

CS 5510

Programming Languages

Fall 2010

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 **Racket** and variants

- we don't assume that you already know Racket

Interpreters

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

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

Interpreters

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

- DrRacket
- 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*, not an *interpreter*.

PLAI Racket

See `quick-ref.rkt`

Bootstrapping

- We'd like to define languages via interpreters
- We'll implement interpreters in PLAI Racket

How do we define PLAI Racket?

Bootstrapping

- We'd like to define languages via interpreters
- We'll implement interpreters in PLAI Racket

How do we define PLAI Racket?

Alternate way of defining languages: **sets and relations**

A Grammar for Algebra Programs

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

$\langle \text{prog} \rangle ::= \langle \text{defn} \rangle^* \langle \text{expr} \rangle$
 $\langle \text{defn} \rangle ::= \langle \text{id} \rangle (\langle \text{id} \rangle) = \langle \text{expr} \rangle$
 $\langle \text{expr} \rangle ::= (\langle \text{expr} \rangle + \langle \text{expr} \rangle)$
 $\quad | (\langle \text{expr} \rangle - \langle \text{expr} \rangle)$
 $\quad | \langle \text{id} \rangle (\langle \text{expr} \rangle)$
 $\quad | \langle \text{id} \rangle$
 $\quad | \langle \text{num} \rangle$
 $\langle \text{id} \rangle ::=$ a variable name: **f**, **x**, **y**, **z**, ...
 $\langle \text{num} \rangle ::=$ a number: **1**, **42**, **17**, ...

A Grammar for Algebra Programs

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

$\langle \text{prog} \rangle ::= \langle \text{defn} \rangle^* \langle \text{expr} \rangle$
 $\langle \text{defn} \rangle ::= \langle \text{id} \rangle (\langle \text{id} \rangle) = \langle \text{expr} \rangle$
 $\langle \text{expr} \rangle ::= (\langle \text{expr} \rangle + \langle \text{expr} \rangle)$
 $\quad | (\langle \text{expr} \rangle - \langle \text{expr} \rangle)$
 $\quad | \langle \text{id} \rangle (\langle \text{expr} \rangle)$
 $\quad | \langle \text{id} \rangle$
 $\quad | \langle \text{num} \rangle$
 $\langle \text{id} \rangle ::=$ a variable name: **f**, **x**, **y**, **z**, ...
 $\langle \text{num} \rangle ::=$ a number: **1**, **42**, **17**, ...

Each **meta-variable**, such as $\langle \text{prog} \rangle$, defines a set

Using a BNF Grammar

$\langle \text{id} \rangle ::=$ a variable name: **f**, **x**, **y**, **z**, ...

$\langle \text{num} \rangle ::=$ a number: **1**, **42**, **17**, ...

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

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

Using a BNF Grammar

$\langle \text{id} \rangle ::=$ a variable name: **f**, **x**, **y**, **z**, ...

$\langle \text{num} \rangle ::=$ a number: **1**, **42**, **17**, ...

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

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

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

Using a BNF Grammar

$\langle \text{id} \rangle ::=$ a variable name: **f**, **x**, **y**, **z**, ...

$\langle \text{num} \rangle ::=$ a number: **1**, **42**, **17**, ...

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

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

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

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

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

Using a BNF Grammar

```
 $\langle \text{expr} \rangle ::= (\langle \text{expr} \rangle + \langle \text{expr} \rangle)$   
|  $(\langle \text{expr} \rangle - \langle \text{expr} \rangle)$   
|  $\langle \text{id} \rangle (\langle \text{expr} \rangle)$   
|  $\langle \text{id} \rangle$   
|  $\langle \text{num} \rangle$ 
```

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

Using a BNF Grammar

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

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 $\langle \text{expr} \rangle$:

- 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 $\langle \text{expr} \rangle$:

- 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

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


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

```
⟨expr⟩ ::= (⟨expr⟩ + ⟨expr⟩)
          | (⟨expr⟩ - ⟨expr⟩)
          | ⟨id⟩ (⟨expr⟩)
          | ⟨id⟩
          | ⟨num⟩
```



