CS 5510
Programming Language Concepts

Fall 2009

Instructor: **Matthew Flatt**

Course Details

`http://www.eng.utah.edu/~cs5510/`

Programming Language Concepts

This course teaches concepts in two ways:

Programming Language Concepts

This course teaches concepts in two ways:

By implementing **interpreters**

- new concept $\Rightarrow$ new interpreter

Programming Language Concepts

This course teaches concepts in two ways:

By implementing **interpreters**

- new concept $\Rightarrow$ new interpreter

By using **Scheme** and variants

- we don't assume that you already know Scheme

Interpreters

An ***interpreter*** takes a program and produces a result

- DrScheme
- x86 processor
- desktop calculator
- **bash**
- Algebra student

Interpreters

An ***interpreter*** takes a program and produces a result

- DrScheme
- x86 processor
- desktop calculator
- **bash**
- Algebra student

A ***compiler*** takes a program and produces another program

In the terminology of programming languages, someone who translates Chinese to English is a compiler!

Interpreters

An ***interpreter*** takes a program and produces a result

- DrScheme
- x86 processor
- desktop calculator
- **bash**
- Algebra student

A ***compiler*** takes a program and produces another program

In the terminology of programming languages, someone who translates Chinese to English is a compiler!

So, what's a ***program***?

A Grammar for Algebra Programs

A grammar of Algebra in ***BNF*** (Backus-Naur Form):

⟨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, ...

A Grammar for Algebra Programs

A grammar of Algebra in ***BNF*** (Backus-Naur Form):

⟨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

$\langle$id$\rangle$	::=	a variable name: **f**, **x**, **y**, **z**, ...
$\langle$num$\rangle$	::=	a number: 1, 42, 17, ...

The set $\langle$id$\rangle$ is the set of all variable names

The set $\langle$num$\rangle$ is the set of all numbers

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

Using a BNF Grammar

$\langle$id$\rangle$	::=	a variable name: **f**, **x**, **y**, **z**, ...
$\langle$num$\rangle$	::=	a number: 1, 42, 17, ...

The set $\langle$id$\rangle$ is the set of all variable names

The set $\langle$num$\rangle$ is the set of all numbers

To make an example member of $\langle$num$\rangle$, simply pick an element from the set

$$1 \in \langle\text{num}\rangle$$

$$198 \in \langle\text{num}\rangle$$

Using a BNF Grammar

$$
\begin{array}{lcl}
\langle \text{expr} \rangle & ::= & (\langle \text{expr} \rangle + \langle \text{expr} \rangle) \\
 & | & (\langle \text{expr} \rangle - \langle \text{expr} \rangle) \\
 & | & \langle \text{id} \rangle(\langle \text{expr} \rangle) \\
 & | & \langle \text{id} \rangle \\
 & | & \langle \text{num} \rangle
\end{array}
$$

The set $\langle \text{expr} \rangle$ is defined in terms of other sets

Using a BNF Grammar

$$
\begin{array}{lcl}
\langle \text{expr} \rangle & ::= & (\langle \text{expr} \rangle + \langle \text{expr} \rangle) \\
 & | & (\langle \text{expr} \rangle - \langle \text{expr} \rangle) \\
 & | & \langle \text{id} \rangle(\langle \text{expr} \rangle) \\
 & | & \langle \text{id} \rangle \\
 & | & \langle \text{num} \rangle
\end{array}
$$

To make an example $\langle \text{expr} \rangle$:

- 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
- combine the examples with literal text

Using a BNF Grammar

$$
\begin{array}{lcl}
\langle \text{expr} \rangle & ::= & (\langle \text{expr} \rangle + \langle \text{expr} \rangle) \\
 & | & (\langle \text{expr} \rangle - \langle \text{expr} \rangle) \\
 & | & \langle \text{id} \rangle(\langle \text{expr} \rangle) \\
 & | & \langle \text{id} \rangle \\
 & | & \langle \text{num} \rangle
\end{array}
$$

To make an example $\langle \text{expr} \rangle$:

- 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

Using a BNF Grammar

$$
\begin{array}{lcl}
\langle\text{expr}\rangle & ::= & (\langle\text{expr}\rangle + \langle\text{expr}\rangle) \\
 & | & (\langle\text{expr}\rangle - \langle\text{expr}\rangle) \\
 & | & \langle\text{id}\rangle(\langle\text{expr}\rangle) \\
 & | & \langle\text{id}\rangle \\
 & | & \langle\text{num}\rangle
\end{array}
$$

To make an example $\langle\text{expr}\rangle$:

