((\mathbf{y} - 2)) \quad \in \text{<prog>} \\ \mathbf{g}(7) \end{array}}$$

- 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) ∈ <defn>

<defn> ::= <id>(<id>) = <expr>

- Three meta-variables, three sub-trees

Demonstrating Set Membership: Example 2

f ∈ <id>   **x** ∈ <id>   (**x** + 1) ∈ <expr>

---

f(**x**) = (**x** + 1) ∈ <defn>

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

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

Demonstrating Set Membership: Example 2

Final tree:

$$\frac{\mathbf{f} \in <\mathrm{id}> \qquad \mathbf{x} \in <\mathrm{id}> \qquad \dfrac{\dfrac{\mathbf{x} \in <\mathrm{id}>}{\mathbf{x} \in <\mathrm{expr}>} \qquad \dfrac{1 \in <\mathrm{num}>}{1 \in <\mathrm{expr}>}}{(\mathbf{x} + 1) \in <\mathrm{expr}>}}{\mathbf{f}(\mathbf{x}) = (\mathbf{x} + 1) \in <\mathrm{defn}>}$$

- This was just one of three sub-trees for the original ∈ <prog> 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:

$$(2 + (7 - 4)) \quad \rightarrow \quad (2 + 3)$$

Evaluation Function

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

f(**x**) = (**x** + 1)	$\rightarrow$	**f**(**x**) = (**x** + 1)
(2 + (7 - 4))		(2 + 3)

Evaluation Function

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

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))

Evaluation Function

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

$\mathbf{f}(\mathbf{x}) = (\mathbf{x} + 1)$ $(2 + (7 - 4))$	$\rightarrow$	$\mathbf{f}(\mathbf{x}) = (\mathbf{x} + 1)$ $(2 + 3)$
$\mathbf{f}(\mathbf{x}) = (\mathbf{x} + 1)$ $(2 + 3)$	$\rightarrow$	$\mathbf{f}(\mathbf{x}) = (\mathbf{x} + 1)$ 5

Evaluation Function

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

$$\begin{array}{lcl} \mathbf{f}(\mathbf{x}) = (\mathbf{x} + 1) & \rightarrow & \mathbf{f}(\mathbf{x}) = (\mathbf{x} + 1) \\ (2 + (7 - 4)) & & (2 + 3) \end{array}$$

due to the pattern rule

$$\textbf{...}\ (7 - 4)\ \textbf{...} \ \rightarrow \ \textbf{...}\ 3\ \textbf{...}$$

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))		(2 + (13 + 1))

due to the pattern rule

... <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>$_2$

Pattern-Matching Rules for Evaluation

- **Rule 1**

$$\ldots \langle id\rangle_1(\langle id\rangle_2) = \langle expr\rangle_1 \ldots \qquad \rightarrow \qquad \ldots \langle id\rangle_1(\langle id\rangle_2) = \langle expr\rangle_1 \ldots$$
$$\ldots \langle id\rangle_1(\langle expr\rangle_2) \ldots \qquad\qquad\qquad \ldots \langle expr\rangle_3 \ldots$$

where $\langle expr\rangle_3$ is $\langle expr\rangle_1$ with $\langle id\rangle_2$ replaced by $\langle expr\rangle_2$

- **Rules 2 - ∞**

... (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 ...
etc.	*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