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

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


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

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

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

- combine the examples with literal text

Using a BNF Grammar

```
 $\langle \text{expr} \rangle ::= (\langle \text{expr} \rangle + \langle \text{expr} \rangle)$   
          |  $(\langle \text{expr} \rangle - \langle \text{expr} \rangle)$   
          |  $\langle \text{id} \rangle (\langle \text{expr} \rangle)$  ←  
          |  $\langle \text{id} \rangle$   
          |  $\langle \text{num} \rangle$ 
```

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

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

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

- combine the examples with literal text

Using a BNF Grammar

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


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

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

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

- combine the examples with literal text

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

Using a BNF Grammar

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


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

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

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

- combine the examples with literal text

Using a BNF Grammar

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


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

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

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

- combine the examples with literal text

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

Using a BNF Grammar

$\langle \text{prog} \rangle ::= \langle \text{defn} \rangle^* \langle \text{expr} \rangle$

$\langle \text{defn} \rangle ::= \langle \text{id} \rangle (\langle \text{id} \rangle) = \langle \text{expr} \rangle$

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

Using a BNF Grammar

$\langle \text{prog} \rangle ::= \langle \text{defn} \rangle^* \langle \text{expr} \rangle$

$\langle \text{defn} \rangle ::= \langle \text{id} \rangle (\langle \text{id} \rangle) = \langle \text{expr} \rangle$

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

To make a $\langle \text{prog} \rangle$ pick some number of $\langle \text{defn} \rangle$ s

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

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

$\mathbf{g(y)} = \mathbf{f((y - 2))} \in \langle \text{prog} \rangle$

$\mathbf{g(7)}$

Programming Languages via Sets and Relations

A programming language can be defined by

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

Programming Languages via Sets and Relations

A programming language can be 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:

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

Programming Languages via Sets and Relations

A programming language can be 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:

$$f(x) = (x + 1)$$

$$f(10) \quad \rightarrow \quad (10 + 1) \quad \rightarrow \quad 11$$

Evaluation

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

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

due to the pattern rule

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

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

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

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

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

- **Rule 1**

... $\langle \text{id} \rangle_1 (\langle \text{id} \rangle_2) = \langle \text{expr} \rangle_1$...

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

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**

... $\langle \text{id} \rangle_1 (\langle \text{id} \rangle_2) = \langle \text{expr} \rangle_1$...

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

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 - ∞**

... (0 + 0) ...	\rightarrow	... 0 (0 - 0) ...	\rightarrow	... 0 ...
... (1 + 0) ...	\rightarrow	... 1 (1 - 0) ...	\rightarrow	... 1 ...
... (0 + 1) ...	\rightarrow	... 1 (0 - 1) ...	\rightarrow	... -1 ...

etc. etc.

Pattern-Matching Rules for Evaluation

- **Rule 1**

... $\langle \text{id} \rangle_1 (\langle \text{id} \rangle_2) = \langle \text{expr} \rangle_1$...

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

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 - ∞**

...	$(0 + 0)$...	\rightarrow	...	0	$(0 - 0)$...	\rightarrow	...	0	...
...	$(1 + 0)$...	\rightarrow	...	1	$(1 - 0)$...	\rightarrow	...	1	...
...	$(0 + 1)$...	\rightarrow	...	1	$(0 - 1)$...	\rightarrow	...	-1	...

etc. etc.

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

PLAI Racket Values

```
⟨val⟩ ::= ⟨num⟩  
      | ⟨bool⟩  
      | ⟨string⟩  
      | ⟨symbol⟩  
      | ⟨prim⟩  
      | (list ⟨val⟩*)  
      | (void)  
      | (⟨variant-id⟩ ⟨val⟩*)  
      | (lambda (⟨id⟩*) ⟨expr⟩)
```

PLAI Racket Expressions

`<expr>` ::= `<val>`
| `<id>`
| `(<expr> <expr>*)`
| `(or <expr>*)`
| `(and <expr>*)`
| `(if <expr> <expr> <expr>)`
| `(cond [<expr> <expr>]*)`
| `(local [<defn>*] <expr>)`
| `(type-case <id> <expr> <clause>)`
| `(set! <id> <expr>)`

`<clause>` ::= `[<variant-id> (<id>*) <expr>]`

PLAI Racket Programs

$\langle \text{prog} \rangle ::= \langle \text{form} \rangle^*$

$\langle \text{form} \rangle ::= \langle \text{expr} \rangle$
 $\quad \quad \quad | \langle \text{defn} \rangle$

