EECS 395
Programming Languages

Winter 2010

Instructor: **Robby Findler**

Course Details

http://www.eecs.northwestern.edu/~robby/courses/395-2010-winter/

(or google "findler" and follow the links)

Programming Language Concepts

This course teaches concepts in two ways:

By implementing **interpreters**

- new concept ⇒ new interpreter

By using **Scheme** and variants

- we don't assume that you already know Scheme

Interpreters vs Compilers

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

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

Interpreters vs Compilers

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

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

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

- DrScheme
- x86 processor
- **`gcc`**
- **`javac`**

Interpreters vs Compilers

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

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

Good for understanding program behavior, easy to implement

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

- DrScheme
- x86 processor
- **`gcc`**
- **`javac`**

Good for speed, more complex (come back next quarter)

Interpreters vs Compilers

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

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

Good for understanding program behavior, easy to implement

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

- DrScheme
- x86 processor
- **`gcc`**
- **`javac`**

Good for speed, more complex (come back next quarter)

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

⟨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

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

⟨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

$$2 \in \langle \text{num} \rangle$$

$$298 \in \langle \text{num} \rangle$$

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
- 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

$$7 \in \langle num \rangle$$

- 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

$$7 \in \langle num \rangle$$

- combine the examples with literal text

$$7 \in \langle expr \rangle$$

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

f ∈ ⟨id⟩

- 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

  f ∈ ⟨id⟩ 7 ∈ ⟨expr⟩

- 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

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

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⟩

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

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)) → (2 + 3) → 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:

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

f(10) → (10 + 1) → 11

Evaluation

- Evaluation → is defined by a set of pattern-matching rules:

(2 + (7 - 4)) → (2 + 3)

due to the pattern rule

... (7 - 4) ... → ... 3 ...

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

... $\langle id \rangle_1$ ($\langle id \rangle_2$) = $\langle expr \rangle_1$ ...

... $\langle id \rangle_1$ ($\langle expr \rangle_2$) ... $\rightarrow$ ... $\langle expr \rangle_3$ ...

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

Rules for Evaluation

- **Rule 1 -** one pattern

  ... ⟨id⟩ $_1$ **(** ⟨id⟩ $_2$ **) =** ⟨expr⟩ $_1$ **...**

  ... ⟨id⟩ $_1$ **(** ⟨expr⟩ $_2$ **) ...** → **...** ⟨expr⟩ $_3$ **...**

  where ⟨expr⟩ $_3$ is ⟨expr⟩ $_1$ with ⟨id⟩ $_2$ replaced by ⟨expr⟩ $_2$

Rules for Evaluation

- **Rule 1 -** one pattern

  ... ⟨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**

- **Rules 2 -** ∞ special cases

  ... (0 + 0) ... → ... 0 ...
  ... (1 + 0) ... → ... 1 ...
  ... (2 + 0) ... → ... 2 ...
  etc.

  ... (0 - 0) ... → ... 0 ...
  ... (1 - 0) ... → ... 1 ...
  ... (2 - 0) ... → ... 2 ...
  etc.

Rules for Evaluation

- **Rule 1 -** one pattern

  ... ⟨id⟩ $_1$**(** ⟨id⟩ $_2$**) =** ⟨expr⟩ $_1$ **...**

  ... ⟨id⟩ $_1$**(** ⟨expr⟩ $_2$**) ...** → **...** ⟨expr⟩ $_3$ **...**

  where ⟨expr⟩ $_3$ is ⟨expr⟩ $_1$ with ⟨id⟩ $_2$ replaced by ⟨expr⟩ $_2$

- **Rules 2 -** ∞ special cases

  ... (0 + 0) ... → ... 0 ...
  ... (1 + 0) ... → ... 1 ...
  ... (2 + 0) ... → ... 2 ...
  etc.

  ... (0 - 0) ... → ... 0 ...
  ... (1 - 0) ... → ... 1 ...
  ... (2 - 0) ... → ... 2 ...
  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 **Friday**

Your code may be featured in class on Monday