- 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

$$
\begin{array}{lcl}
\langle \text{expr} \rangle & ::= & (\langle \text{expr} \rangle + \langle \text{expr} \rangle) \\
& | & (\langle \text{expr} \rangle - \langle \text{expr} \rangle) \\
& | & \langle \text{id} \rangle(\langle \text{expr} \rangle) \\
& | & \langle \text{id} \rangle \\
& | & \langle \text{num} \rangle
\end{array}
$$

To make an example $\langle \text{expr} \rangle$:

- choose one case in the grammar
- pick an example for each meta-variable
- combine the examples with literal text

Using a BNF Grammar

$$
\begin{array}{lcl}
\langle \text{expr} \rangle & ::= & (\langle \text{expr} \rangle + \langle \text{expr} \rangle) \\
 & | & (\langle \text{expr} \rangle - \langle \text{expr} \rangle) \\
 & | & \langle \text{id} \rangle(\langle \text{expr} \rangle) \\
 & | & \langle \text{id} \rangle \\
 & | & \langle \text{num} \rangle
\end{array}
$$

To make an example ⟨expr⟩:

- choose one case in the grammar
- pick an example for each meta-variable

$$\mathbf{f} \in \langle \text{id} \rangle$$

- combine the examples with literal text

Using a BNF Grammar

$$
\begin{array}{lll}
\langle \text{expr} \rangle & ::= & (\langle \text{expr} \rangle + \langle \text{expr} \rangle) \\
 & | & (\langle \text{expr} \rangle - \langle \text{expr} \rangle) \\
 & | & \langle \text{id} \rangle(\langle \text{expr} \rangle) \\
 & | & \langle \text{id} \rangle \\
 & | & \langle \text{num} \rangle
\end{array}
$$

To make an example $\langle \text{expr} \rangle$:

- choose one case in the grammar
- pick an example for each meta-variable

  $\textbf{f} \in \langle \text{id} \rangle \qquad\qquad 7 \in \langle \text{expr} \rangle$

- combine the examples with literal text

Using a BNF Grammar

$$
\begin{array}{lcl}
\langle \text{expr} \rangle & ::= & (\langle \text{expr} \rangle + \langle \text{expr} \rangle) \\
 & | & (\langle \text{expr} \rangle - \langle \text{expr} \rangle) \\
 & | & \langle \text{id} \rangle(\langle \text{expr} \rangle) \\
 & | & \langle \text{id} \rangle \\
 & | & \langle \text{num} \rangle
\end{array}
$$

To make an example $\langle \text{expr} \rangle$:

- choose one case in the grammar
- pick an example for each meta-variable

$$\mathbf{f} \in \langle \text{id} \rangle \qquad\qquad 7 \in \langle \text{expr} \rangle$$

- combine the examples with literal text

$$\mathbf{f}(7) \in \langle \text{expr} \rangle$$

Using a BNF Grammar

$$
\begin{array}{lll}
\langle\text{expr}\rangle & ::= & (\langle\text{expr}\rangle + \langle\text{expr}\rangle) \\
 & | & (\langle\text{expr}\rangle - \langle\text{expr}\rangle) \\
 & | & \langle\text{id}\rangle(\langle\text{expr}\rangle) \\
 & | & \langle\text{id}\rangle \\
 & | & \langle\text{num}\rangle
\end{array}
$$

To make an example ⟨expr⟩:

- choose one case in the grammar
- pick an example for each meta-variable

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

- combine the examples with literal text

Using a BNF Grammar

$$
\begin{array}{lcl}
\langle\text{expr}\rangle & ::= & (\langle\text{expr}\rangle + \langle\text{expr}\rangle) \\
 & | & (\langle\text{expr}\rangle - \langle\text{expr}\rangle) \\
 & | & \langle\text{id}\rangle(\langle\text{expr}\rangle) \\
 & | & \langle\text{id}\rangle \\
 & | & \langle\text{num}\rangle
\end{array}
$$

To make an example $\langle\text{expr}\rangle$:

- choose one case in the grammar
- pick an example for each meta-variable

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

- combine the examples with literal text

$$\mathbf{f}(\mathbf{f}(7)) \in \langle\text{expr}\rangle$$

Using a BNF Grammar

$$\langle \text{prog} \rangle \quad ::= \quad \langle \text{defn} \rangle^{*} \ \langle \text{expr} \rangle$$
$$\langle \text{defn} \rangle \quad ::= \quad \langle \text{id} \rangle(\langle \text{id} \rangle) = \langle \text{expr} \rangle$$

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

Using a BNF Grammar

$$\langle \text{prog} \rangle \quad ::= \quad \langle \text{defn} \rangle^{*} \ \langle \text{expr} \rangle$$
$$\langle \text{defn} \rangle \quad ::= \quad \langle \text{id} \rangle (\langle \text{id} \rangle) = \langle \text{expr} \rangle$$