$\langle \text{defn} \rangle ::= (\text{define } \langle \text{id} \rangle \langle \text{expr} \rangle)$
 $\quad \quad \quad | (\text{define } (\langle \text{id} \rangle \langle \text{id} \rangle^*) \langle \text{expr} \rangle)$
 $\quad \quad \quad | (\text{define-type } \langle \text{id} \rangle \langle \text{variant} \rangle^*)$

$\langle \text{variant} \rangle ::= [\langle \text{id} \rangle (\langle \text{id} \rangle \langle \text{expr} \rangle)^*]$

PLAI Racket Expressions

```
<expr> ::= <val>  
| <id>  
| ( <expr> <expr>*)  
| (or <expr>*)  
| (and <expr>*)  
| (if <expr> <expr> <expr>)  
| (cond [ <expr> <expr>]*)  
| (local [ <defn>*] <expr>)  
| (type-case <id> <expr> <clause>)  
| (set! <id> <expr>)  
  
<clause> ::= [ <variant-id> ( <id>*) <expr> ]
```



PLAI Racket Evaluation

... **(define** `<id>`₁ `<val>`₁) ...

... `<id>`₁ ...

→

... `<val>`₁ ...

PLAI Racket Evaluation

... (define <id>_ <val>_) ...
... <id>_ ... → ... <val>_ ...

Example:

(define pi 3)
(+ pi 1) → (+ 3 1)

PLAI Racket Expressions

`<expr>` ::= `<val>`
| `<id>`
| `(<expr> <expr>*)` ←
| `(or <expr>*)`
| `(and <expr>*)`
| `(if <expr> <expr> <expr>)`
| `(cond [<expr> <expr>]*)`
| `(local [<defn>*] <expr>)`
| `(type-case <id> <expr> <clause>)`
| `(set! <id> <expr>)`

`<clause>` ::= `[<variant-id> (<id>*) <expr>]`

PLAI Racket Evaluation

... (⟨prim⟩ ⟨val⟩₁ ... ⟨val⟩_n) ... → ... ⟨val⟩₀ ...

based on some “memorized” rule for ⟨prim⟩

PLAI Racket Evaluation

... (`<prim>` `<val>1` ... `<val>n`) ... \rightarrow ... `<val>0` ...

based on some “memorized” rule for `<prim>`

Examples:

(+ 1 1) \rightarrow 2

• (string-append "a" "b") \rightarrow "ab"

PLAI Racket Evaluation

... **(define** (**<id>₁** **<id>₂**) **<expr>₁**) ...

... (**<id>₁** **<val>₂**) ... \rightarrow ... **<expr>₃** ...

where **<expr>₃** is **<expr>₁** with **<id>₂** replaced by
<val>₂

PLAI Racket Evaluation

... **(define** (**<id>₁** **<id>₂**) **<expr>₁**) ...

... (**<id>₁** **<val>₂**) ... \rightarrow ... **<expr>₃** ...

where **<expr>₃** is **<expr>₁** with **<id>₂** replaced by **<val>₂**

Example:

(define (f x) (+ x 1))
(f 2) \rightarrow **(+ 2 1)** \rightarrow **3**

PLAI Racket Expressions

```
<expr> ::= <val>
| <id>
| (<expr> <expr>*)
| (or <expr>*)
| (and <expr>*)
| (if <expr> <expr> <expr>)
| (cond [<expr> <expr>]*)
| (local [<defn>*] <expr>)
| (type-case <id> <expr> <clause>)
| (set! <id> <expr>)
```



```
<clause> ::= [ <variant-id> (<id>*) <expr> ]
```

PLAI Racket Evaluation

(or #f <expr>₁ ... <expr>_n) → **(or <expr>₁ ... <expr>_n)**

(or #t <expr>₁ ... <expr>_n) → **#t**

(or) → **#f**

(and #f <expr>₁ ... <expr>_n) → **#f**

(and #t <expr>₁ ... <expr>_n) → **(and <expr>₁ ... <expr>_n)**

(and) → **#t**

PLAI Racket Evaluation

`(or #f <expr>1 ... <expr>n)` → `(or <expr>1 ... <expr>n)`

`(or #t <expr>1 ... <expr>n)` → `#t`

`(or)` → `#f`

`(and #f <expr>1 ... <expr>n)` → `#f`

`(and #t <expr>1 ... <expr>n)` → `(and <expr>1 ... <expr>n)`

`(and)` → `#t`

Example:

