CS 5510
Programming Languages

Fall 2011

Instructor: **Matthew Flatt**

Course Details

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

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

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?

Alternate way of defining languages: sets and relations

A Grammar for Algebra Programs

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

$$
\begin{array}{lcl}
\langle \text{prog} \rangle & ::= & \langle \text{defn} \rangle^{*} \ \langle \text{expr} \rangle \\
\langle \text{defn} \rangle & ::= & \langle \text{id} \rangle \, \mathtt{(} \langle \text{id} \rangle \mathtt{)} \ \mathtt{=} \ \langle \text{expr} \rangle \\
\langle \text{expr} \rangle & ::= & \mathtt{(} \langle \text{expr} \rangle \ \mathtt{+} \ \langle \text{expr} \rangle \mathtt{)} \\
 & | & \mathtt{(} \langle \text{expr} \rangle \ \mathtt{-} \ \langle \text{expr} \rangle \mathtt{)} \\
 & | & \langle \text{id} \rangle \, \mathtt{(} \langle \text{expr} \rangle \mathtt{)} \\
 & | & \langle \text{id} \rangle \\
 & | & \langle \text{num} \rangle \\
\langle \text{id} \rangle & ::= & \text{a variable name: } \mathtt{f},\ \mathtt{x},\ \mathtt{y},\ \mathtt{z},\ \ldots \\
\langle \text{num} \rangle & ::= & \text{a number: } \mathtt{1},\ \mathtt{42},\ \mathtt{17},\ \ldots
\end{array}
$$

Each ***meta-variable***, such as ⟨prog⟩, 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

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

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

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

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

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

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

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

Using a BNF Grammar

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

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

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

$$\mathbf{7} \in \langle \mathrm{num} \rangle$$

- combine the examples with literal text

$$\mathbf{7} \in \langle \mathrm{expr} \rangle$$

Using a BNF Grammar

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

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

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

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

- combine the examples with literal text

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

Using a BNF Grammar

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

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

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

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

- combine the examples with literal text

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

Using a BNF Grammar

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

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

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

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

```
f(x) = (x + 1)
g(y) = f((y - 2))
g(7)
```

$\in \langle\text{prog}\rangle$

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))   →   (2 + 3)   →   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)
```

$\texttt{f(10)} \longrightarrow \texttt{(10 + 1)} \longrightarrow \texttt{11}$

Evaluation

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

```
(2 + (7 - 4))    →    (2 + 3)
```

due to the pattern rule

```
... (7 - 4) ...    →    ... 3 ...
```

Evaluation

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

```
f(x) = (x + 1)
f(10)              →      (10 + 1)
```

due to the pattern rule

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

$$\ldots\ \langle\text{id}\rangle_1 \texttt{(}\langle\text{expr}\rangle_2\texttt{)}\ \ldots \quad \longrightarrow \quad \ldots\ \langle\text{expr}\rangle_3\ \ldots$$

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**

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

$$\ldots\ \langle\text{id}\rangle_1\,(\langle\text{expr}\rangle_2)\ \ldots \quad\longrightarrow\quad \ldots\ \langle\text{expr}\rangle_3\ \ldots$$

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)` …	→	… `0` …	… `(0 - 0)` …	→	… `0` …
… `(1 + 0)` …	→	… `1` …	… `(1 - 0)` …	→	… `1` …
… `(0 + 1)` …	→	… `1` …	… `(0 - 1)` …	→	… `-1` …
etc.			*etc.*		

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

PLAI Racket Values

$$
\begin{array}{lcl}
\langle \text{val} \rangle & ::= & \langle \text{num} \rangle \\
 & | & \langle \text{bool} \rangle \\
 & | & \langle \text{string} \rangle \\
 & | & \langle \text{symbol} \rangle \\
 & | & \langle \text{prim} \rangle \\
 & | & \texttt{(list } \langle \text{val} \rangle^{*}\texttt{)} \\
 & | & \texttt{(void)} \\
 & | & \texttt{(}\langle \text{variant-id} \rangle \ \langle \text{val} \rangle^{*}\texttt{)} \\
 & | & \texttt{(lambda (}\langle \text{id} \rangle^{*}\texttt{)} \ \langle \text{expr} \rangle\texttt{)}
\end{array}
$$

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

$$
\begin{array}{lcl}
\langle \text{prog} \rangle & ::= & \langle \text{form} \rangle^{*} \\
\\
\langle \text{form} \rangle & ::= & \langle \text{expr} \rangle \\
 & | & \langle \text{defn} \rangle \\
\\
\langle \text{defn} \rangle & ::= & \texttt{(define}\ \langle \text{id} \rangle\ \langle \text{expr} \rangle\texttt{)} \\
 & | & \texttt{(define (}\langle \text{id} \rangle\ \langle \text{id} \rangle^{*}\texttt{)}\ \langle \text{expr} \rangle\texttt{)} \\
 & | & \texttt{(define-type}\ \langle \text{id} \rangle\ \langle \text{variant} \rangle^{*}\texttt{)} \\
\\
\langle \text{variant} \rangle & ::= & \texttt{[}\langle \text{id} \rangle\ \texttt{(}\langle \text{id} \rangle\ \langle \text{expr} \rangle\texttt{)}^{*}\texttt{]}
\end{array}
$$

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** $\langle\text{id}\rangle_1$ $\langle\text{val}\rangle_1$**)** ...

... $\langle\text{id}\rangle_1$ ... $\longrightarrow$ ... $\langle\text{val}\rangle_1$ ...

Example:

```
(define pi 3)
(+ pi 1)          →   (+ 3 1)
```

PLAI Racket Expressions

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

PLAI Racket Evaluation

... $(\langle\text{prim}\rangle\ \langle\text{val}\rangle_1\ \ldots\ \langle\text{val}\rangle_n)$ ... $\rightarrow$ ... $\langle\text{val}\rangle_0$ ...

based on some "memorized" rule for $\langle\text{prim}\rangle$

Examples:

`(+ 1 1)` $\rightarrow$ `2`

- `(string-append "a" "b")` $\rightarrow$ `"ab"`

PLAI Racket Evaluation

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

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

where $\langle\text{expr}\rangle_3$ is $\langle\text{expr}\rangle_1$ with $\langle\text{id}\rangle_2$ replaced by $\langle\text{val}\rangle_2$

Example:

```
(define (f x) (+ x 1))
(f 2)                    →  (+ 2 1)  →  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` $\langle$expr$\rangle_1$ ... $\langle$expr$\rangle_n$`)` $\rightarrow$ `(or` $\langle$expr$\rangle_1$ ... $\langle$expr$\rangle_n$`)`

`(or #t` $\langle$expr$\rangle_1$ ... $\langle$expr$\rangle_n$`)` $\rightarrow$ `#t`

`(or)` $\rightarrow$ `#f`

`(and #f` $\langle$expr$\rangle_1$ ... $\langle$expr$\rangle_n$`)` $\rightarrow$ `#f`

`(and #t` $\langle$expr$\rangle_1$ ... $\langle$expr$\rangle_n$`)` $\rightarrow$ `(and` $\langle$expr$\rangle_1$ ... $\langle$expr$\rangle_n$`)`

`(and)` $\rightarrow$ `#t`

Example:

`(or (= 1 1) (= 1 2))` $\rightarrow$ `(or #t (= 1 2))` $\rightarrow$ `#t`

PLAI Racket Evaluation

$\texttt{(or \#f } \langle\text{expr}\rangle_1 \ldots \langle\text{expr}\rangle_n\texttt{)} \rightarrow \texttt{(or } \langle\text{expr}\rangle_1 \ldots \langle\text{expr}\rangle_n\texttt{)}$

$\texttt{(or \#t } \langle\text{expr}\rangle_1 \ldots \langle\text{expr}\rangle_n\texttt{)} \rightarrow \texttt{\#t}$

$\texttt{(or)} \rightarrow \texttt{\#f}$

$\texttt{(and \#f } \langle\text{expr}\rangle_1 \ldots \langle\text{expr}\rangle_n\texttt{)} \rightarrow \texttt{\#f}$

$\texttt{(and \#t } \langle\text{expr}\rangle_1 \ldots \langle\text{expr}\rangle_n\texttt{)} \rightarrow \texttt{(and } \langle\text{expr}\rangle_1 \ldots \langle\text{expr}\rangle_n\texttt{)}$

$\texttt{(and)} \rightarrow \texttt{\#t}$

Example:

$\texttt{(and (= 1 1) (= 1 2))} \rightarrow \texttt{(and \#t (= 1 2))}$
$\rightarrow \texttt{(and (= 1 2))} \rightarrow \texttt{(and \#f)} \rightarrow \texttt{\#f}$

PLAI Racket Evaluation

$\texttt{(or \#f}\ \langle\text{expr}\rangle_1 \ \ldots\ \langle\text{expr}\rangle_n\texttt{)} \rightarrow \texttt{(or}\ \langle\text{expr}\rangle_1 \ \ldots\ \langle\text{expr}\rangle_n\texttt{)}$

$\texttt{(or}\ \langle\text{val}\rangle\ \langle\text{expr}\rangle_1 \ \ldots\ \langle\text{expr}\rangle_n\texttt{)} \rightarrow \langle\text{val}\rangle$

$\texttt{(or)} \rightarrow \texttt{\#f}$

$\texttt{(and \#f}\ \langle\text{expr}\rangle_1 \ \ldots\ \langle\text{expr}\rangle_n\texttt{)} \rightarrow \texttt{\#f}$

$\texttt{(and}\ \langle\text{val}\rangle\ \langle\text{expr}\rangle_1 \ \ldots\ \langle\text{expr}\rangle_n\texttt{)} \rightarrow \texttt{(and}\ \langle\text{expr}\rangle_1 \ \ldots\ \langle\text{expr}\rangle_n\texttt{)}$

$\texttt{(and}\ \langle\text{val}\rangle\texttt{)} \rightarrow \langle\text{val}\rangle$

$\texttt{(and)} \rightarrow \texttt{\#t}$

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

$\texttt{(if \#f } \langle\text{expr}\rangle_1\ \langle\text{expr}\rangle_2\texttt{)} \longrightarrow \langle\text{expr}\rangle_2$

