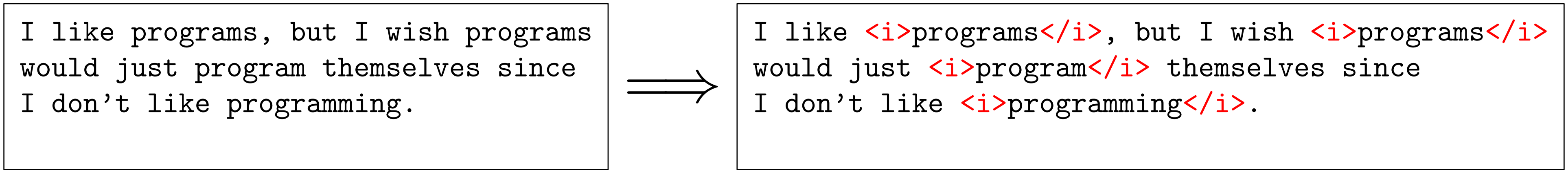
Learning Programs:
A Hierarchical Bayesian Approach

ICML - Haifa, Israel June 24, 2010

Percy Liang Michael I. Jordan Dan Klein

Motivating Application: Repetitive Text Editing



```
I like programs, but I wish programs
would just program themselves since
I don't like programming.
```

⟹

```
I like <i>programs</i>, but I wish <i>programs</i>
would just <i>program</i> themselves since
I don't like <i>programming</i>.
```

Motivating Application: Repetitive Text Editing

```
I like programs, but I wish programs
would just program themselves since
I don't like programming.
```

$\Longrightarrow$

```
I like <i>programs</i>, but I wish <i>programs</i>
would just <i>program</i> themselves since
I don't like <i>programming</i>.
```

Goal: Programming by Demonstration

If the user demonstrates italicizing the first occurrence, can we generalize to the remaining?

Motivating Application: Repetitive Text Editing

```
I like programs, but I wish programs
would just program themselves since
I don’t like programming.
```

⟹

```
I like <i>programs</i>, but I wish <i>programs</i>
would just <i>program</i> themselves since
I don’t like <i>programming</i>.
```

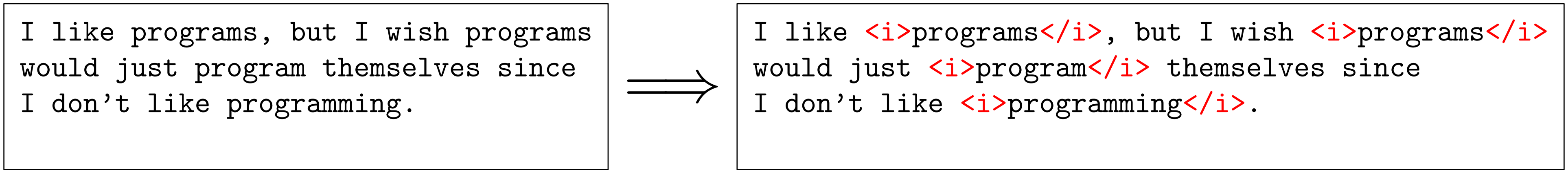
Goal: Programming by Demonstration

If the user demonstrates italicizing the first occurrence, can we generalize to the remaining?

Solution: represent task by a program to be learned

1. Move to next occurrence of word with prefix `program`
2. Insert `<i>`
3. Move to end of word
4. Insert `</i>`

Motivating Application: Repetitive Text Editing



```
I like programs, but I wish programs
would just program themselves since
I don't like programming.
```

⟹

```
I like <i>programs</i>, but I wish <i>programs</i>
would just <i>program</i> themselves since
I don't like <i>programming</i>.
```

Goal: Programming by Demonstration

If the user demonstrates italicizing the first occurrence, can we generalize to the remaining?

Solution: represent task by a program to be learned

1. Move to next occurrence of word with prefix `program`
2. Insert `<i>`
3. Move to end of word
4. Insert `</i>`

Challenge: learn from very few examples

General Setup

Goal:

$$\begin{gathered}(X_1, Y_1)\\ \cdots \\ (X_n, Y_n)\end{gathered}$$

Training data

General Setup

Goal:

$$\begin{array}{c} (X_1, Y_1) \\ \cdots \\ (X_n, Y_n) \end{array} \quad \Longrightarrow \quad Z \text{ such that } (Z\ X_j) = Y_j$$

Training data Consistent program

General Setup

Goal:

$$\begin{matrix}(X_1, Y_1) \\ \cdots \\ (X_n, Y_n)\end{matrix} \quad \Longrightarrow \quad Z \text{ such that } (Z\ X_j) = Y_j$$

Training data | Consistent program

Challenge:

When n small, many programs consistent with training data.

```
I like <i>programs</i>, but I wish programs
would just program themselves since
I don't like programming.
```

Move to beginning of third word, ...
Move to beginning of word after `like`, ...
Move 7 spaces to the right, ...
Move to word with prefix `program`, ...
$\cdots$

General Setup

Goal:

$$\begin{matrix}(X_1, Y_1) \\ \cdots \\ (X_n, Y_n)\end{matrix} \quad \Longrightarrow \quad Z \text{ such that } (Z\ X_j) = Y_j$$

Training data

Consistent program

Challenge:

When n small, many programs consistent with training data.

```
I like <i>programs</i>, but I wish programs
would just program themselves since
I don't like programming.
```

Move to beginning of third word, ...
Move to beginning of word after `like`, ...
Move 7 spaces to the right, ...
Move to word with prefix `program`, ...
$\cdots$

Which program to choose?

Key Intuition

One task:

Want to choose a program which is simple (Occam's razor).

Examples $\Longrightarrow$ Z

Key Intuition

One task:

Want to choose a program which is simple (Occam's razor).

$$\text{Examples} \implies Z$$

What's the right complexity metric (prior)?

Key Intuition

One task:

Want to choose a program which is simple (Occam's razor).

Examples $\Longrightarrow$ Z

What's the right complexity metric (prior)? No general answer.

Key Intuition

One task:

Want to choose a program which is simple (Occam's razor).

$$\text{Examples} \quad \Longrightarrow \quad Z$$

What's the right complexity metric (prior)? No general answer.

Multiple tasks:

$$\begin{array}{ccc} \text{Task 1 examples} & \Longrightarrow & Z_1 \\ \cdots & & \cdots \\ \text{Task } K \text{ examples} & \Longrightarrow & Z_K \end{array}$$

Key Intuition

One task:

Want to choose a program which is simple (Occam's razor).

$$\text{Examples} \quad \Longrightarrow \quad Z$$

What's the right complexity metric (prior)? No general answer.

Multiple tasks:

$$\begin{array}{ccc} \text{Task 1 examples} & \Longrightarrow & Z_1 \\ \cdots & & \cdots \\ \text{Task } K \text{ examples} & \Longrightarrow & Z_K \end{array}$$

Find programs that share common subprograms.

Key Intuition

One task:

Want to choose a program which is simple (Occam's razor).

Examples $\Longrightarrow$ Z

What's the right complexity metric (prior)? No general answer.

Multiple tasks:

$$\begin{array}{ccc} \text{Task 1 examples} & \Longrightarrow & Z_1 \\ \cdots & & \cdots \\ \text{Task } K \text{ examples} & \Longrightarrow & Z_K \end{array}$$

Find programs that share common subprograms.

- Programs do tend to share common components.

Key Intuition

One task:

Want to choose a program which is simple (Occam's razor).

$$\text{Examples} \quad \Longrightarrow \quad Z$$

What's the right complexity metric (prior)? No general answer.

Multiple tasks:

$$\begin{array}{cc} \text{Task 1 examples} & \Longrightarrow \\ \cdots & \\ \text{Task } K \text{ examples} & \Longrightarrow \end{array} \boxed{\begin{array}{c} Z_1 \\ \cdots \\ Z_K \end{array}}$$

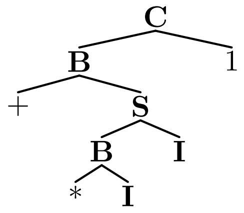
Find programs that share common subprograms.

- Programs do tend to share common components.
- Penalize joint complexity of all K programs.

Outline of Proposed Solution

Program representation: What are subprograms?

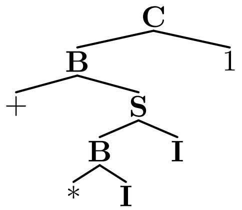
Combinatory logic



Outline of Proposed Solution

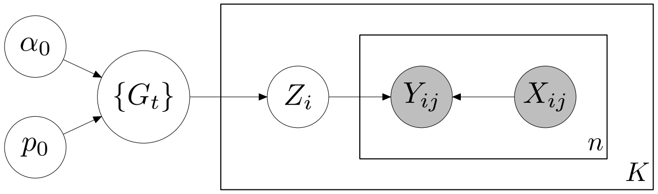
Program representation: What are subprograms?

Combinatory logic



Probabilistic model: Which programs are favorable?

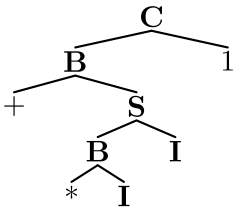
Nonparametric Bayes



Outline of Proposed Solution

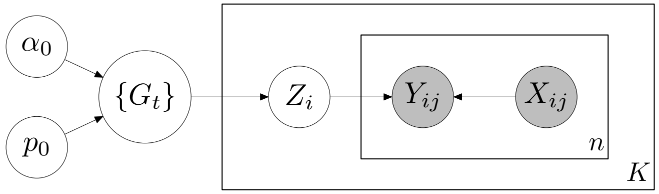
Program representation: What are subprograms?

Combinatory logic



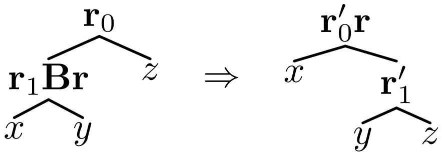
Probabilistic model: Which programs are favorable?

Nonparametric Bayes



Statistical inference: How do we search for good programs?

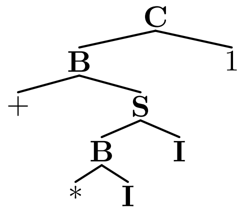
MCMC



Outline of Proposed Solution

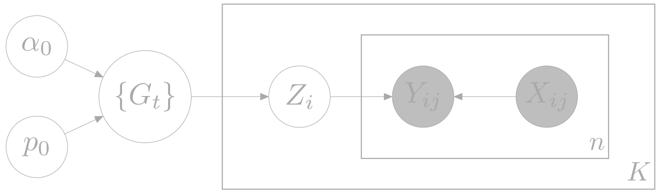
Program representation: What are subprograms?

Combinatory logic



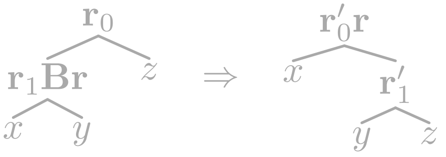
Probabilistic model: Which programs are favorable?

Nonparametric Bayes



Statistical inference: How do we search for good programs?

MCMC



Representation: What Language?

Goal: allow sharing of subprograms

Representation: What Language?

Goal: allow sharing of subprograms

Our language:

Combinatory logic [Schönfinkel, 1924]

Representation: What Language?

Goal: allow sharing of subprograms

Our language:

Combinatory logic [Schönfinkel, 1924]

+ higher-order combinators (new)

+ routing intuition, visual representation (new)

Representation: What Language?

Goal: allow sharing of subprograms

Our language:

Combinatory logic [Schönfinkel, 1924]

$+$ higher-order combinators (new)

$+$ routing intuition, visual representation (new)

Properties: no mutation, no variables $\Rightarrow$ simple semantics

Representation: What Language?

