

Standardizing RPAL AST's



Programming Languages
Lecture 6

Adeesha Wijayasiri

The Semantics of RPAL

We use the operational approach:

To specify the semantics of a language, we give an **OPERATIONAL PROCESS**.

PL constructs are “denoted” with functions whose behavior specify the meaning of the program.



The RPAL Operational Specification

- In the case of RPAL, the process is
 1. Scan and parse the program, transduce to a tree.
 2. Standardize the tree.

The RPAL Operational Specification (cont'd)

3. Flatten the tree into either:
 - a. A lambda expression.
 - b. Control Structures for a machine (more later)
- Important:
 - RPAL = lambda-calculus + syntactic sugar

Standardizing an RPAL AST

- "Desugar" the tree.
- Transform into a binary tree, whose internal nodes are exclusively 'gamma' and 'lambda'.

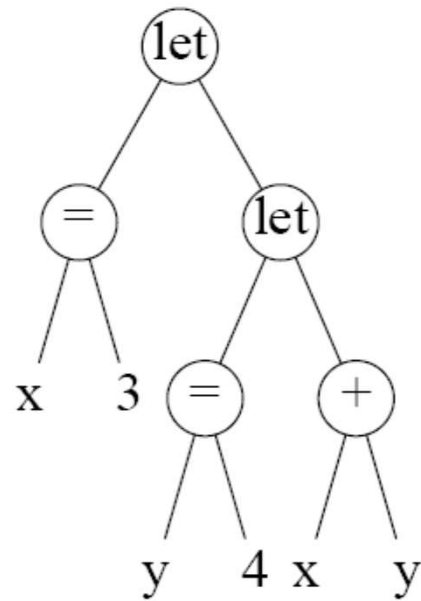
Standardizing 'let' and 'where'

- See example

| let | | gamma | | where |
|----------------|---------------|-----------------------------|--------------|----------------|
| $\frac{}{= P}$ | \Rightarrow | $\frac{}{\text{lambda } E}$ | \Leftarrow | $\frac{}{P =}$ |
| $\frac{}{X E}$ | | $\frac{}{X P}$ | | $\frac{}{X E}$ |

Standardizing 'let' and 'where':

let x=3 in let y=4 in x+y



Standardizing 'fcn_form'

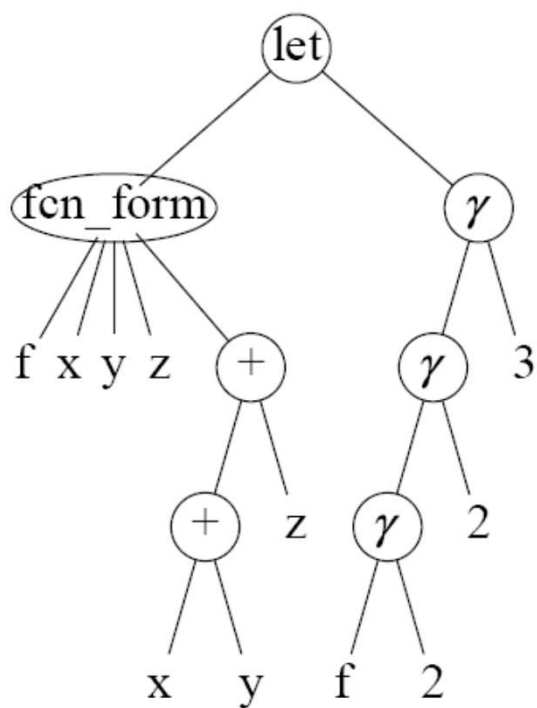
- See example

$$\begin{array}{ccc} \text{fcn_form} & & = \\ / \mid \backslash & & / \backslash \\ P \ V^+ \ E & \Rightarrow & P \ \text{+lambda} \\ & & / \backslash \\ & & V \ .E \end{array}$$

- V+ means "one or more", to be repeated likewise on right side.
- The 'dot' indicates the location of the pattern's repetition.

Standardizing 'fcn_form':

```
let f x y z = x+y+z in f 1 2 3
```



Standardizing Tuples

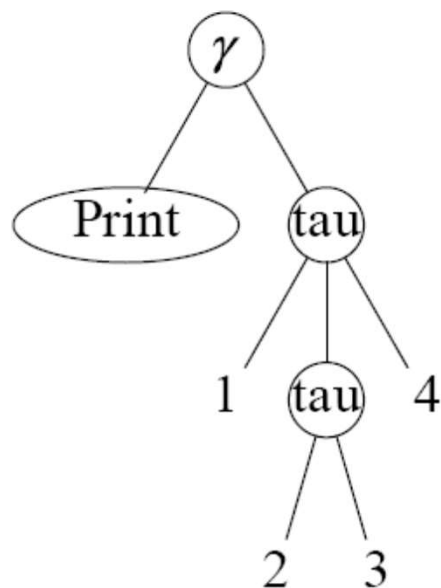
- See example

```
tau  =>  ++gamma
|        / \
E++     gamma E
        / \
        aug .nil
```

E++ means “two or more” E’s.

