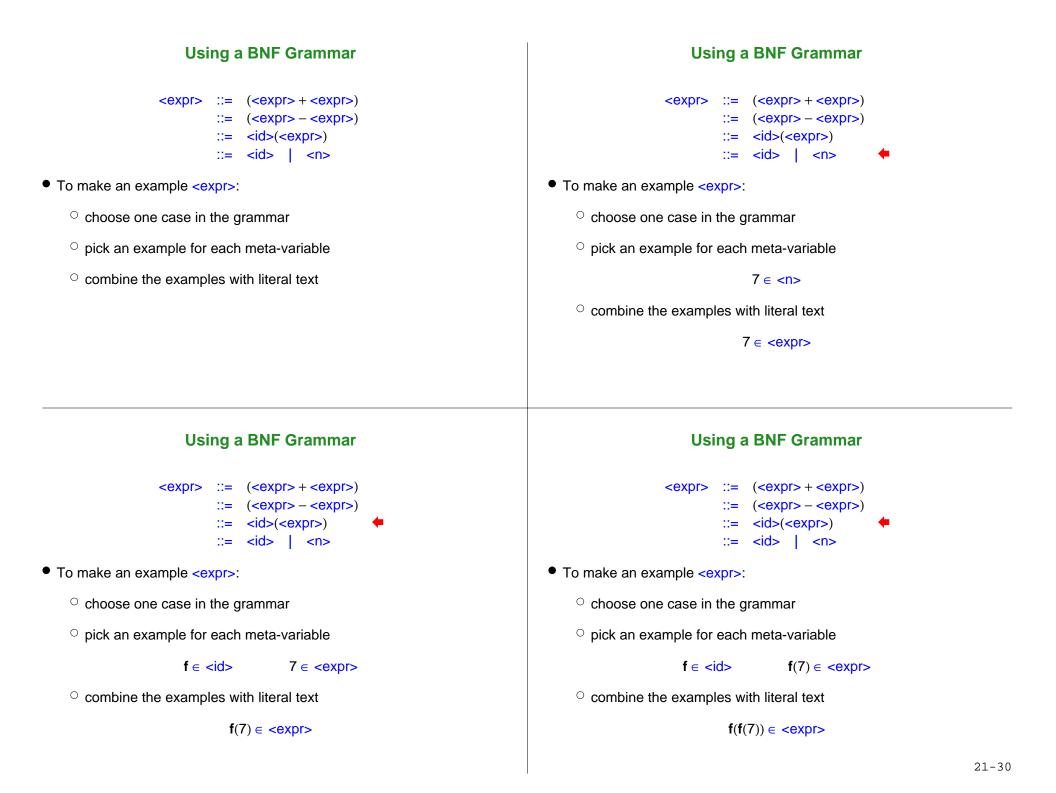
CS3520 Programming Languages Concepts Instructor: Matthew Flatt	<ul> <li>This course teaches concepts in two ways:</li> <li>By implementing interpreters <ul> <li>new concept =&gt; extend interpreter</li> </ul> </li> <li>By using Scheme <ul> <li>we assume that you <i>don't</i> already know Scheme</li> </ul> </li> </ul>
Course Details	Bootstrapping Problem
http://www.cs.utah.edu/classes/cs3520/	<ul> <li>We'll learn about languages by writing interpreters in Scheme</li> <li>We'll learn about Scheme by writing an interpreter in Scheme set theory</li> <li>More specifically, we'll define Scheme as an extension of algebra <i>Algebra is a programming language?</i></li> </ul>