Goal: allow sharing of subprograms

Our language:

Combinatory logic [Schönfinkel, 1924]

+ higher-order combinators (new)

+ routing intuition, visual representation (new)

Properties: no mutation, no variables ⇒ simple semantics

Result:

- Programs are trees
- Subprograms are subtrees

Programs with No Arguments

Example: compute $\min(3,4)$

Programs with No Arguments

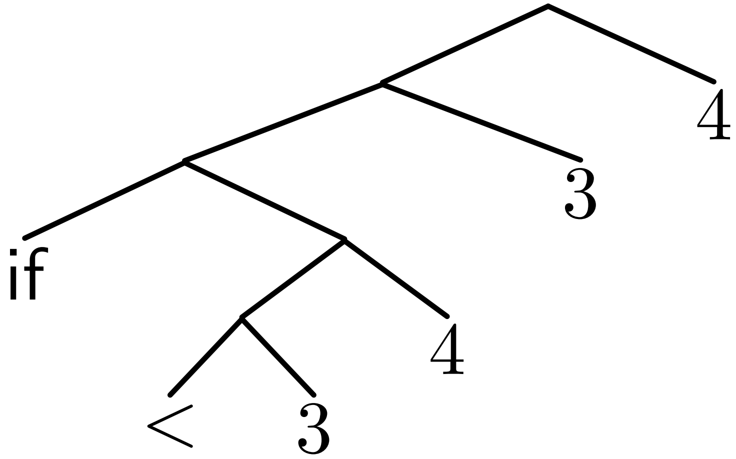
Example: compute $\min(3, 4)$

```
(if (< 3 4) 3 4)
```

Programs with No Arguments

Example: compute $\min(3, 4)$

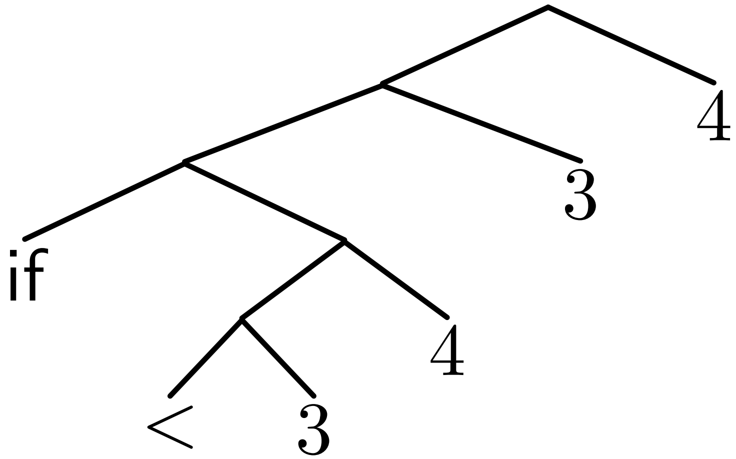
(if $(<$ 3 4) 3 4)



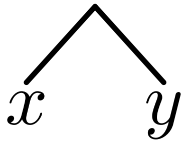
Programs with No Arguments

Example: compute $\min(3, 4)$

(if ($<$ 3 4) 3 4)



General:

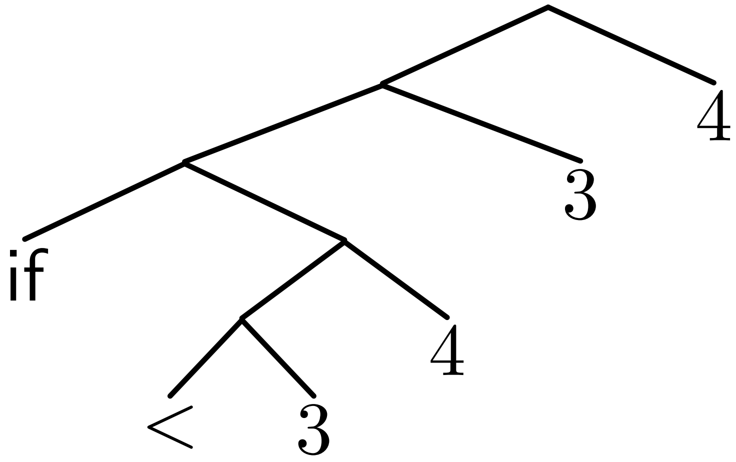


$\Rightarrow$ result of applying function x to argument y

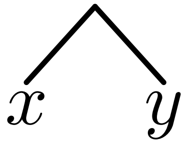
Programs with No Arguments

Example: compute $\min(3, 4)$

(if (< 3 4) 3 4)



General:



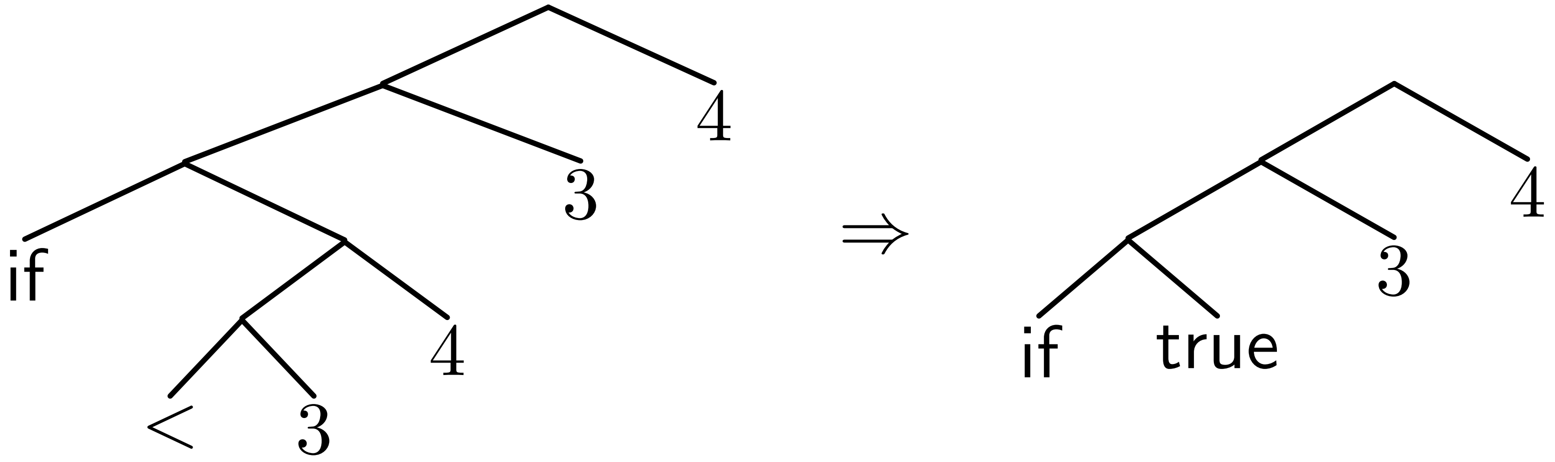
$\Rightarrow$ result of applying function x to argument y

Arguments are curried

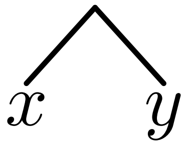
Programs with No Arguments

Example: compute $\min(3, 4)$

(if ($<$ 3 4) 3 4)   (if true 3 4)



General:



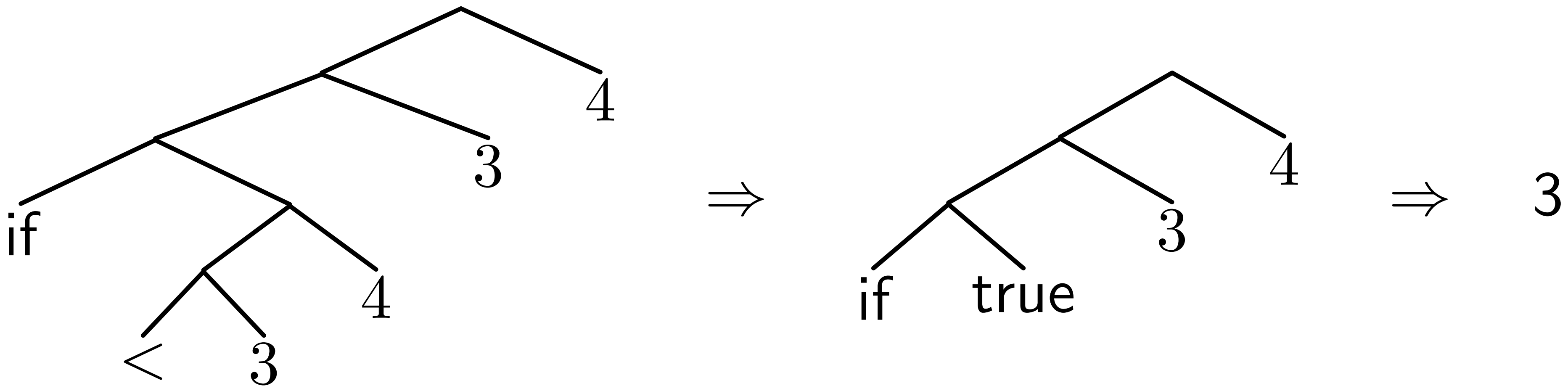
$\Rightarrow$ result of applying function x to argument y

Arguments are curried

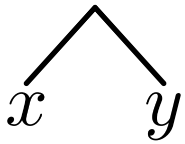
Programs with No Arguments

Example: compute $\min(3, 4)$

(if (< 3 4) 3 4) (if true 3 4)



General:

 $\Rightarrow$ result of applying function x to argument y

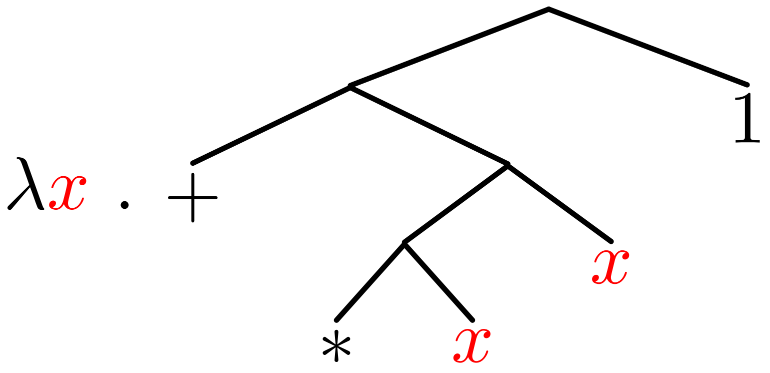
Arguments are curried

Programs with One Argument

Example: $x \mapsto x^2 + 1$

Programs with One Argument

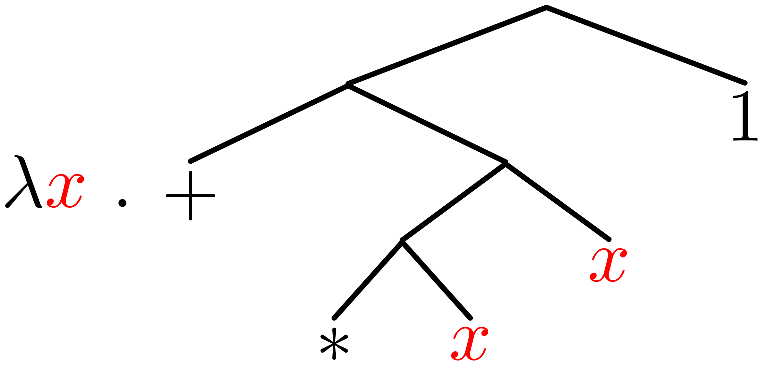
Example: $x \mapsto x^2 + 1$



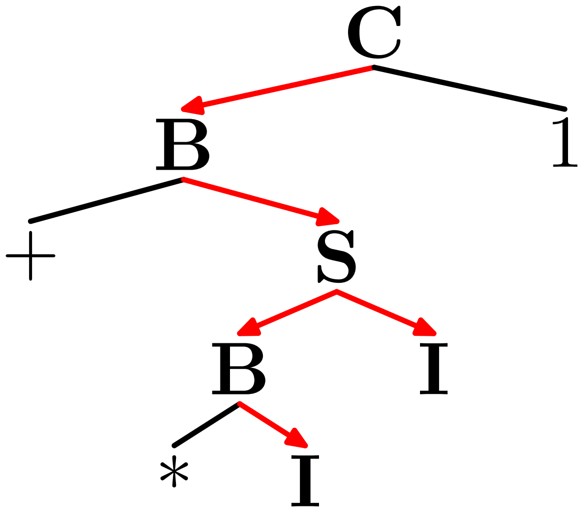
Lambda calculus

Programs with One Argument

Example: $x \mapsto x^2 + 1$



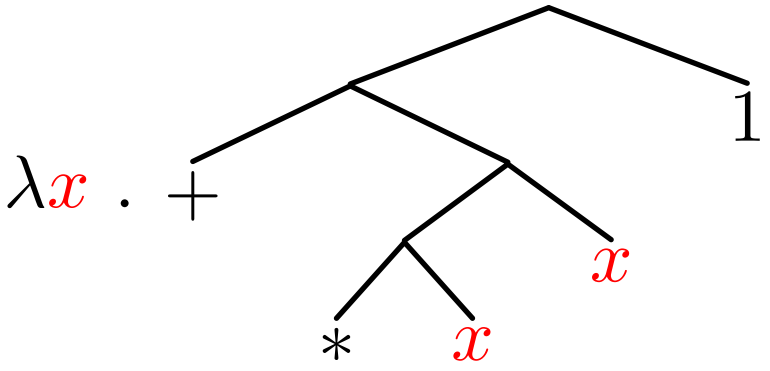
Lambda calculus



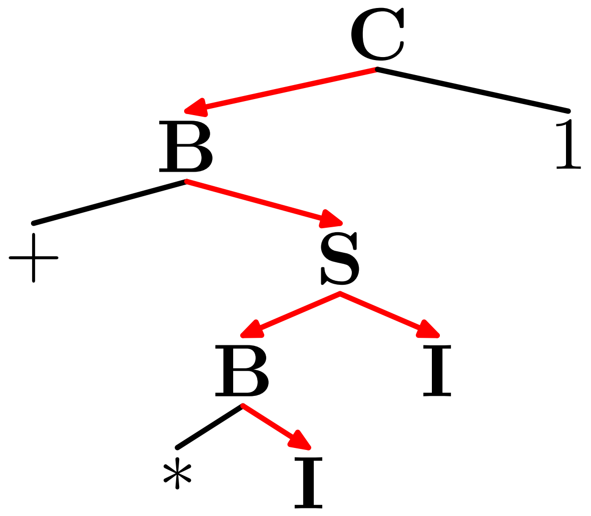
Combinatory logic

Programs with One Argument

Example: $x \mapsto x^2 + 1$



Lambda calculus



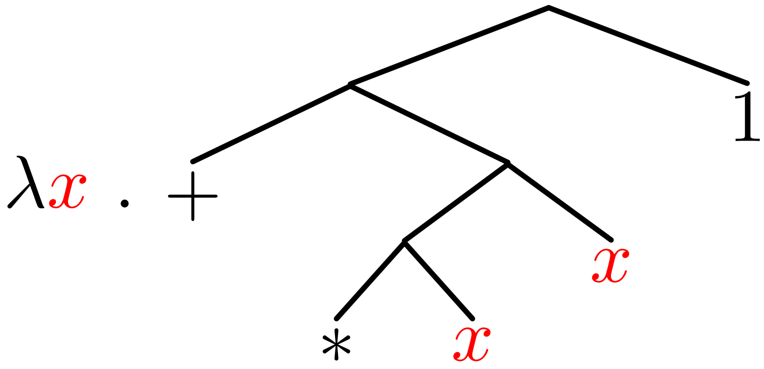
Combinatory logic

Intuition:

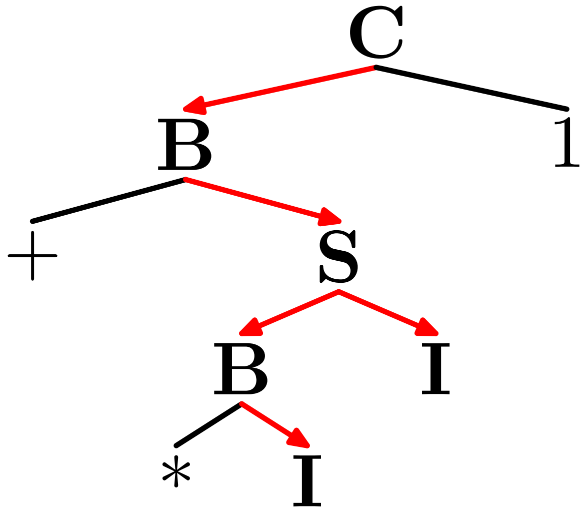
Combinators $\{\mathbf{B}, \mathbf{C}, \mathbf{S}, \mathbf{I}\}$ encode placement of arguments

Programs with One Argument

Example: $x \mapsto x^2 + 1$



Lambda calculus

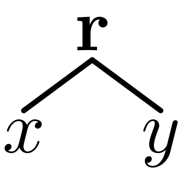


Combinatory logic

Intuition:

Combinators $\{\mathbf{B}, \mathbf{C}, \mathbf{S}, \mathbf{I}\}$ encode placement of arguments

Semantics:

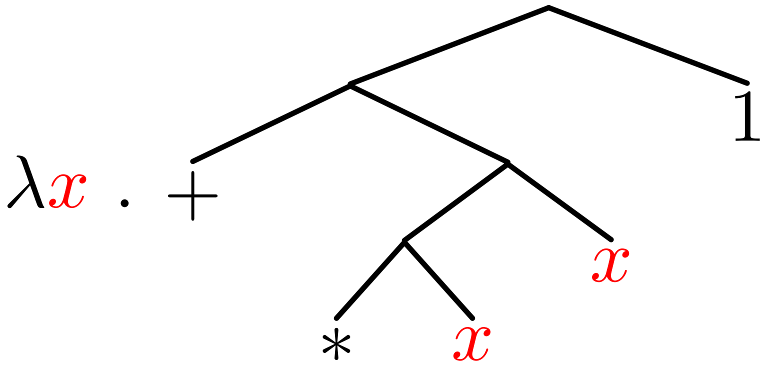


$\Leftrightarrow \quad (\mathbf{r}\ x\ y)$

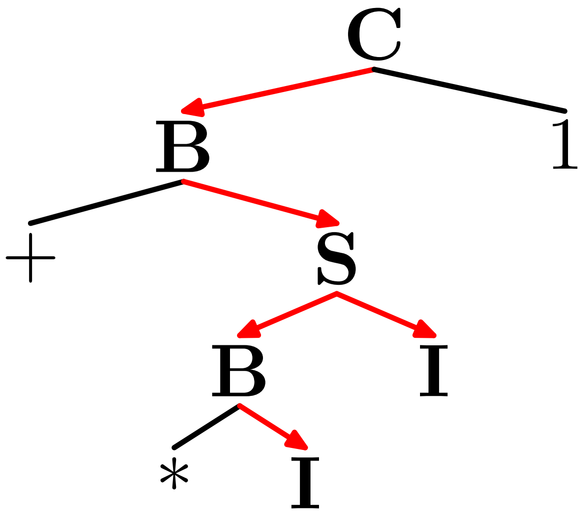
$\mathbf{r} \in \{\mathbf{B}, \mathbf{C}, \mathbf{S}, \mathbf{I}\}$

Programs with One Argument

Example: $x \mapsto x^2 + 1$



Lambda calculus

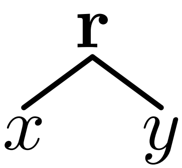


Combinatory logic

Intuition:

Combinators $\{\mathbf{B}, \mathbf{C}, \mathbf{S}, \mathbf{I}\}$ encode placement of arguments

Semantics:



$\Leftrightarrow \quad (\mathbf{r}\ x\ y)$

$\mathbf{r} \in \{\mathbf{B}, \mathbf{C}, \mathbf{S}, \mathbf{I}\}$

Rules:

$(\mathbf{B}\ f\ g\ x) = (f\ (g\ x))$

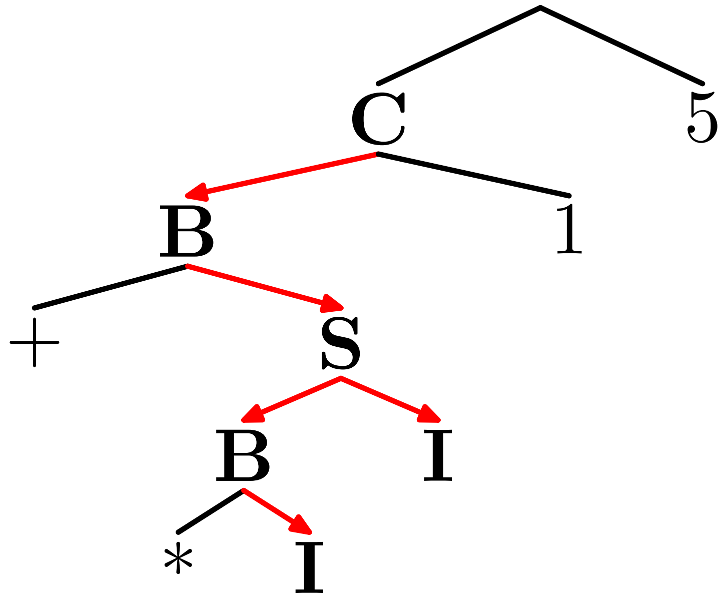
$\ldots$

Programs with One Argument

Example: Apply $x \mapsto x^2 + 1$ to 5

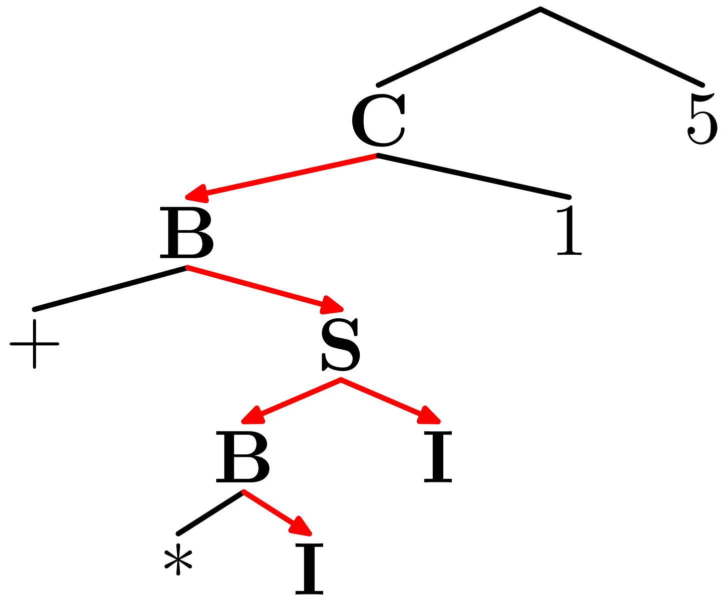
Programs with One Argument

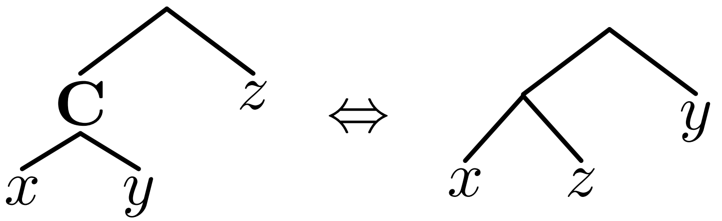
Example: Apply $x \mapsto x^2 + 1$ to 5



Programs with One Argument

Example: Apply $x \mapsto x^2 + 1$ to 5

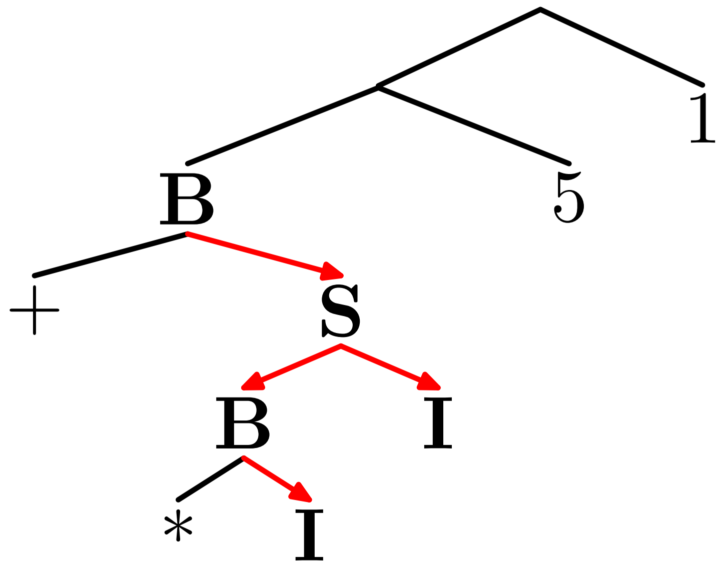


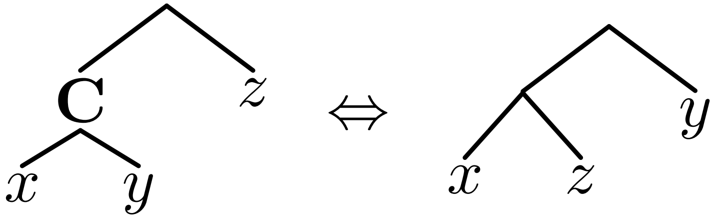


route left

Programs with One Argument

Example: Apply $x \mapsto x^2 + 1$ to 5

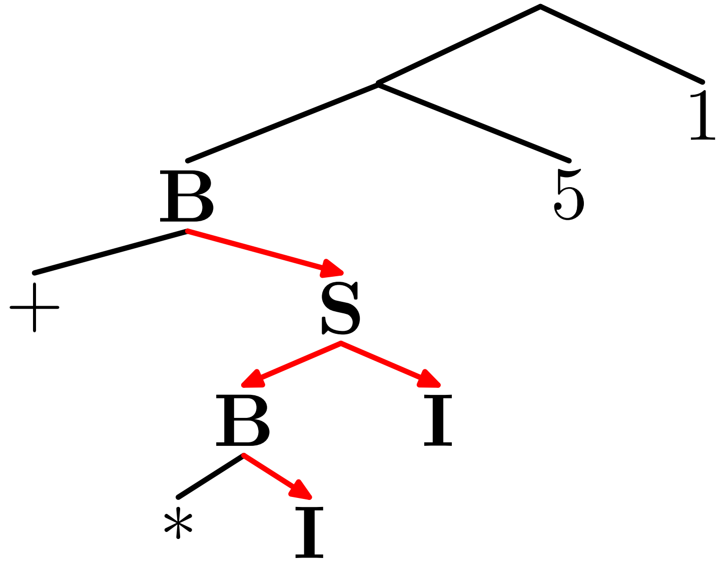


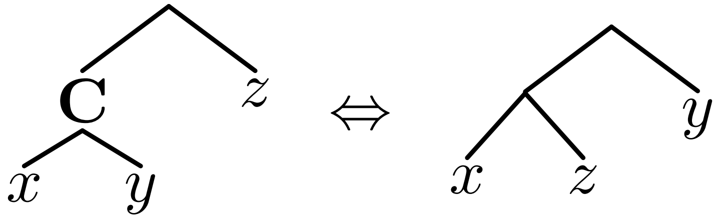


route left

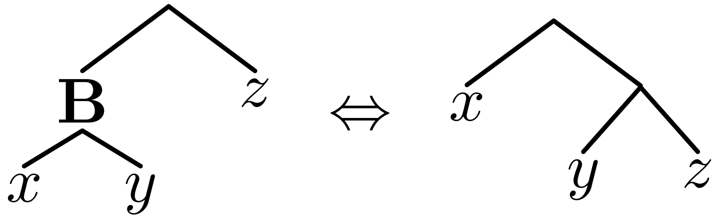
Programs with One Argument

Example: Apply $x \mapsto x^2 + 1$ to 5





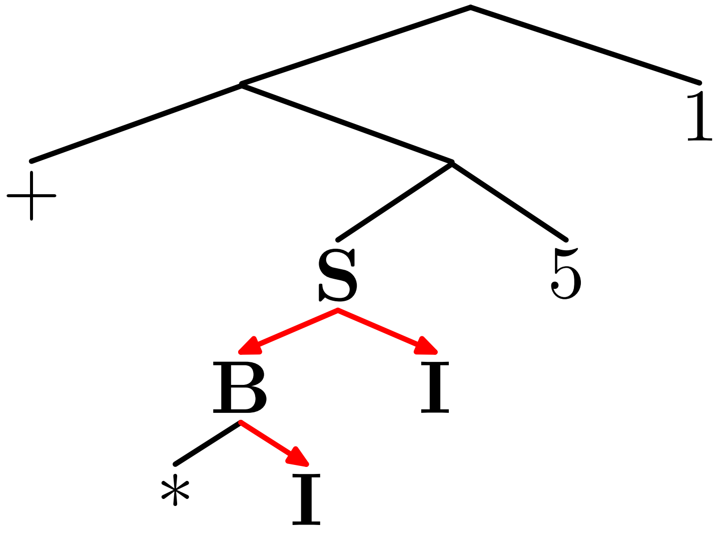
route left

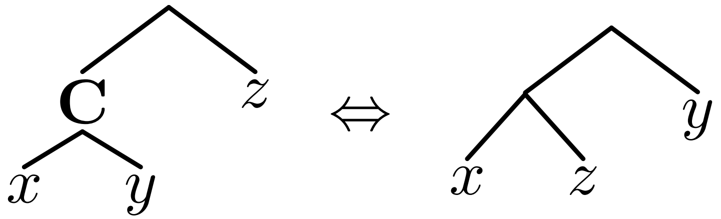


route right

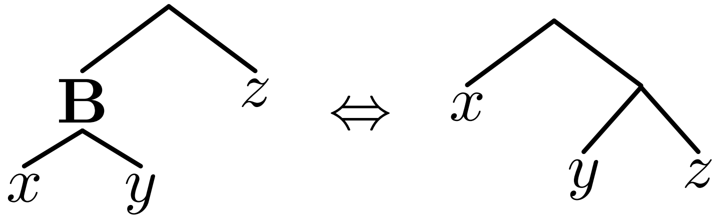
Programs with One Argument

Example: Apply $x \mapsto x^2 + 1$ to 5





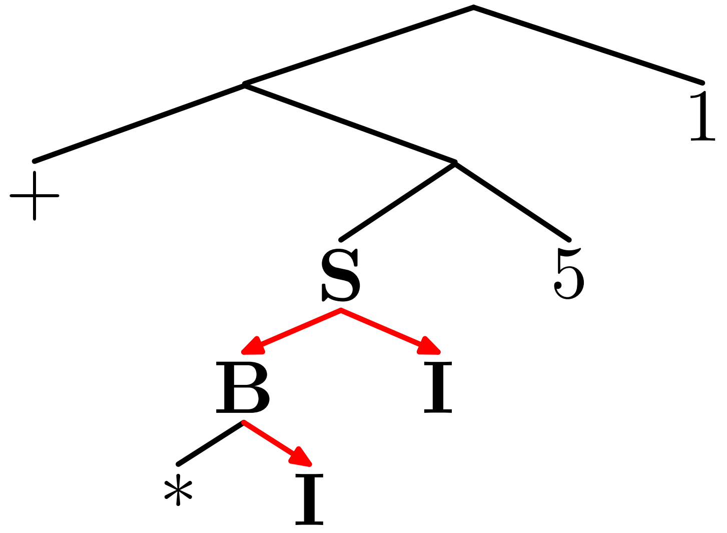
route left

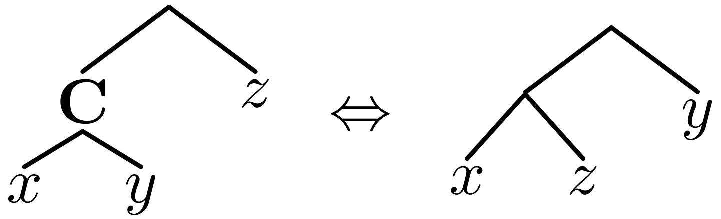


route right

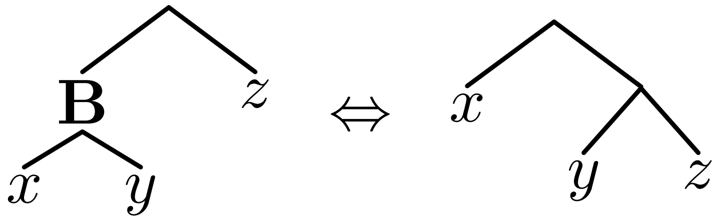
Programs with One Argument

Example: Apply $x \mapsto x^2 + 1$ to 5

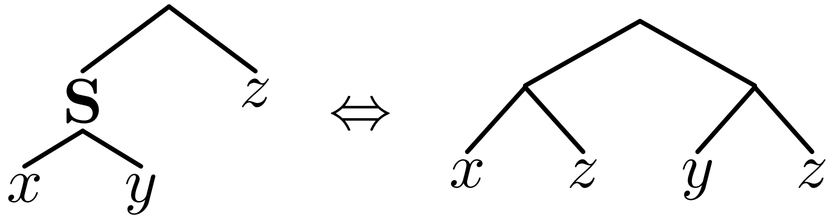




route left



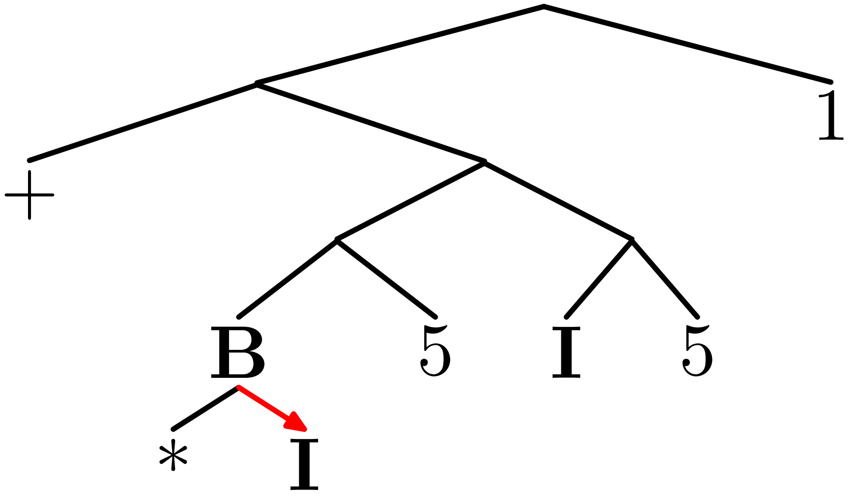
route right

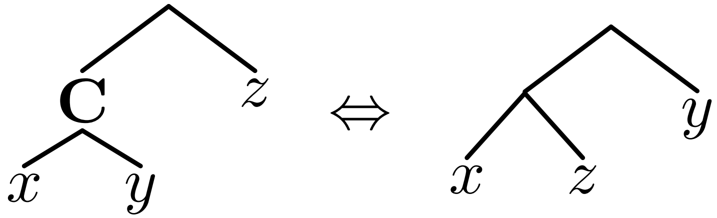


route left and right

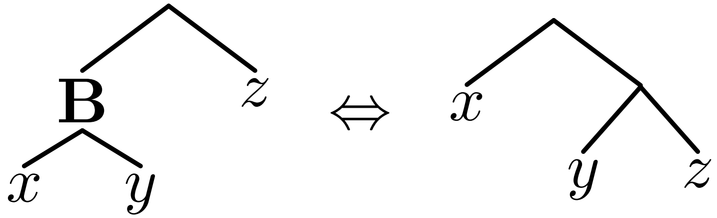
Programs with One Argument

Example: Apply $x \mapsto x^2 + 1$ to 5

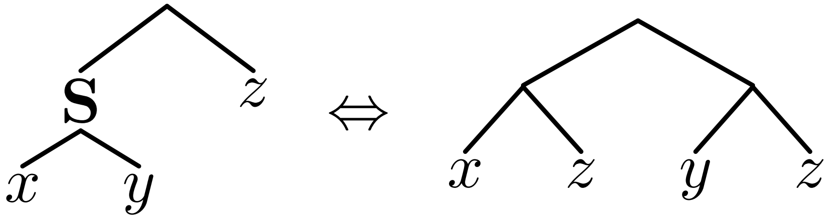




route left



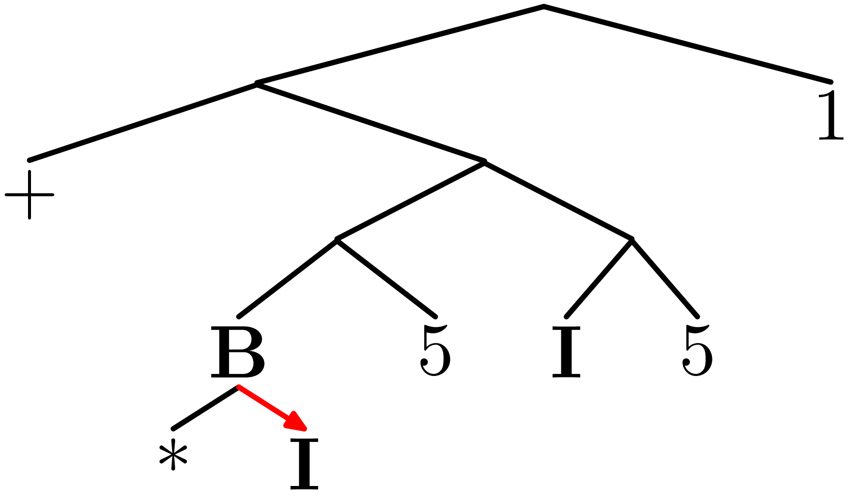
route right

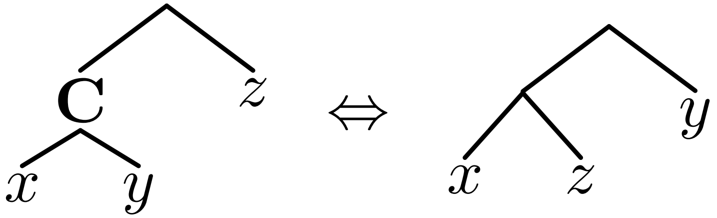


route left and right

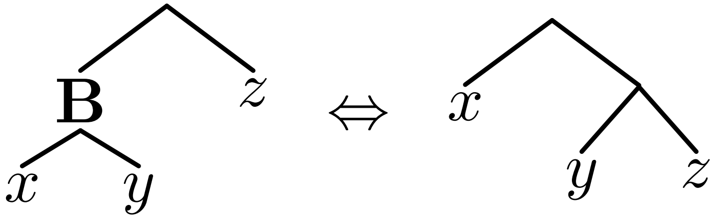
Programs with One Argument

Example: Apply $x \mapsto x^2 + 1$ to 5