Standardizing tuples:

Print (1, (2, 3), 4)



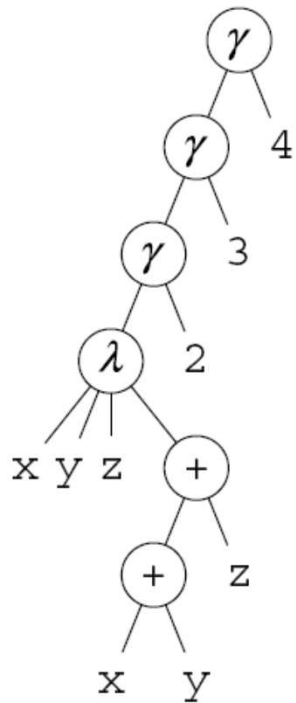
Standardizing Multi-Parameter Functions

- See example

| | | |
|--------|----|----------|
| lambda | => | ++lambda |
| / \ | | / \ |
| V++ E | | V .E |

Standardizing multi-parameter functions:

```
(fn x y z. x + y + z) 2 3 4
```



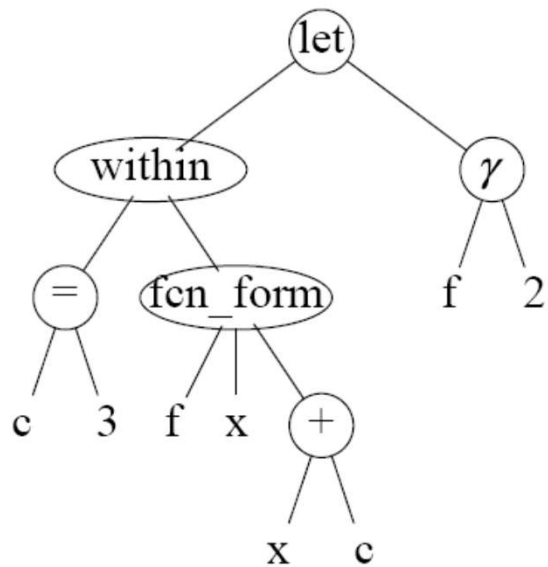
Standardizing the 'within'

- See example

$$\begin{array}{ccc} \text{within} & \Rightarrow & \\ \begin{array}{cc} / & \backslash \\ = & = \\ /\backslash & / \backslash \\ X1 \ E1 & X2 \ E2 \end{array} & & \begin{array}{c} = \\ \begin{array}{cc} / & \backslash \\ X2 \ \text{gamma} & \\ & / \backslash \\ \text{lambda} & E1 \\ & / \backslash \\ X1 & E2 \end{array} \end{array} \end{array}$$

Standardizing the 'within':

```
let c = 3 within f x = x + c in f 2
```



Standardizing Unary and Binary Operators

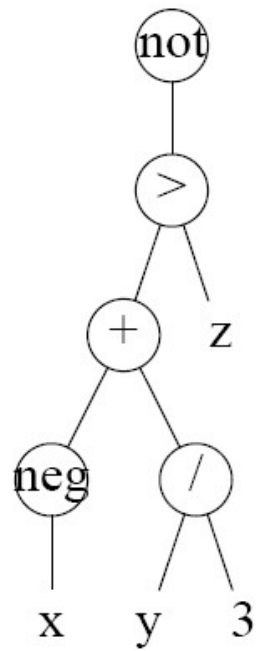
- See example

Uop => gamma
| / \
E Uop E

Op => gamma
/ \ / \
E1 E2 gamma E2
 / \
 Op E1

Standardizing unary and binary operators:

not - x + y / 3 > z



Standardizing the '@' Operator

- See example

@ => gamma

/ | \ / \

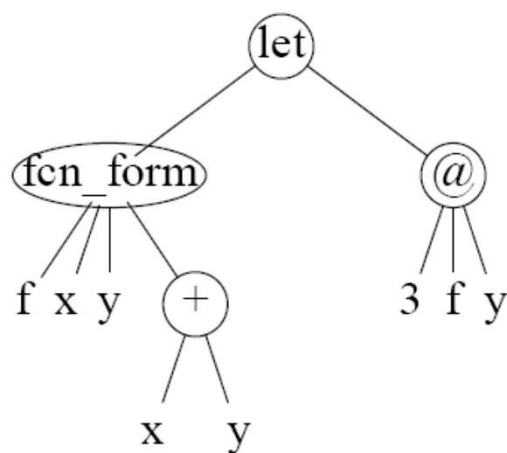
E1 N E2 gamma E2

 / \

 N E1

Standardizing the '@' operator:

```
let f x y = x + y in 3 @f y
```



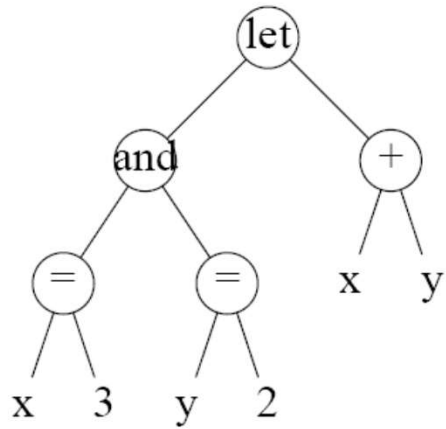
Standardizing Simultaneous Definitions

- See example

| | | |
|-----|---------|---|
| and | => | = |
| | /\ | |
| =++ | , tau | |
| /\ | | |
| X E | X++ E++ | |

Standardizing simultaneous definitions:

let $x = 3$ and $y = 2$ in $x + y$



Standardizing Simultaneous Definitions (cont'd)

| | | |
|--------|------|---------------|
| lambda | => | lambda |
| / \ | | / \ |
| , E | Temp | ++gamma |
| | | / \ |
| X++i | | lambda gamma |
| | | / \ / \ |
| | | X.i .E Temp \ |
| | | <INTEGER:i> |

Standardizing the Conditional Operator

$\rightarrow \Rightarrow$ gamma

$\begin{array}{c} / \mid \backslash \\ B \ T \ F \end{array} \quad \begin{array}{c} / \backslash \\ \text{gamma} \ \text{nil} \end{array}$

$\begin{array}{c} / \backslash \\ \text{gamma} \ \text{lambda} \end{array}$

$\begin{array}{c} / \backslash \quad / \backslash \\ \text{gamma} \ \text{lambda} \ () \ F \end{array}$

$\begin{array}{c} / \backslash \ / \backslash \\ \text{Cond} \ B \quad () \ T \end{array}$

$\text{Cond} = \text{fn } B. \text{fn } T. \text{fn } F. B \rightarrow T \mid F$

Circular semantic definition !



Circular Semantic Definitions

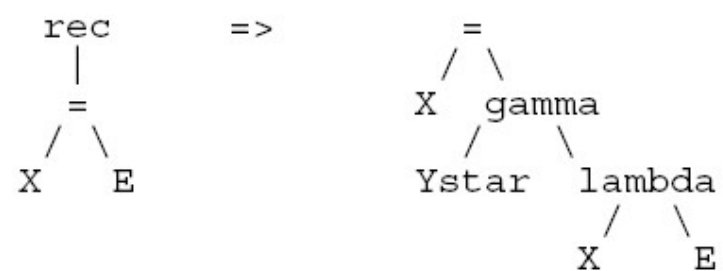
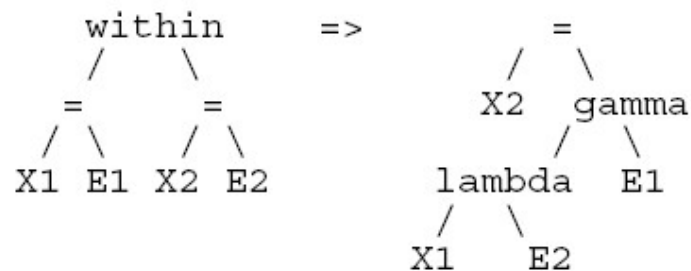
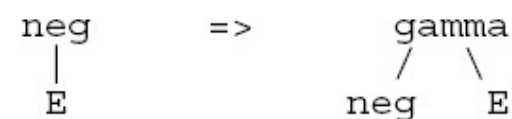
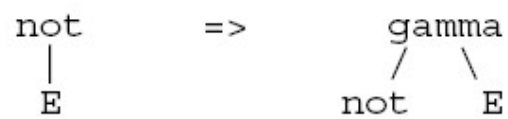
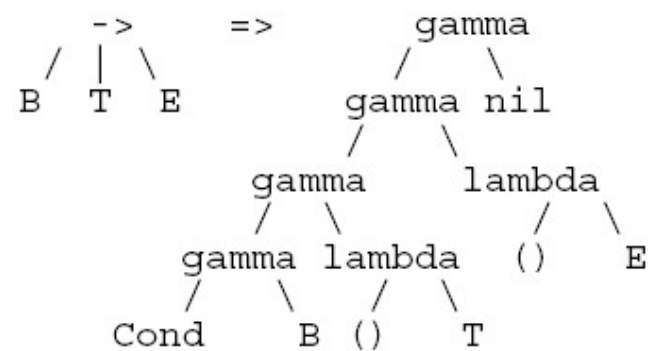
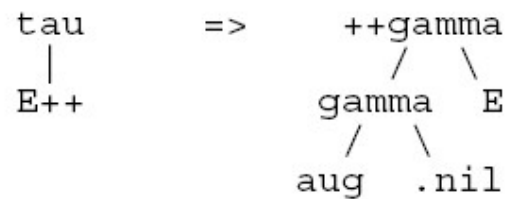
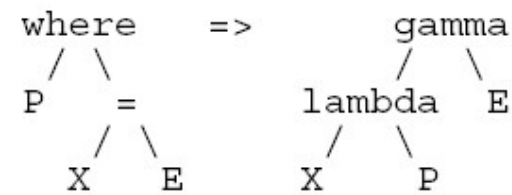
- K. Goedel's Incompleteness Theorem (1930's):
Every logic system is either incomplete or inconsistent.
- Incomplete is preferable to inconsistent.
- Inevitable in semantics
- English dictionary is useless to someone who understands no English.