Programming Languages Concepts

Algebra as a Programming Language	<b>A Grammar for Algebra Programs</b> The grammar in <b>BNF</b> (Backus-Naur Form; <i>EoPL</i> sec 1.1.2):	
<ul> <li>Algebra has a grammar:</li> </ul>		
$^{\circ}$ (1 + 2) is a legal expression	<prog> ::= <defn>* <expr></expr></defn></prog>	
$^{\circ}$ (1 + +) is not a legal expression	<defn> ::= <id>(<id>) = <expr> <expr> ::= (<expr> + <expr>) ::= (<expr> - <expr>)</expr></expr></expr></expr></expr></expr></id></id></defn>	
<ul> <li>Algebra has rules for evaluation:</li> </ul>	::= <id>(<expr>) ::= <id>   <n></n></id></expr></id>	
<sup>○</sup> (1 + 2) = 3	<id> ::= a variable name: f, x, y, z,</id>	
○ $f(17) = (17 + 3) = 20$ if $f(x) = (x + 3)$	<n> ::= a number: 1, 42, 17,</n>	
	Each <i>meta-variable</i> , such as <prog>, defines a set</prog>	
Using a BNF Grammar	Using a BNF Grammar	
<id> ::= a variable name: <b>f</b>, <b>x</b>, <b>y</b>, <b>z</b>, <n> ::= a number: 1, 42, 17,</n></id>	<expr> ::= (<expr> + <expr>) ::= (<expr> - <expr>)</expr></expr></expr></expr></expr>	
• The set <id> is the set of all variable names</id>	::= <id>(<expr>) ::= <id>(<expr>) ::= <id>   <n></n></id></expr></id></expr></id>	
The set <n> is the set of all numbers</n>	<ul> <li>The set <expr> is defined in terms of other sets</expr></li> </ul>	
<ul> <li>To make an example member of <n>, simply pick an element from the set</n></li> </ul>		
1 ∈ <n></n>		
198 ∈ <i>&lt;</i> n>		



#### **Using a BNF Grammar Demonstrating Set Membership** • We can run the element-generation process in reverse to prove that ::= <defn>\* <expr> <prog> $::= \langle id \rangle \langle \langle id \rangle \rangle = \langle expr \rangle$ some item is a member of a set <defn> $\mathbf{f}(\mathbf{x}) = (\mathbf{x} + 1) \in \langle \text{defn} \rangle$ • Such proofs have a standard tree format: sub-claim to prove sub-claim to prove • To make a <prog> pick some number of <defn>s claim to prove $(\mathbf{x} + \mathbf{y}) \in \langle \mathsf{prog} \rangle$ • Immediate membership claims serve as leaves on the tree: 7 ∈ <n> f(x) = (x + 1) $\mathbf{g}(\mathbf{y}) = \mathbf{f}((\mathbf{y} - 2)) \in \langle \mathsf{prog} \rangle$ **g**(7) **Demonstrating Set Membership Demonstrating Set Membership** • We can run the element-generation process in reverse to *prove* that • We can run the element-generation process in reverse to prove that some item is a member of a set some item is a member of a set • Such proofs have a standard tree format: • Such proofs have a standard tree format: sub-claim to prove sub-claim to prove sub-claim to prove sub-claim to prove ... ... claim to prove claim to prove • Immediate membership claims serve as leaves on the tree: • Other membership claims generate branches in the tree: $\mathbf{f} \in \langle \mathbf{id} \rangle$ 7 ∈ <n> $7 \in \langle expr \rangle$

#### **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:

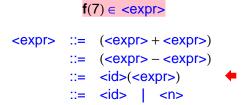
sub-claim to prove ... sub-claim to prove claim to prove

• Other membership claims generate branches in the tree:

 $f \in \langle id \rangle \qquad \begin{array}{c} 7 \in \langle n \rangle \\ \hline 7 \in \langle expr \rangle \\ \hline f(7) \in \langle expr \rangle \end{array}$ 

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

#### **Demonstrating Set Membership: Example**



• Two meta-variables on the left means two sub-trees:

- One for  $f \in \langle id \rangle$
- $^{\circ}$  One for 7  $\in$  <expr>

#### **Demonstrating Set Membership: Example**

 $f \in \langle id \rangle \qquad 7 \in \langle expr \rangle$   $f(7) \in \langle expr \rangle$   $\langle id \rangle \qquad ::= a \text{ variable name: } \mathbf{f}, \mathbf{x}, \mathbf{y}, \mathbf{z}, ...$   $\langle expr \rangle \qquad ::= (\langle expr \rangle + \langle expr \rangle)$   $i:= (\langle expr \rangle - \langle expr \rangle)$   $i:= \langle id \rangle (\langle expr \rangle)$   $i:= \langle id \rangle \mid \langle en \rangle$ 