`(or (= 1 1) (= 1 2))` → `(or #t (= 1 2))` → `#t`

PLAI Racket Evaluation

`(or #f <expr>1 ... <expr>n)` \rightarrow `(or <expr>1 ... <expr>n)`

`(or #t <expr>1 ... <expr>n)` \rightarrow `#t`

`(or)` \rightarrow `#f`

`(and #f <expr>1 ... <expr>n)` \rightarrow `#f`

`(and #t <expr>1 ... <expr>n)` \rightarrow `(and <expr>1 ... <expr>n)`

`(and)` \rightarrow `#t`

Example:

`(and (= 1 1) (= 1 2))` \rightarrow `(and #t (= 1 2))`
 \rightarrow `(and (= 1 2))` \rightarrow `(and #f)` \rightarrow `#f`

PLAI Racket Evaluation

(or #f <expr>₁ ... <expr>_n) → **(or <expr>₁ ... <expr>_n)**

(or <val> <expr>₁ ... <expr>_n) → **<val>**

(or) → **#f**

(and #f <expr>₁ ... <expr>_n) → **#f**

(and <val> <expr>₁ ... <expr>_n) → **(and <expr>₁ ... <expr>_n)**

(and <val>) → **<val>**

(and) → **#t**

in this notation, earlier rules take precedence

PLAI Racket Expressions

`<expr>` ::= `<val>`
| `<id>`
| `(<expr> <expr>*)`
| `(or <expr>*)`
| `(and <expr>*)`
| `(if <expr> <expr> <expr>)` ←
| `(cond [<expr> <expr>]*)`
| `(local [<defn>*] <expr>)`
| `(type-case <id> <expr> <clause>)`
| `(set! <id> <expr>)`

`<clause>` ::= `[<variant-id> (<id>*) <expr>]`

PLAI Racket Evaluation

`(if #f <expr>1 <expr>2) → <expr>2`

`(if <val> <expr>1 <expr>2) → <expr>1`

PLAI Racket Evaluation

`(if #f <expr>1 <expr>2) → <expr>2`

`(if <val> <expr>1 <expr>2) → <expr>1`

Examples:

`(if (= 1 1) 10 20) → (if #t 10 20) → 10`

`(if (= 1 2) 10 20) → (if #f 10 20) → 20`

PLAI Racket Expressions

$\langle \text{expr} \rangle ::= \langle \text{val} \rangle$
| $\langle \text{id} \rangle$
| $(\langle \text{expr} \rangle \langle \text{expr} \rangle^*)$
| $(\text{or } \langle \text{expr} \rangle^*)$
| $(\text{and } \langle \text{expr} \rangle^*)$
| $(\text{if } \langle \text{expr} \rangle \langle \text{expr} \rangle \langle \text{expr} \rangle)$
| $(\text{cond } [\langle \text{expr} \rangle \langle \text{expr} \rangle]^*)$ ←
| $(\text{local } [\langle \text{defn} \rangle^*] \langle \text{expr} \rangle)$
| $(\text{type-case } \langle \text{id} \rangle \langle \text{expr} \rangle \langle \text{clause} \rangle)$
| $(\text{set! } \langle \text{id} \rangle \langle \text{expr} \rangle)$

$\langle \text{clause} \rangle ::= [\langle \text{variant-id} \rangle (\langle \text{id} \rangle^*) \langle \text{expr} \rangle]$

PLAI Racket Evaluation

(cond → **(cond**
[#f <expr>_{1a}] [<expr>_{2q} <expr>_{2a}] ...
[<expr>_{2q} <expr>_{2a}] ... [<expr>_{nq} <expr>_{na}])
[<expr>_{nq} <expr>_{na}])

(cond → <expr>_{1a}
[<val> <expr>_{1a}]
[<expr>_{2q} <expr>_{2a}] ...
[<expr>_{nq} <expr>_{na}])

PLAI Racket Evaluation

(cond \rightarrow **(cond**
 [**#f** $\langle \text{expr} \rangle_{1a}$] [$\langle \text{expr} \rangle_{2q}$ $\langle \text{expr} \rangle_{2a}$] ...
 [$\langle \text{expr} \rangle_{2q}$ $\langle \text{expr} \rangle_{2a}$] ... [$\langle \text{expr} \rangle_{nq}$ $\langle \text{expr} \rangle_{na}$])
 [$\langle \text{expr} \rangle_{nq}$ $\langle \text{expr} \rangle_{na}$])

