CS 5510
Programming Language Concepts

Fall 2007

Instructor: **Matthew Flatt**

Course Details

http://www.cs.utah.edu/classes/cs5510/

Programming Language Concepts

This course teaches concepts in two ways:

By implementing **interpreters**

- new concept $\Rightarrow$ new interpreter

By using **Scheme** and variants

- new concept $\Rightarrow$ new variant of Scheme
- we don't assume that you already know Scheme

Interpreters

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

Examples:

- 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, ...

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 $\in$ <num>

198 $\in$ <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)

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

$$\ldots\ (7 - 4)\ \ldots \quad \rightarrow \quad \ldots\ 3\ \ldots$$

Evaluation

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

f(**x**) = (**x** + 1)

f(10) $\rightarrow$ (10 + 1)

due to the pattern rule

... <id>$_{\mathbf{1}}$(<id>$_{\mathbf{2}}$) = <expr>$_{\mathbf{1}}$ **...**

... <id>$_{\mathbf{1}}$(<expr>$_{\mathbf{2}}$) **...** $\rightarrow$ **...** <expr>$_{\mathbf{3}}$ **...**

where <expr>$_{\mathbf{3}}$ is <expr>$_{\mathbf{1}}$ with <id>$_{\mathbf{2}}$ replaced by <expr>$_{\mathbf{2}}$

Pattern-Matching Rules for Evaluation

- **Rule 1**

... <id>$_1$(<id>$_2$) = <expr>$_1$ **...**

... <id>$_1$(<expr>$_2$) **...** $\rightarrow$ **...** <expr>$_3$ **...**

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

- **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