•  $f \in \langle id \rangle$  is immediate

•  $7 \in \langle expr \rangle$  has one meta-variable, so one subtree

#### **Demonstrating Set Membership: Example**

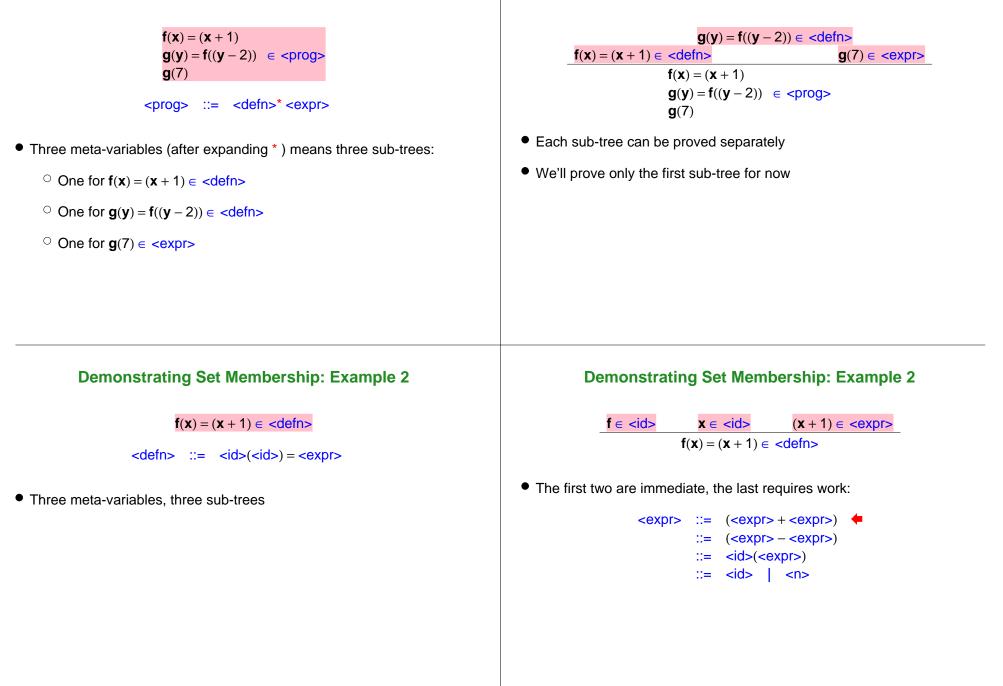
$$f \in \langle id \rangle \qquad \begin{array}{c} 7 \in \langle n \rangle \\ \hline 7 \in \langle expr \rangle \\ \hline f(7) \in \langle expr \rangle \end{array}$$

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

•  $7 \in \langle n \rangle$  is immediate, so the proof is complete

## **Demonstrating Set Membership: Another Example**

# Demonstrating Set Membership: Example 2



Final tree:

		<b>x</b> ∈ <id></id>	1 ∈ <n></n>
		<b>x</b> ∈ <expr></expr>	1 ∈ <expr></expr>
<b>f</b> ∈ <id></id>	<b>x</b> ∈ <id></id>	( <b>x</b> + 1) ∈ <expr></expr>	
	<b>f</b> ( <b>x</b> ) =	( <b>x</b> + 1) ∈ <defn></defn>	

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

## Algebra as a Programming Language

• Algebra has a grammar:

 $^{\circ}$  (1 + 2) is a legal expression

- $^{\circ}$  (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:

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

### **Evaluation Function**

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

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

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

$$\begin{array}{ccc} f({\bf x}) = ({\bf x}+1) \\ (2+(7-4)) \end{array} & \to & \begin{array}{c} f({\bf x}) = ({\bf x}+1) \\ (2+3) \end{array} \\ \end{array} \\ \begin{array}{c} f({\bf x}) = ({\bf x}+1) \\ (2+3) \end{array} & \to & \begin{array}{c} f({\bf x}) = ({\bf x}+1) \\ 5 \end{array} \\ \end{array}$$

## **Evaluation Function**

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

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

due to the pattern rule

$$\dots (7-4) \dots \rightarrow \dots 3 \dots$$

#### **Evaluation Function**

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

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

due to the pattern rule

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

## Pattern-Matching Rules for Evaluation

#### Homework

#### • Rule 1

$$\begin{array}{cccc} \dots & < \mathrm{id}_1 (< \mathrm{id}_2) = < \mathrm{expr}_1 \dots & & \dots & < \mathrm{id}_1 (< \mathrm{id}_2) = < \mathrm{expr}_1 \dots & \\ \dots & < \mathrm{id}_1 (< \mathrm{expr}_2) \dots & & \dots & < \mathrm{expr}_3 \dots \end{array}$$

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

#### • Rules 2 - $\infty$

- 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