CS3520
Programming Languages Concepts

Instructor: **Matthew Flatt**

Programming Languages 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:
  - $(1+2) = 3$
  - $\mathbf{f}(17) = (17+3) = 20$ if $\mathbf{f}(\mathbf{x}) = (\mathbf{x}+3)$

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

- The set <id> is the set of all variable names
- The set <n> is the set of all numbers
- To make an example member of <n>, simply pick an element from the set

$1 \in$ <n>

$198 \in$ <n>

Using a BNF Grammar

<expr>	::=	(<expr> + <expr>)
	::=	(<expr> − <expr>)
	::=	<id>(<expr>)
	::=	<id> | <n>

- The set <expr> is defined in terms of other sets

Using a BNF Grammar

<expr> ::= (<expr> + <expr>)
::= (<expr> – <expr>)
::= <id>(<expr>)
::= <id> | <n>

- 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> | <n>

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

    7 ∈ <n>

  - combine the examples with literal text

    7 ∈ <expr>

Using a BNF Grammar

<expr> ::= (<expr> + <expr>)
::= (<expr> – <expr>)
::= <id>(<expr>)
::= <id> | <n>

- 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> | <n>

- 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 ∈ <n>

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:

$$\frac{7 \in \langle n \rangle}{7 \in \langle expr \rangle}$$

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:

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

- Other membership claims generate branches in the tree:

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

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> | <n>

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

Demonstrating Set Membership: Example

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

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

<expr>	::=	(<expr> + <expr>)
	::=	(<expr> – <expr>)
	::=	<id>(<expr>)
	::=	<id> | <n>

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

Demonstrating Set Membership: Example

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

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

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

Demonstrating Set Membership: Another Example

$$\left.\begin{array}{l}\mathbf{f}(\mathbf{x}) = (\mathbf{x} + 1) \\ \mathbf{g}(\mathbf{y}) = \mathbf{f}((\mathbf{y} - 2)) \\ \mathbf{g}(7)\end{array}\right. \in \langle prog\rangle$$

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

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

Demonstrating Set Membership: Example 2

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

- Each sub-tree can be proved separately
- We'll prove only the first sub-tree for now

Demonstrating Set Membership: Example 2

$$\mathbf{f}(\mathbf{x}) = (\mathbf{x} + 1) \in \langle defn\rangle$$

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

- Three meta-variables, three sub-trees

Demonstrating Set Membership: Example 2

$$\frac{\mathbf{f} \in \langle id\rangle \qquad \mathbf{x} \in \langle id\rangle \qquad (\mathbf{x} + 1) \in \langle expr\rangle}{\mathbf{f}(\mathbf{x}) = (\mathbf{x} + 1) \in \langle defn\rangle}$$

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

```
<expr> ::= (<expr> + <expr>)   ⬅
       ::= (<expr> – <expr>)
       ::= <id>(<expr>)
       ::= <id>  |  <n>
```

Demonstrating Set Membership: Example 2

Final tree:

$$\dfrac{\mathbf{f} \in \langle\text{id}\rangle \qquad \mathbf{x} \in \langle\text{id}\rangle \qquad \dfrac{\dfrac{\mathbf{x} \in \langle\text{id}\rangle}{\mathbf{x} \in \langle\text{expr}\rangle} \qquad \dfrac{1 \in \langle\text{n}\rangle}{1 \in \langle\text{expr}\rangle}}{(\mathbf{x} + 1) \in \langle\text{expr}\rangle}}{\mathbf{f}(\mathbf{x}) = (\mathbf{x} + 1) \in \langle\text{defn}\rangle}$$

- 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:
  - $(1 + 2) = 3$
  - $\mathbf{f}(17) = (17 + 3) = 20$ if $\mathbf{f}(\mathbf{x}) = (\mathbf{x} + 3)$

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:

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

Evaluation Function

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

$$\begin{array}{lcl} \mathbf{f}(\mathbf{x}) = (\mathbf{x}+1) & & \mathbf{f}(\mathbf{x}) = (\mathbf{x}+1) \\ \mathbf{g}(\mathbf{y}) = (\mathbf{y}-1) & \rightarrow & \mathbf{g}(\mathbf{y}) = (\mathbf{y}-1) \\ \mathbf{h}(\mathbf{z}) = \mathbf{f}(\mathbf{z}) & & \mathbf{h}(\mathbf{z}) = \mathbf{f}(\mathbf{z}) \\ (2+\mathbf{f}(13)) & & (2+(13+1)) \end{array}$$

Evaluation Function

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

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

$$\begin{array}{lcl} \mathbf{f}(\mathbf{x}) = (\mathbf{x}+1) & \rightarrow & \mathbf{f}(\mathbf{x}) = (\mathbf{x}+1) \\ (2+3) & & 5 \end{array}$$

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:

$$\begin{array}{lcl} \mathbf{f}(\mathbf{x}) = (\mathbf{x}+1) & \rightarrow & \mathbf{f}(\mathbf{x}) = (\mathbf{x}+1) \\ (2+\mathbf{f}(13)) & & (2+(13+1)) \end{array}$$

due to the pattern rule

$$\begin{array}{lcl} \textbf{...}\ \langle\text{id}\rangle_1(\langle\text{id}\rangle_2) = \langle\text{expr}\rangle_1\ \textbf{...} & \rightarrow & \textbf{...}\ \langle\text{id}\rangle_1(\langle\text{id}\rangle_2) = \langle\text{expr}\rangle_1\ \textbf{...} \\ \textbf{...}\ \langle\text{id}\rangle_1(\langle\text{expr}\rangle_2)\ \textbf{...} & & \textbf{...}\ \langle\text{expr}\rangle_3\ \textbf{...} \end{array}$$

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

Pattern-Matching Rules for Evaluation

- **Rule 1**

$$\begin{array}{lcl} \dots\ \langle\text{id}\rangle_1(\langle\text{id}\rangle_2) = \langle\text{expr}\rangle_1\ \dots & & \dots\ \langle\text{id}\rangle_1(\langle\text{id}\rangle_2) = \langle\text{expr}\rangle_1\ \dots \\ & \rightarrow & \\ \dots\ \langle\text{id}\rangle_1(\langle\text{expr}\rangle_2)\ \dots & & \dots\ \langle\text{expr}\rangle_3\ \dots \end{array}$$

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

- **Rules 2 -** ∞

$$\begin{array}{lcl@{\qquad\qquad}lcl} \dots\ (0+0)\ \dots & \rightarrow & \dots\ 0\ \dots & \dots\ (0-0)\ \dots & \rightarrow & \dots\ 0\ \dots \\ \dots\ (1+0)\ \dots & \rightarrow & \dots\ 1\ \dots & \dots\ (1-0)\ \dots & \rightarrow & \dots\ 1\ \dots \\ \dots\ (0+1)\ \dots & \rightarrow & \dots\ 1\ \dots & \dots\ (0-1)\ \dots & \rightarrow & \dots\ -1\ \dots \\ \dots\ (2+0)\ \dots & \rightarrow & \dots\ 2\ \dots & \dots\ (2-0)\ \dots & \rightarrow & \dots\ 2\ \dots \\ & \textit{etc.} & & & \textit{etc.} & \end{array}$$

Homework

- Some evaluations
- Some membership proofs
- See the web page for details
- Due next Tuesday, August 28, 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