(cond \rightarrow $\langle \text{expr} \rangle_{1a}$
 [$\langle \text{val} \rangle$ $\langle \text{expr} \rangle_{1a}$]
 [$\langle \text{expr} \rangle_{2q}$ $\langle \text{expr} \rangle_{2a}$] ...
 [$\langle \text{expr} \rangle_{nq}$ $\langle \text{expr} \rangle_{na}$])

Example:

(cond [**#t **0**] [**else** **10**]) \rightarrow **0****

PLAI Racket Evaluation

(cond \rightarrow **(cond**
 [#f $\langle \text{expr} \rangle_{1a}$] [$\langle \text{expr} \rangle_{2q}$ $\langle \text{expr} \rangle_{2a}$] ...
 [$\langle \text{expr} \rangle_{2q}$ $\langle \text{expr} \rangle_{2a}$] ... [$\langle \text{expr} \rangle_{nq}$ $\langle \text{expr} \rangle_{na}$])
 [$\langle \text{expr} \rangle_{nq}$ $\langle \text{expr} \rangle_{na}$])

(cond \rightarrow $\langle \text{expr} \rangle_{1a}$
 [$\langle \text{val} \rangle$ $\langle \text{expr} \rangle_{1a}$]
 [$\langle \text{expr} \rangle_{2q}$ $\langle \text{expr} \rangle_{2a}$] ...
 [$\langle \text{expr} \rangle_{nq}$ $\langle \text{expr} \rangle_{na}$])

Example:

(cond [#f 0] [else 10]) \rightarrow **(cond [else 10])**

PLAI Racket Evaluation

`(cond [else <expr>]) → <expr>`

`(cond) → (void)`

PLAI Racket Expressions

`<expr>` ::= `<val>`
| `<id>`
| `(<expr> <expr>*)`
| `(or <expr>*)`
| `(and <expr>*)`
| `(if <expr> <expr> <expr>)`
| `(cond [<expr> <expr>]*)`
| `(local [<defn>*] <expr>)` ←
| `(type-case <id> <expr> <clause>)`
| `(set! <id> <expr>)`

`<clause>` ::= `[<variant-id> (<id>*) <expr>]`

PLAI Racket Evaluation

... **(local [(define <id>_i <expr>_i)]**
 <expr>_o) ...

→ **(define <id>_{ia} <expr>_{ia})**
 ... **<expr>_{oa}** ...

where **<id>_{ia}** is unused and **<expr>_{ia}** is the same as **<expr>_i** with
<id>_i replaced by **<id>_{ia}**

PLAI Racket Evaluation

```
... (local [(define <id>i <expr>i)]  
      <expr>0) ...
```

```
→ (define <id>ia <expr>ia)  
   ... <expr>0a ...
```

where $\langle \text{id} \rangle_{ia}$ is unused and $\langle \text{expr} \rangle_{ia}$ is the same as $\langle \text{expr} \rangle_i$ with $\langle \text{id} \rangle_i$ replaced by $\langle \text{id} \rangle_{ia}$

Example:

```
(define x 1) → (define x 1)  
(local [(define x 2)] (define x42 2)  
  (+ x 3))      (+ x42 3)
```

PLAI Racket Expressions

`<expr>` ::= `<val>`
| `<id>`
| `(<expr> <expr>*)`
| `(or <expr>*)`
| `(and <expr>*)`
| `(if <expr> <expr> <expr>)`
| `(cond [<expr> <expr>]*)`
| `(local [<defn>*] <expr>)`
| `(type-case <id> <expr> <clause>)` ←
| `(set! <id> <expr>)`

`<clause>` ::= `[<variant-id> (<id>*) <expr>]`

PLAI Racket Evaluation

(type-case $\langle id \rangle_0$ ($\langle variant-id \rangle_1$ $\langle val \rangle$) \rightarrow $\langle expr \rangle_{1a}$
 [$\langle variant-id \rangle_1$ ($\langle id \rangle$) $\langle expr \rangle_1$]
 $\langle clause \rangle_1$... $\langle clause \rangle_n$)

where $\langle expr \rangle_{1a}$ is the same as $\langle expr \rangle_1$ with $\langle id \rangle$ replaced by $\langle val \rangle$