$\texttt{(if } \langle\text{val}\rangle\ \langle\text{expr}\rangle_1\ \langle\text{expr}\rangle_2\texttt{)} \longrightarrow \langle\text{expr}\rangle_1$

Examples:

`(if (= 1 1) 10 20)` $\longrightarrow$ `(if #t 10 20)` $\longrightarrow$ `10`

`(if (= 1 2) 10 20)` $\longrightarrow$ `(if #f 10 20)` $\longrightarrow$ `20`

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

$$\begin{array}{lcl}
\texttt{(cond} & \rightarrow & \texttt{(cond} \\
\quad \texttt{[\#f } \langle\text{expr}\rangle_{\mathbf{1a}}\texttt{]} & & \quad \texttt{[}\langle\text{expr}\rangle_{\mathbf{2q}}\ \langle\text{expr}\rangle_{\mathbf{2a}}\texttt{]}\ \ldots \\
\quad \texttt{[}\langle\text{expr}\rangle_{\mathbf{2q}}\ \langle\text{expr}\rangle_{\mathbf{2a}}\texttt{]}\ \ldots & & \quad \texttt{[}\langle\text{expr}\rangle_{\mathbf{nq}}\ \langle\text{expr}\rangle_{\mathbf{na}}\texttt{])} \\
\quad \texttt{[}\langle\text{expr}\rangle_{\mathbf{nq}}\ \langle\text{expr}\rangle_{\mathbf{na}}\texttt{])} & &
\end{array}$$

$$\begin{array}{lcl}
\texttt{(cond} & \rightarrow & \langle\text{expr}\rangle_{\mathbf{1a}} \\
\quad \texttt{[}\langle\text{val}\rangle\ \langle\text{expr}\rangle_{\mathbf{1a}}\texttt{]} & & \\
\quad \texttt{[}\langle\text{expr}\rangle_{\mathbf{2q}}\ \langle\text{expr}\rangle_{\mathbf{2a}}\texttt{]}\ \ldots & & \\
\quad \texttt{[}\langle\text{expr}\rangle_{\mathbf{nq}}\ \langle\text{expr}\rangle_{\mathbf{na}}\texttt{])} & &
\end{array}$$

Example:

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

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])
```

Example:

```
(cond [#f 0] [else 10]) → (cond [else 10])
```

PLAI Racket Evaluation

$\texttt{(cond [else } \langle\text{expr}\rangle\texttt{])} \longrightarrow \langle\text{expr}\rangle$

$\texttt{(cond)} \longrightarrow \texttt{(void)}$

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** $\langle\text{id}\rangle_{\mathbf{l}}$ $\langle\text{expr}\rangle_{\mathbf{l}}$**)]**
$\langle\text{expr}\rangle_{\mathbf{0}}$**)** ...

$\longrightarrow$ **(define** $\langle\text{id}\rangle_{\mathbf{la}}$ $\langle\text{expr}\rangle_{\mathbf{la}}$**)**
... $\langle\text{expr}\rangle_{\mathbf{0a}}$ ...

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

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

$\texttt{(type-case}\ \langle\text{id}\rangle_0\ \texttt{(}\langle\text{variant-id}\rangle_1\ \langle\text{val}\rangle\texttt{)}\quad \rightarrow \quad \langle\text{expr}\rangle_{1a}$

$\quad \texttt{[}\langle\text{variant-id}\rangle_1\ \texttt{(}\langle\text{id}\rangle\texttt{)}\ \langle\text{expr}\rangle_1\texttt{]}$

$\quad \langle\text{clause}\rangle_1\ \ldots\ \langle\text{clause}\rangle_n\texttt{)}$

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

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 (led 1) →
  [bulb (w) (< w 75)]
  [led (v) (zero? v)])
```

PLAI Racket Evaluation

```
(type-case ⟨id⟩_0 (⟨variant-id⟩_1 ⟨val⟩)
  [⟨variant-id⟩_2 (⟨id⟩) ⟨expr⟩_2]
  ⟨clause⟩_1 ... ⟨clause⟩_n)

        → (type-case ⟨id⟩_0 (⟨variant-id⟩_1 ⟨val⟩)
            ⟨clause⟩_1 ... ⟨clause⟩_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

$$\texttt{(type-case}\ \langle \text{id} \rangle_0\ \texttt{(}\langle \text{variant-id} \rangle_1\ \langle \text{val} \rangle\texttt{)}\quad \longrightarrow \quad \langle \text{expr} \rangle$$
$$\texttt{[else}\ \langle \text{expr} \rangle\texttt{])}$$

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` $\langle \text{id} \rangle_1$ $\langle \text{val} \rangle_1$`)` ... $\rightarrow$ ... `(define` $\langle \text{id} \rangle_1$ $\langle \text{val} \rangle_2$`)` ...

... `(set!` $\langle \text{id} \rangle_1$ $\langle \text{val} \rangle_2$`)` ... ... `(void)` ...

Example:

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

Homework 0

- Create handin account
- Racket warm-up exercises

Due Tuesday, August 30