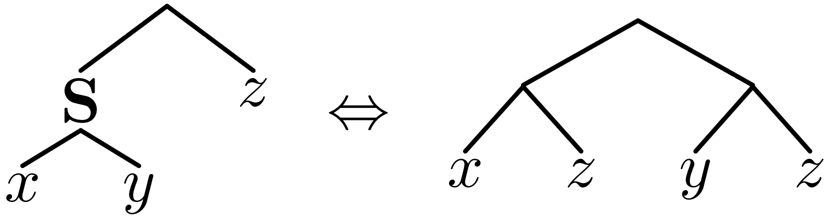




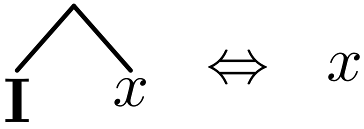
route left



route right



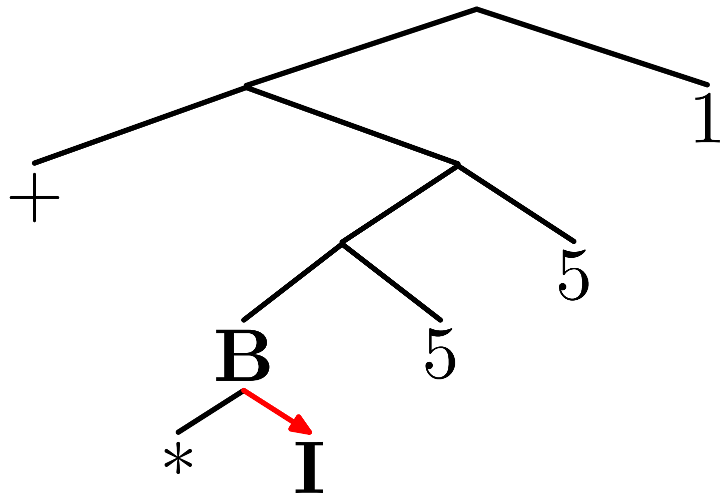
route left and right

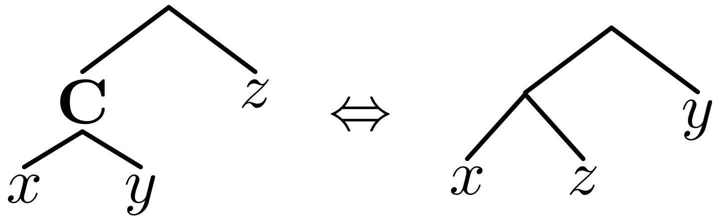


stop

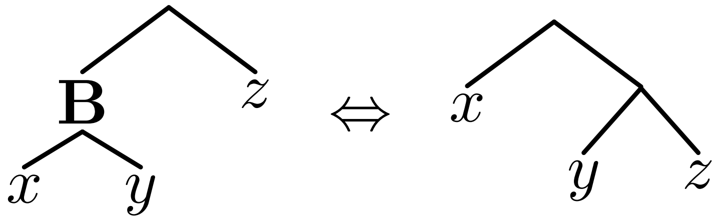
Programs with One Argument

Example: Apply $x \mapsto x^2 + 1$ to 5

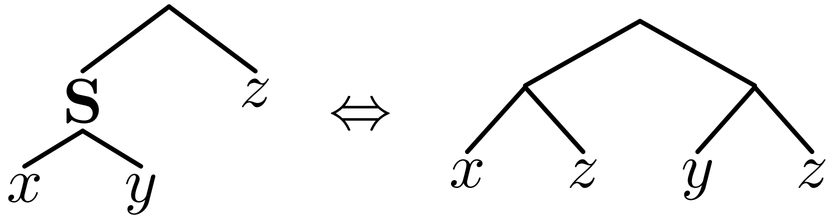




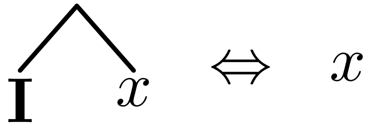
route left



route right



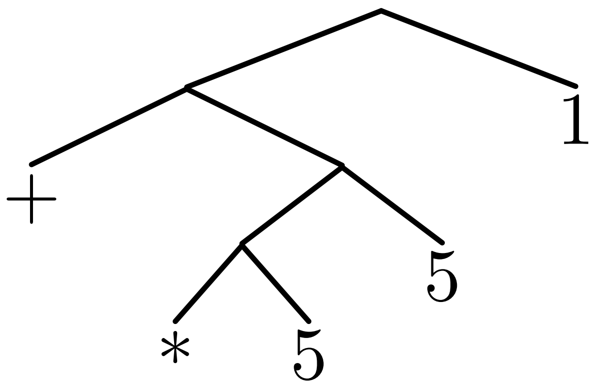
route left and right

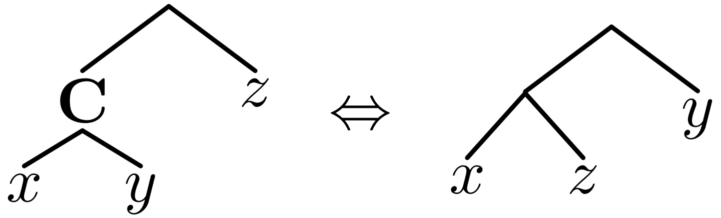


stop

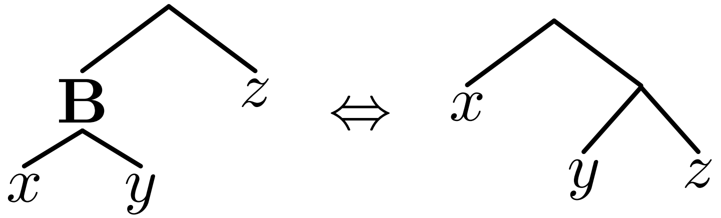
Programs with One Argument

Example: Apply $x \mapsto x^2 + 1$ to 5

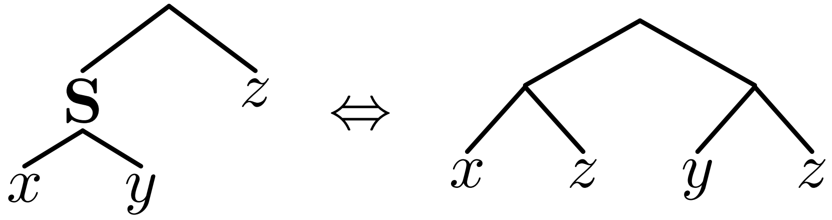




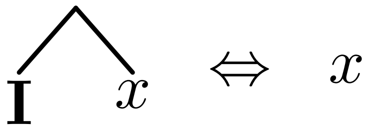
route left



route right



route left and right



stop

Programs with Multiple Arguments

Example: $(x, y) \mapsto \min(x, y)$

Programs with Multiple Arguments

Example: $(x, y) \mapsto \min(x, y)$

Classical: first-order combinators $\{\mathbf{B}, \mathbf{C}, \mathbf{S}, \mathbf{I}\}$

Complete basis, so can implement $\min$, but cumbersome

Programs with Multiple Arguments

Example: $(x, y) \mapsto \min(x, y)$

Classical: first-order combinators $\{\mathbf{B}, \mathbf{C}, \mathbf{S}, \mathbf{I}\}$

Complete basis, so can implement $\min$, but cumbersome

New: **higher-order combinators** $\{\mathbf{B}, \mathbf{C}, \mathbf{S}, \mathbf{I}\}^*$

Infinite basis, but resulting programs are more intuitive

e.g., $\mathbf{CS}$ routes 1st arg. left, 2nd arg. left and right

Programs with Multiple Arguments

Example: $(x, y) \mapsto \min(x, y)$

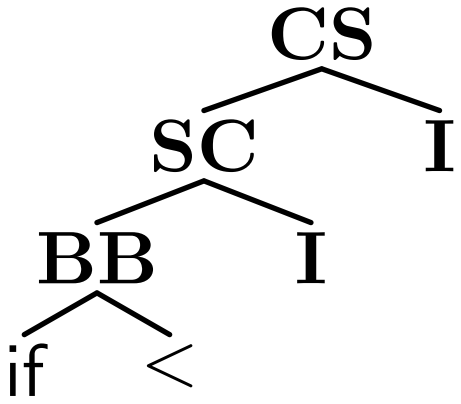
Classical: first-order combinators $\{\mathbf{B}, \mathbf{C}, \mathbf{S}, \mathbf{I}\}$