Standardizing 'rec'

- Will do later

$$\begin{array}{lcl} \text{rec} & \Rightarrow & = \\ | & & /\backslash \\ = & & X \text{ gamma} \\ /\backslash & & / \backslash \\ X \ E & & Y\text{star} \ \text{lambda} \\ & & / \backslash \\ & & X \ E \end{array}$$

RPAL SUBTREE TRANSFORMATIONAL GRAMMA



$$\begin{array}{c} \text{fcn_form} \\ / \quad | \quad \backslash \\ P \quad V+ \quad E \end{array} \Rightarrow \begin{array}{c} = \\ / \quad \backslash \\ P \quad +\text{lambda} \\ \quad / \quad \backslash \\ \quad V \quad .E \end{array}$$

$$\begin{array}{c} \text{lambda} \\ / \quad \backslash \\ X++i \quad E \end{array} \Rightarrow \begin{array}{c} \text{lambda} \\ / \quad \backslash \\ \text{Temp} \quad ++\text{gamma} \\ \quad / \quad \backslash \\ \quad \text{lambda} \quad \text{gamma} \\ \quad / \quad \backslash \quad / \quad \backslash \\ X.i \quad .E \quad \text{Temp} \quad <\text{INTEGER:i}> \end{array}$$

$$\begin{array}{c} \text{lambda} \\ / \quad \backslash \\ V++ \quad E \end{array} \Rightarrow \begin{array}{c} ++\text{lambda} \\ / \quad \backslash \\ V \quad .E \end{array}$$

$$\begin{array}{c} \text{Op} \\ / \quad \backslash \\ E1 \quad E2 \end{array} \Rightarrow \begin{array}{c} \text{gamma} \\ / \quad \backslash \\ \text{gamma} \quad E2 \\ / \quad \backslash \\ \text{Op} \quad E1 \end{array}$$

$$\begin{array}{c} \text{and} \\ | \\ =++ \\ / \quad \backslash \\ X \quad E \end{array} \Rightarrow \begin{array}{c} = \\ / \quad \backslash \\ , \quad \text{tau} \\ | \quad | \\ X++ \quad E++ \end{array}$$

$$\begin{array}{c} @ \\ / \quad | \quad \backslash \\ E1 \quad N \quad E2 \end{array} \Rightarrow \begin{array}{c} \text{gamma} \\ / \quad \backslash \\ \text{gamma} \quad E2 \\ / \quad \backslash \\ N \quad E1 \end{array}$$

$$\begin{array}{c} \text{Uop} \\ | \\ E \end{array} \Rightarrow \begin{array}{c} \text{gamma} \\ / \quad \backslash \\ \text{Uop} \quad E \end{array}$$

Op in [aug,or,&,+,-,/,**,gr ...]

Uop in [not, neg]



Summary

- Transform AST into a standardized tree (ST).
- ST is binary.
- All internal nodes in the ST are either gamma or lambda.
- It's called "desugaring" the program: reducing it to two constructs: function abstraction/definition (lambda), and function application (gamma).



Thank You!



REFERENCES

- Programming Language Pragmatics by Michael L. Scott. 3rd edition. Morgan Kaufmann Publishers. (April 2009).
- Lecture Slides of Dr.Malaka Walpola and Dr.Bermudez