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

To make a ⟨prog⟩ pick some number of ⟨defn⟩s

$$(\mathbf{x} + \mathbf{y}) \in \langle \text{prog} \rangle$$

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

Programming Language

A ***programming language*** is defined by

- a grammar for programs
- rules for evaluating any program to produce a result

Programming Language

A ***programming language*** is defined by

- a grammar for programs
- rules for evaluating any program to produce a result

For example, Algebra evaluation is defined in terms of evaluation steps:

$$(2 + (7 - 4)) \quad \rightarrow \quad (2 + 3) \quad \rightarrow \quad 5$$

Programming Language

A ***programming language*** is defined by

- a grammar for programs
- rules for evaluating any program to produce a result

For example, Algebra evaluation is defined in terms of evaluation steps:

$$\mathbf{f}(\mathbf{x}) = (\mathbf{x} + 1)$$

$$\mathbf{f}(10) \quad \rightarrow \quad (10 + 1) \quad \rightarrow \quad 11$$

Evaluation

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

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

due to the pattern rule

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

Evaluation

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

$$\mathbf{f}(\mathbf{x}) = (\mathbf{x} + 1)$$

$$\mathbf{f}(10) \quad \rightarrow \quad (10 + 1)$$

due to the pattern rule

$$\textbf{...}\ \langle \text{id} \rangle_{\mathbf{1}}(\langle \text{id} \rangle_{\mathbf{2}}) = \langle \text{expr} \rangle_{\mathbf{1}}\ \textbf{...}$$

$$\textbf{...}\ \langle \text{id} \rangle_{\mathbf{1}}(\langle \text{expr} \rangle_{\mathbf{2}})\ \textbf{...} \quad \rightarrow \quad \textbf{...}\ \langle \text{expr} \rangle_{\mathbf{3}}\ \textbf{...}$$

where $\langle \text{expr} \rangle_{\mathbf{3}}$ is $\langle \text{expr} \rangle_{\mathbf{1}}$ with $\langle \text{id} \rangle_{\mathbf{2}}$ replaced by $\langle \text{expr} \rangle_{\mathbf{2}}$

Pattern-Matching Rules for Evaluation

- **Rule 1**

$$\textbf{...}\ \langle \text{id} \rangle_{\mathbf{1}}(\langle \text{id} \rangle_{\mathbf{2}}) = \langle \text{expr} \rangle_{\mathbf{1}}\ \textbf{...}$$

$$\textbf{...}\ \langle \text{id} \rangle_{\mathbf{1}}(\langle \text{expr} \rangle_{\mathbf{2}})\ \textbf{...} \qquad \rightarrow \qquad \textbf{...}\ \langle \text{expr} \rangle_{\mathbf{3}}\ \textbf{...}$$

where $\langle \text{expr} \rangle_{\mathbf{3}}$ is $\langle \text{expr} \rangle_{\mathbf{1}}$ with $\langle \text{id} \rangle_{\mathbf{2}}$ replaced by $\langle \text{expr} \rangle_{\mathbf{2}}$

Pattern-Matching Rules for Evaluation

- **Rule 1**

... $\langle id \rangle_{\mathbf{1}}(\langle id \rangle_{\mathbf{2}}) = \langle expr \rangle_{\mathbf{1}}$ **...**

... $\langle id \rangle_{\mathbf{1}}(\langle expr \rangle_{\mathbf{2}})$ **...** $\rightarrow$ **...** $\langle expr \rangle_{\mathbf{3}}$ **...**

where $\langle expr \rangle_{\mathbf{3}}$ is $\langle expr \rangle_{\mathbf{1}}$ with $\langle id \rangle_{\mathbf{2}}$ replaced by $\langle expr \rangle_{\mathbf{2}}$

- **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.*

Pattern-Matching Rules for Evaluation

- **Rule 1**

  ... $\langle \text{id} \rangle_{\mathbf{1}}(\langle \text{id} \rangle_{\mathbf{2}}) = \langle \text{expr} \rangle_{\mathbf{1}}$ **...**

  ... $\langle \text{id} \rangle_{\mathbf{1}}(\langle \text{expr} \rangle_{\mathbf{2}})$ **...** $\rightarrow$ **...** $\langle \text{expr} \rangle_{\mathbf{3}}$ **...**

  where $\langle \text{expr} \rangle_{\mathbf{3}}$ is $\langle \text{expr} \rangle_{\mathbf{1}}$ with $\langle \text{id} \rangle_{\mathbf{2}}$ replaced by $\langle \text{expr} \rangle_{\mathbf{2}}$

- **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.*

When the interpreter is a program instead of an Algebra student, the rules look a little different

HW 1

On the course web page:

Write an interpreter for a small language of string manipulations

Assignment is due **Monday**

Your code may be featured in class on Monday