Complete basis, so can implement $\min$, but cumbersome

New: **higher-order combinators** $\{\mathbf{B}, \mathbf{C}, \mathbf{S}, \mathbf{I}\}^*$

Infinite basis, but resulting programs are more intuitive

e.g., $\mathbf{CS}$ routes 1st arg. left, 2nd arg. left and right



Programs with Multiple Arguments

Example: $(x, y) \mapsto \min(x, y)$

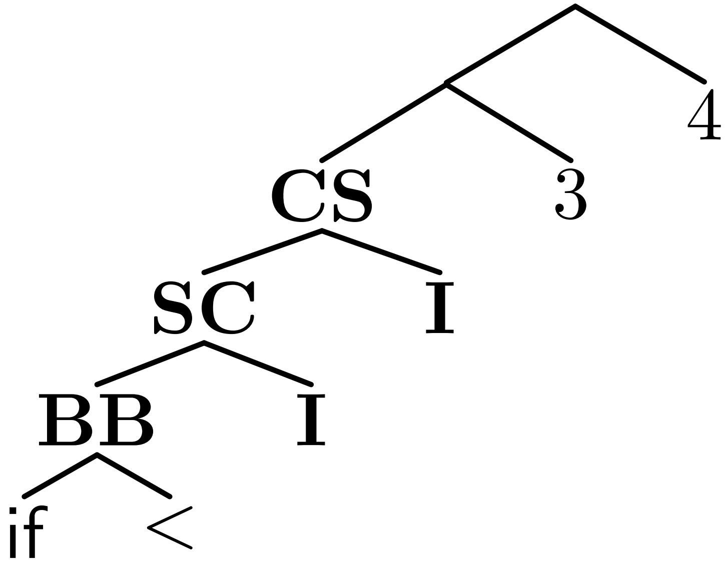
Classical: first-order combinators $\{\mathbf{B}, \mathbf{C}, \mathbf{S}, \mathbf{I}\}$

Complete basis, so can implement $\min$, but cumbersome

New: **higher-order combinators** $\{\mathbf{B}, \mathbf{C}, \mathbf{S}, \mathbf{I}\}^*$

Infinite basis, but resulting programs are more intuitive

e.g., $\mathbf{CS}$ routes 1st arg. left, 2nd arg. left and right



Programs with Multiple Arguments

Example: $(x, y) \mapsto \min(x, y)$

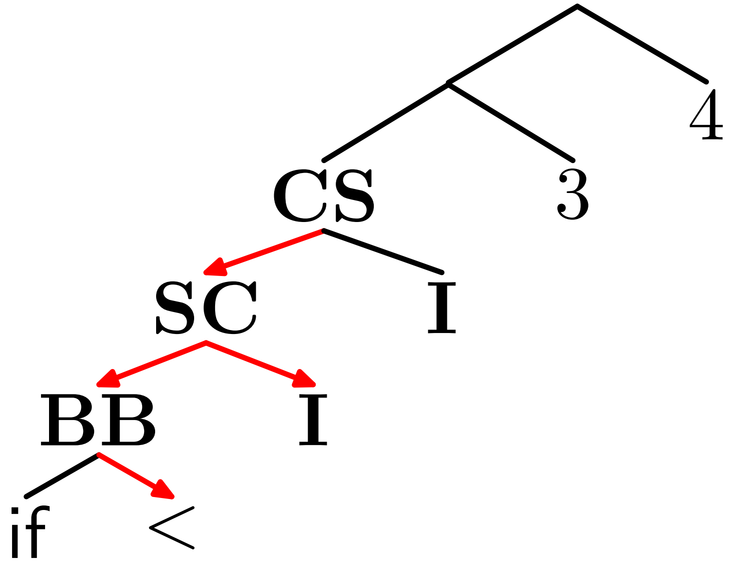
Classical: first-order combinators $\{\mathbf{B}, \mathbf{C}, \mathbf{S}, \mathbf{I}\}$

Complete basis, so can implement $\min$, but cumbersome

New: **higher-order combinators** $\{\mathbf{B}, \mathbf{C}, \mathbf{S}, \mathbf{I}\}^*$

Infinite basis, but resulting programs are more intuitive

e.g., $\mathbf{CS}$ routes 1st arg. left, 2nd arg. left and right



Programs with Multiple Arguments

Example: $(x, y) \mapsto \min(x, y)$

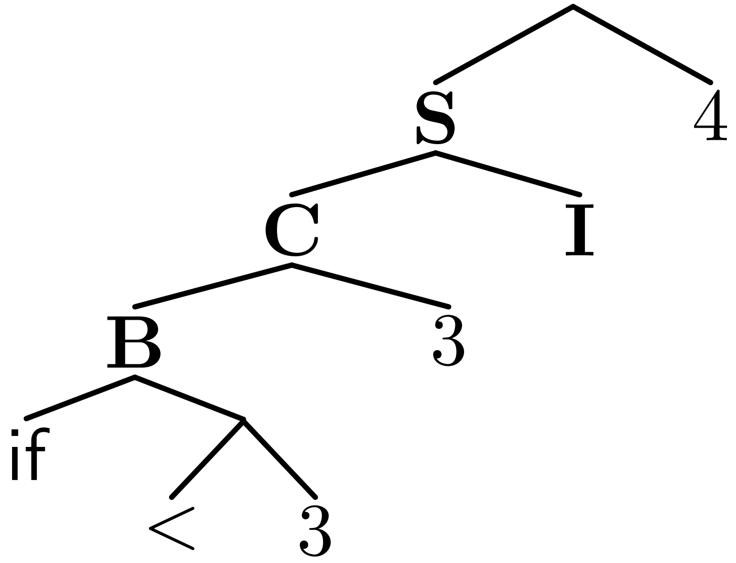
Classical: first-order combinators $\{\mathbf{B}, \mathbf{C}, \mathbf{S}, \mathbf{I}\}$

Complete basis, so can implement $\min$, but cumbersome

New: **higher-order combinators** $\{\mathbf{B}, \mathbf{C}, \mathbf{S}, \mathbf{I}\}^*$

Infinite basis, but resulting programs are more intuitive

e.g., $\mathbf{CS}$ routes 1st arg. left, 2nd arg. left and right



Programs with Multiple Arguments

Example: $(x, y) \mapsto \min(x, y)$

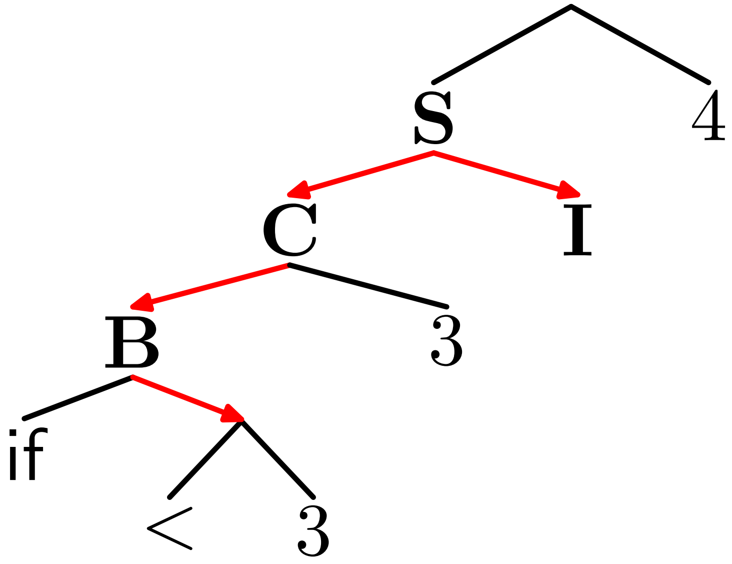
Classical: first-order combinators $\{\mathbf{B}, \mathbf{C}, \mathbf{S}, \mathbf{I}\}$

Complete basis, so can implement $\min$, but cumbersome

New: **higher-order combinators** $\{\mathbf{B}, \mathbf{C}, \mathbf{S}, \mathbf{I}\}^*$

Infinite basis, but resulting programs are more intuitive

e.g., $\mathbf{CS}$ routes 1st arg. left, 2nd arg. left and right



Programs with Multiple Arguments

Example: $(x, y) \mapsto \min(x, y)$

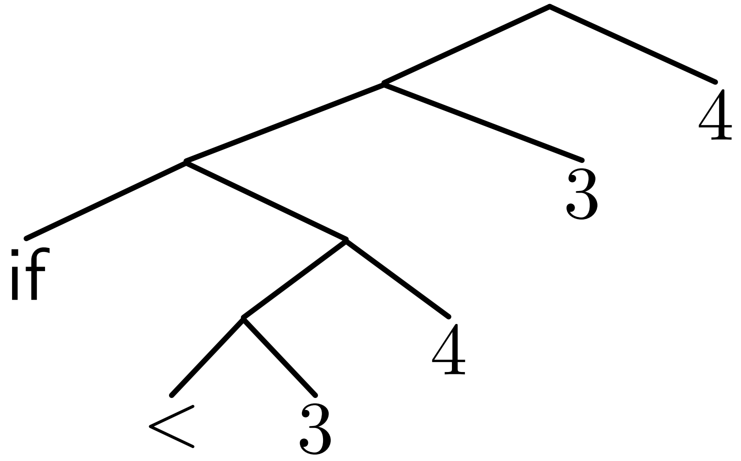
Classical: first-order combinators $\{\mathbf{B}, \mathbf{C}, \mathbf{S}, \mathbf{I}\}$

Complete basis, so can implement $\min$, but cumbersome

New: **higher-order combinators** $\{\mathbf{B}, \mathbf{C}, \mathbf{S}, \mathbf{I}\}^*$

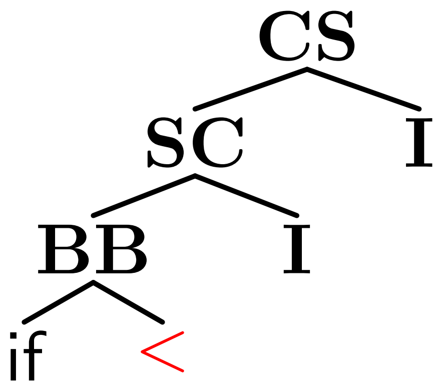
Infinite basis, but resulting programs are more intuitive