PLAI Racket Evaluation

```
(type-case <id>0 ( <variant-id>1 <val> ) → <expr>1a  
  [ <variant-id>1 ( <id> ) <expr>1 ]  
  <clause>1 ... <clause>n)
```

where $\langle \text{expr} \rangle_{1a}$ is the same as $\langle \text{expr} \rangle_1$ with $\langle \text{id} \rangle$ replaced by $\langle \text{val} \rangle$

Example:

```
(define-type Light  
  [bulb (watts number?)]  
  [led (volts number?)])  
  
(type-case Light (bulb 100) → (< 100 75)  
  [bulb (w) (< w 75)]  
  [led (v) (zero? v)])
```

PLAI Racket Evaluation

(type-case $\langle id \rangle_0$ ($\langle variant-id \rangle_1$ $\langle val \rangle$) \rightarrow $\langle expr \rangle_{1a}$
 [$\langle variant-id \rangle_1$ ($\langle id \rangle$) $\langle expr \rangle_1$]
 $\langle clause \rangle_1$... $\langle clause \rangle_n$)

where $\langle expr \rangle_{1a}$ is the same as $\langle expr \rangle_1$ with $\langle id \rangle$ replaced by $\langle val \rangle$

Example:

```
(define-type Light  
  [bulb (watts number?)]  
  [led (volts number?)])  
  
(type-case Light (led 1) →  
  [bulb (w) (< w 75)]  
  [led (v) (zero? v)])
```

PLAI Racket Evaluation

(type-case $\langle id \rangle_0$ ($\langle variant-id \rangle_1$ $\langle val \rangle$)

[$\langle variant-id \rangle_2$ ($\langle id \rangle$) $\langle expr \rangle_2$]

$\langle clause \rangle_1$... $\langle clause \rangle_n$)

→ **(type-case** $\langle id \rangle_0$ ($\langle variant-id \rangle_1$ $\langle val \rangle$)
 $\langle clause \rangle_1$... $\langle clause \rangle_n$)

Example:

```
(define-type Light
  [bulb (watts number?)]
  [led (volts number?)])
```

```
(type-case Light (led 1) → (type-case Light (led 1)
  [bulb (w) (< w 75)]      [led (v) (zero? v)])
  [led (v) (zero? v)])
```

PLAI Racket Evaluation

`(type-case <id>0 (<variant-id>1 <val>) → <expr>
[else <expr>])`

PLAI Racket Expressions

$\langle \text{expr} \rangle ::= \langle \text{val} \rangle$
| $\langle \text{id} \rangle$
| $(\langle \text{expr} \rangle \langle \text{expr} \rangle^*)$
| $(\text{or } \langle \text{expr} \rangle^*)$
| $(\text{and } \langle \text{expr} \rangle^*)$
| $(\text{if } \langle \text{expr} \rangle \langle \text{expr} \rangle \langle \text{expr} \rangle)$
| $(\text{cond } [\langle \text{expr} \rangle \langle \text{expr} \rangle]^*)$
| $(\text{local } [\langle \text{defn} \rangle^*] \langle \text{expr} \rangle)$
| $(\text{type-case } \langle \text{id} \rangle \langle \text{expr} \rangle \langle \text{clause} \rangle)$
| $(\text{set! } \langle \text{id} \rangle \langle \text{expr} \rangle)$ 

$\langle \text{clause} \rangle ::= [\langle \text{variant-id} \rangle (\langle \text{id} \rangle^*) \langle \text{expr} \rangle]$

PLAI Racket Evaluation

... **(define** $\langle id \rangle_1$ $\langle val \rangle_1$) ... \rightarrow ... **(define** $\langle id \rangle_1$ $\langle val \rangle_2$) ...
... **(set!** $\langle id \rangle_1$ $\langle val \rangle_2$) **(void)** ...

PLAI Racket Evaluation

... (define <id>₁ <val>₁) ... → ... (define <id>₁ <val>₂) ...
... (set! <id>₁ <val>₂) (void) ...

Example:

(define pi 3) → (define pi 4)
(set! pi 4) (void)

Homework 0

- Create handin account
- Racket warm-up exercises

Due Tuesday, August 31