e.g., $\mathbf{CS}$ routes 1st arg. left, 2nd arg. left and right

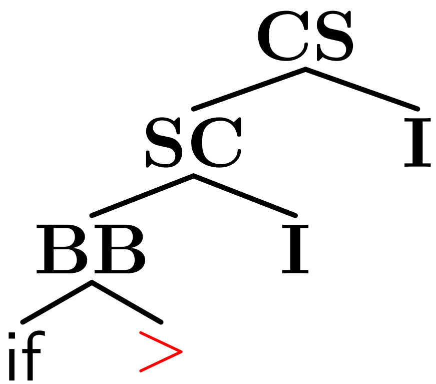


Using Combinators for Refactoring

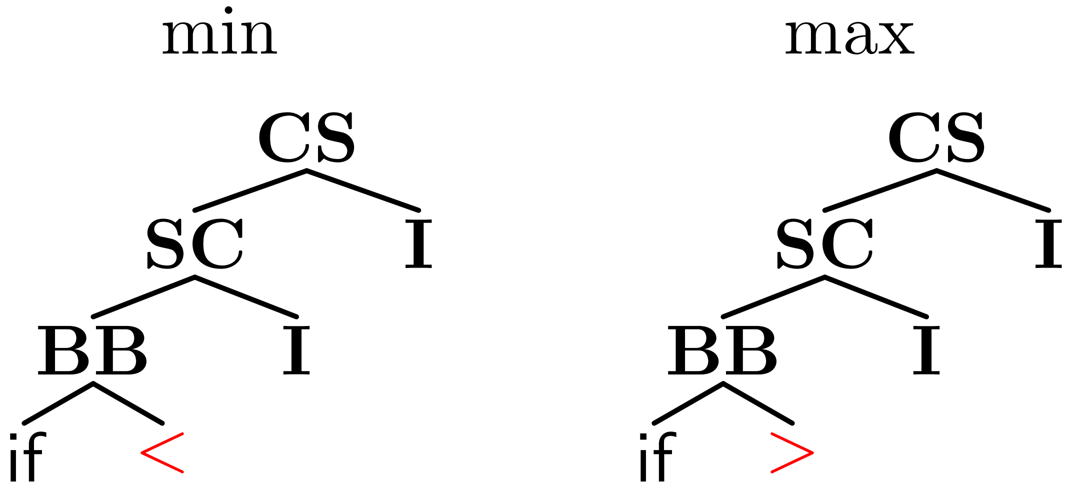
min



max

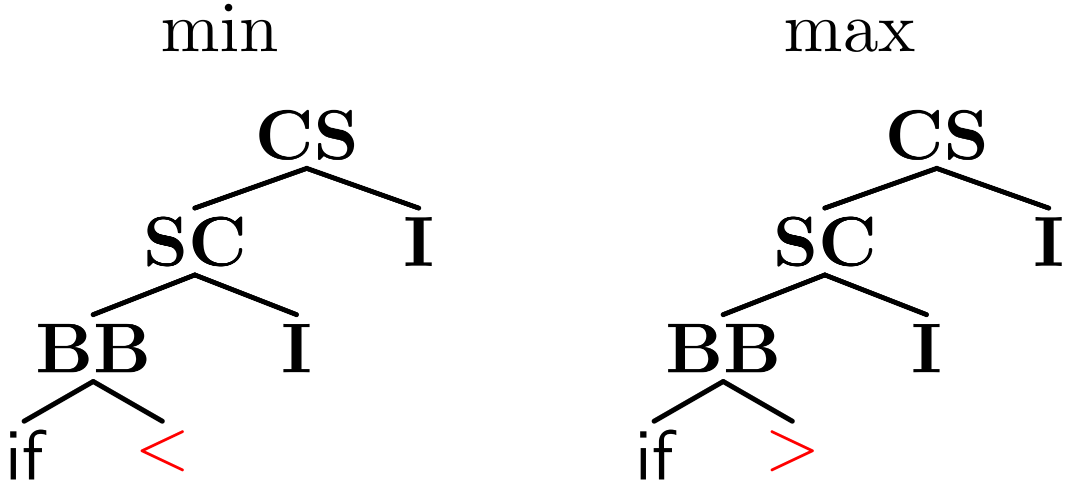


Using Combinators for Refactoring



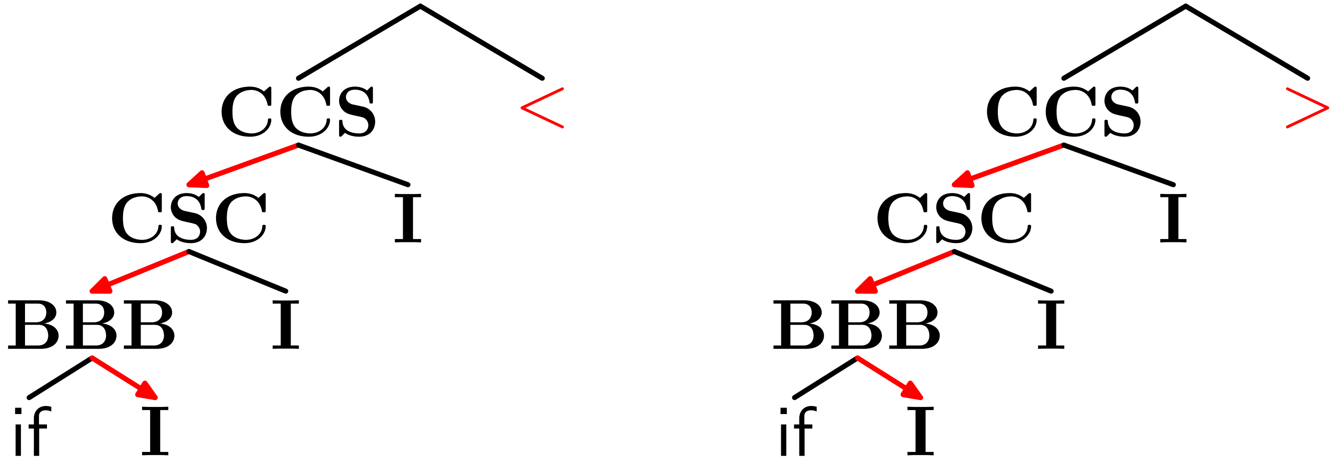
No significant sharing of subtrees (subprograms)

Using Combinators for Refactoring

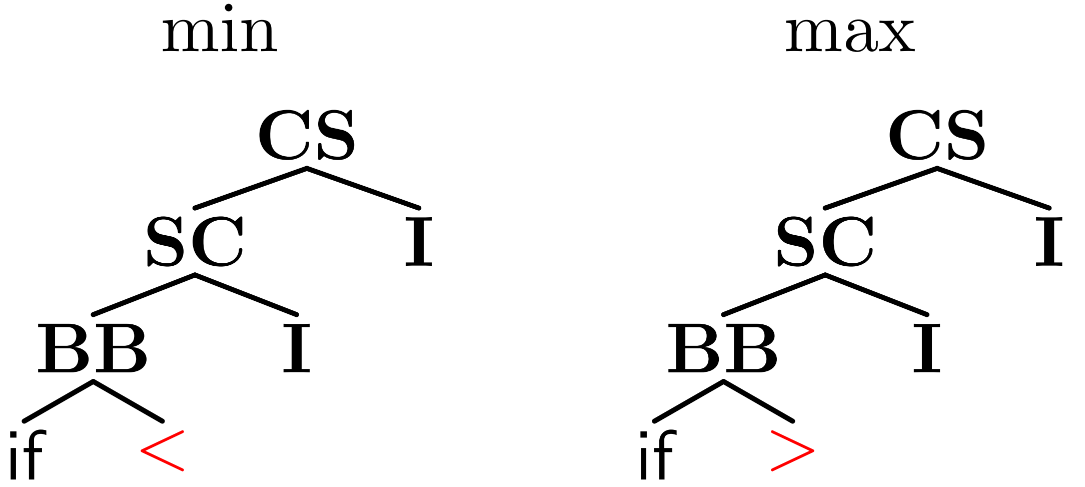


No significant sharing of subtrees (subprograms)

Refactored:

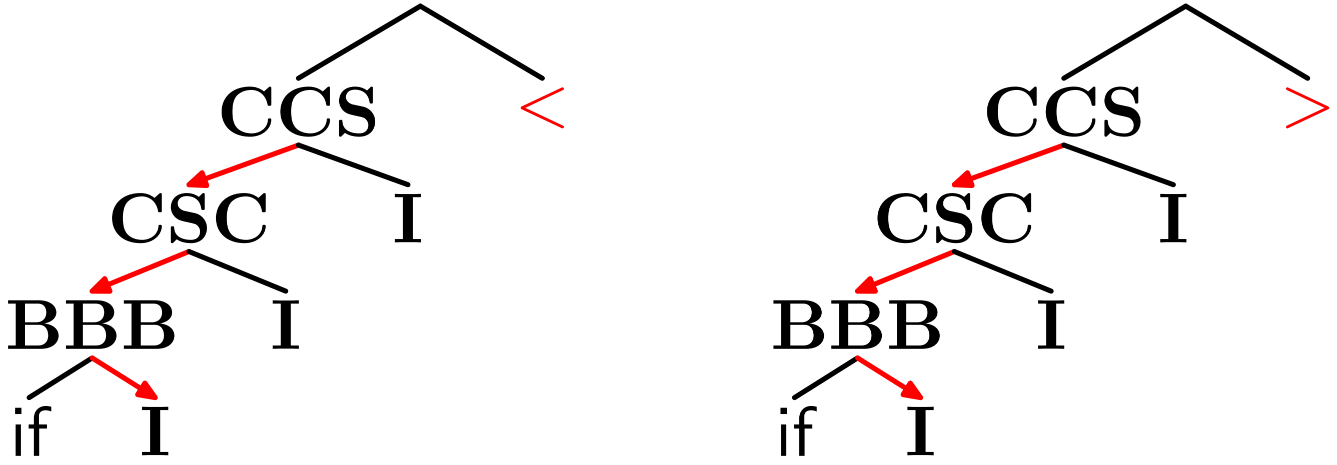


Using Combinators for Refactoring



No significant sharing of subtrees (subprograms)

Refactored:



Fruitful sharing of subtrees (subprograms)

Summary

Introduced new combinatory logic basis (intuition: routing)

Summary

Introduced new combinatory logic basis (intuition: routing)

Purpose of these combinators:

- Represent multi-argument functions
- Allow refactoring to expose common substructures

Summary

Introduced new combinatory logic basis (intuition: routing)

Purpose of these combinators:

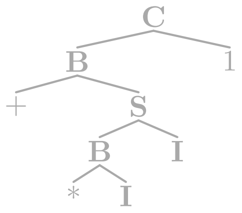
- Represent multi-argument functions
- Allow refactoring to expose common substructures

Achieved uniformity: Every subtree is a subprogram

Outline of Proposed Solution

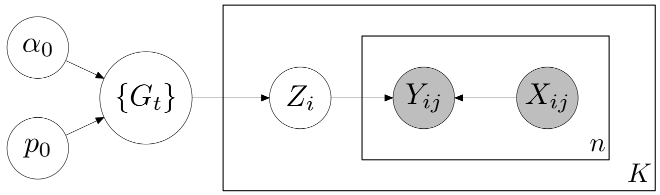
Program representation: What are subprograms?

Combinatory logic



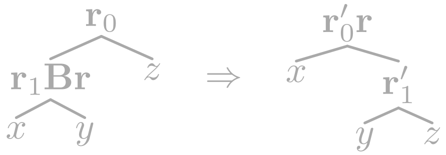
Probabilistic model: Which programs are favorable?

Nonparametric Bayes



Statistical inference: How do we search for good programs?

MCMC



Probabilistic Context-Free Grammars

$\textsc{GenIndep}(t)$: [returns a combinator of type t]

Probabilistic Context-Free Grammars

```
GenIndep($t$): [returns a combinator of type $t$]
  With probability $\lambda_0$:
```

Probabilistic Context-Free Grammars

```
GenIndep($t$): [returns a combinator of type $t$]
  With probability $\lambda_0$:
    Return a random primitive combinator (e.g., $+$, $3$, $\mathbf{I}$)
```

Probabilistic Context-Free Grammars

```
$\textsc{GenIndep}(t)$: [returns a combinator of type $t$]
  With probability $\lambda_0$:
    Return a random primitive combinator (e.g., $+$, $3$, $\mathbf{I}$)
  Else:
    Choose a type $s$
    $x \leftarrow \textsc{GenIndep}(s \rightarrow t)$
```

Probabilistic Context-Free Grammars

```
$\textsc{GenIndep}(t)$: [returns a combinator of type $t$]
  With probability $\lambda_0$:
    Return a random primitive combinator (e.g., $+$, $3$, $\mathbf{I}$)
  Else:
    Choose a type $s$
    $x \leftarrow \textsc{GenIndep}(s \to t)$
    $y \leftarrow \textsc{GenIndep}(s)$
```

Probabilistic Context-Free Grammars

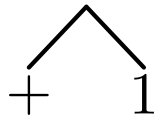
```
$\textsc{GenIndep}(t)$: [returns a combinator of type $t$]
  With probability $\lambda_0$:
    Return a random primitive combinator (e.g., $+$, $3$, $\mathbf{I}$)
  Else:
    Choose a type $s$
    $x \leftarrow \textsc{GenIndep}(s \rightarrow t)$
    $y \leftarrow \textsc{GenIndep}(s)$
    return $(x, y)$
```

Probabilistic Context-Free Grammars

```
GenIndep($t$): [returns a combinator of type $t$]
  With probability $\lambda_0$:
    Return a random primitive combinator (e.g., $+$, $3$, $\mathbf{I}$)
  Else:
    Choose a type $s$
    $x \leftarrow \textsc{GenIndep}(s \to t)$
    $y \leftarrow \textsc{GenIndep}(s)$
    return $(x, y)$
```

Example:

$\textsc{GenIndep}(\text{int} \to \text{int}) \quad \Longrightarrow$

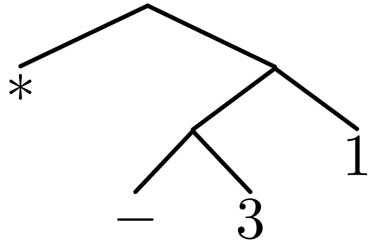


Probabilistic Context-Free Grammars

```
$\textsc{GenIndep}(t)$: [returns a combinator of type $t$]
  With probability $\lambda_0$:
    Return a random primitive combinator (e.g., $+$, $3$, $\mathbf{I}$)
  Else:
    Choose a type $s$
    $x \leftarrow \textsc{GenIndep}(s \rightarrow t)$
    $y \leftarrow \textsc{GenIndep}(s)$
    return $(x, y)$
```

Example:

$\textsc{GenIndep}(\text{int} \rightarrow \text{int}) \quad \Longrightarrow$

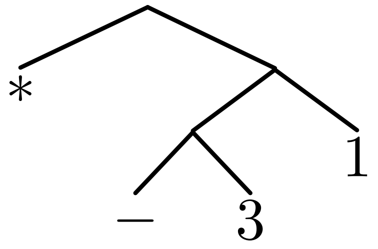


Probabilistic Context-Free Grammars

```
$\textsc{GenIndep}(t)$: [returns a combinator of type $t$]
  With probability $\lambda_0$:
    Return a random primitive combinator (e.g., $+$, $3$, $\mathbf{I}$)
  Else:
    Choose a type $s$
    $x \leftarrow \textsc{GenIndep}(s \rightarrow t)$
    $y \leftarrow \textsc{GenIndep}(s)$
    return $(x, y)$
```

Example:

$\textsc{GenIndep}(\text{int} \rightarrow \text{int}) \quad \Longrightarrow$



Problem: No encouragement to share subprograms

Adaptor Grammars [Johnson, 2007]

$C_t \leftarrow [\,]$ for each type t [cached list of combinators]

Adaptor Grammars [Johnson, 2007]

$C_t \leftarrow [\,]$ for each type t [cached list of combinators]
(notation: return* c adds c to C_t and returns c)

Adaptor Grammars [Johnson, 2007]

$C_t \leftarrow [\,]$ for each type t [cached list of combinators]
  (notation: return* c adds c to C_t and returns c)

```
$\textsc{GenCache}(t)$: [returns a combinator of type $t$]
  With probability $\frac{\alpha_0 + N_t d}{\alpha_0 + |C_t|}$:
```

Adaptor Grammars [Johnson, 2007]

$C_t \leftarrow [\,]$ for each type t [cached list of combinators]
(notation: return* c adds c to C_t and returns c)

```
GenCache($t$): [returns a combinator of type $t$]
  With probability $\frac{\alpha_0 + N_t d}{\alpha_0 + |C_t|}$:
    With probability $\lambda_0$:
      Return* a random primitive combinator (e.g., $+$, $3$, $\mathbf{I}$)
    Else:
      Choose a type $s$
      $x \leftarrow$ GenCache($s \rightarrow t$)
      $y \leftarrow$ GenCache($s$)
      Return* $(x, y)$
  Else:
```

Adaptor Grammars [Johnson, 2007]

$C_t \leftarrow [\,]$ for each type t [cached list of combinators]

(notation: return* c adds c to C_t and returns c)

```
$\textsc{GenCache}(t)$: [returns a combinator of type $t$]
  With probability $\frac{\alpha_0 + N_t d}{\alpha_0 + |C_t|}$:
    With probability $\lambda_0$:
      Return* a random primitive combinator (e.g., $+$, $3$, $\mathbf{I}$)
    Else:
      Choose a type $s$
      $x \leftarrow \textsc{GenCache}(s \to t)$
      $y \leftarrow \textsc{GenCache}(s)$
      Return* $(x, y)$
  Else:
    Return* $z \in C_t$ with probability $\frac{M_z - d}{|C_t| - N_t d}$
```

Adaptor Grammars [Johnson, 2007]

$C_t \leftarrow [\,]$ for each type t [cached list of combinators]
(notation: return* c adds c to C_t and returns c)

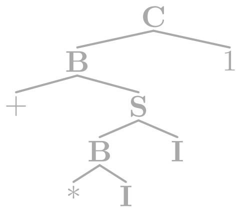
```
GenCache($t$): [returns a combinator of type $t$]
  With probability $\frac{\alpha_0 + N_t d}{\alpha_0 + |C_t|}$:
    With probability $\lambda_0$:
      Return* a random primitive combinator (e.g., $+$, $3$, $\mathbf{I}$)
    Else:
      Choose a type $s$
      $x \leftarrow \textsc{GenCache}(s \to t)$
      $y \leftarrow \textsc{GenCache}(s)$
      Return* $(x, y)$
  Else:
    Return* $z \in C_t$ with probability $\frac{M_z - d}{|C_t| - N_t d}$
```

Interpretation of cache C_t: library of generally useful (unnamed) subroutines which are reused.

Outline of Proposed Solution

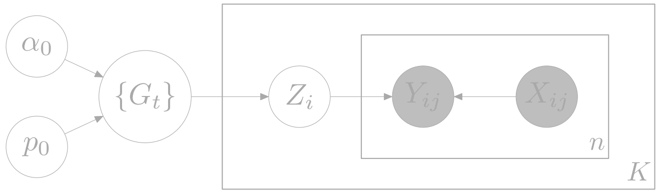
Program representation: What are subprograms?

Combinatory logic



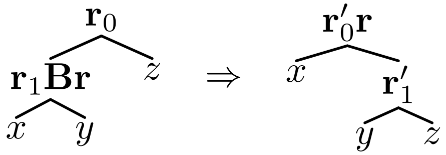
Probabilistic model: Which programs are favorable?

Nonparametric Bayes



Statistical inference: How do we search for good programs?

MCMC



Inference via MCMC

User provides tree structure that encodes set of programs U

Objective: sample from posterior given program in U

Inference via MCMC

User provides tree structure that encodes set of programs U

Objective: sample from posterior given program in U

Use Metropolis-Hastings

- Proposal: sample a random program transformation

Inference via MCMC

User provides tree structure that encodes set of programs U

Objective: sample from posterior given program in U

Use Metropolis-Hastings

  Proposal: sample a random program transformation

Program transformations maintain invariant that
  program is correct (likelihood is 1)

Inference via MCMC

User provides tree structure that encodes set of programs U

Objective: sample from posterior given program in U

Use Metropolis-Hastings

Proposal: sample a random program transformation

Program transformations maintain invariant that program is correct (likelihood is 1)

Two types of transformations:

1. Switching
2. Refactoring

Program transformations (MCMC moves)

Switching: Change content, preserve empirical semantics

Program transformations (MCMC moves)

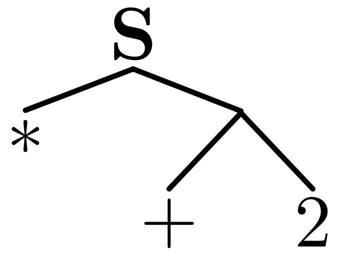
Switching: Change content, preserve empirical semantics

Data: $\{(2, 8)\}$

Program transformations (MCMC moves)

Switching: Change content, preserve empirical semantics

Data: $\{(2,8)\}$

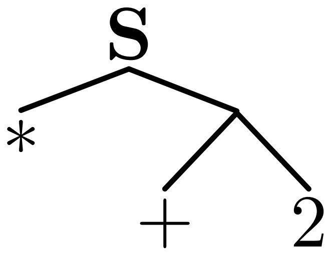


$[x \mapsto x(x+2)]$

Program transformations (MCMC moves)

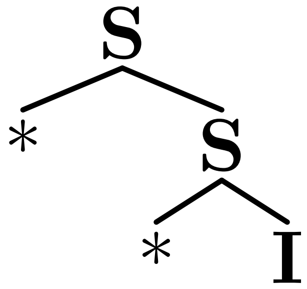
Switching: Change content, preserve empirical semantics

Data: $\{(2, 8)\}$



$[x \mapsto x(x + 2)]$

$\Leftrightarrow$

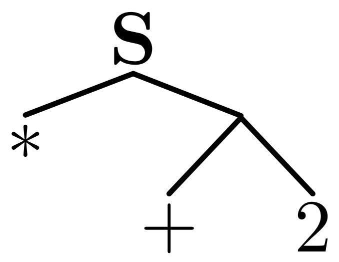


$[x \mapsto x^3]$

Program transformations (MCMC moves)

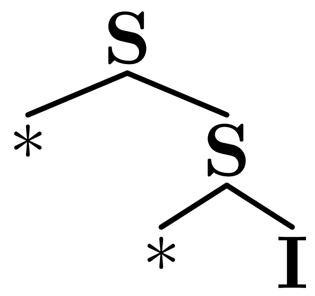
Switching: Change content, preserve empirical semantics

Data: $\{(2,8)\}$



$[x \mapsto x(x+2)]$

$\Leftrightarrow$



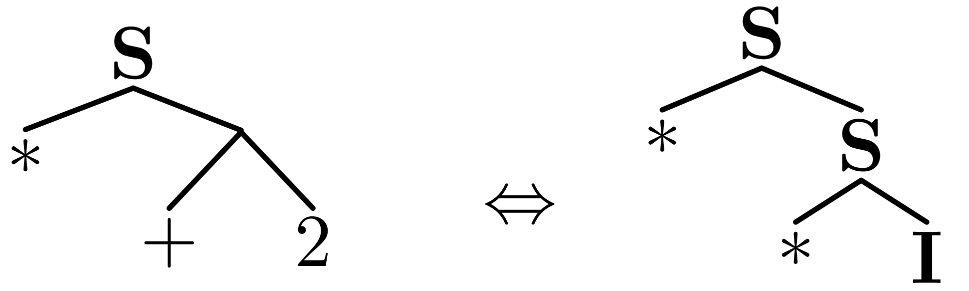
$[x \mapsto x^3]$

Purpose: change generalization

Program transformations (MCMC moves)

Switching: Change content, preserve empirical semantics

Data: $\{(2, 8)\}$



$[x \mapsto x(x+2)]$ $[x \mapsto x^3]$

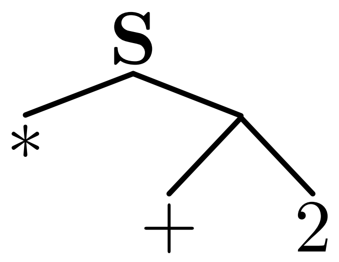
Purpose: change generalization

Refactoring: Change form, preserve total semantics

Program transformations (MCMC moves)

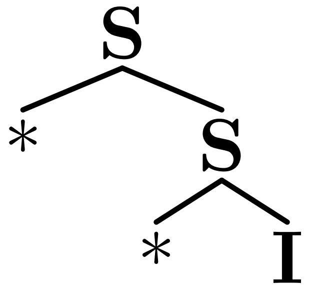
Switching: Change content, preserve empirical semantics

Data: $\{(2, 8)\}$



$[x \mapsto x(x+2)]$

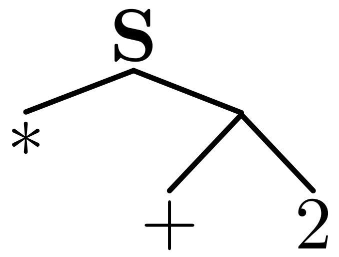
$\Leftrightarrow$



$[x \mapsto x^3]$

Purpose: change generalization

Refactoring: Change form, preserve total semantics

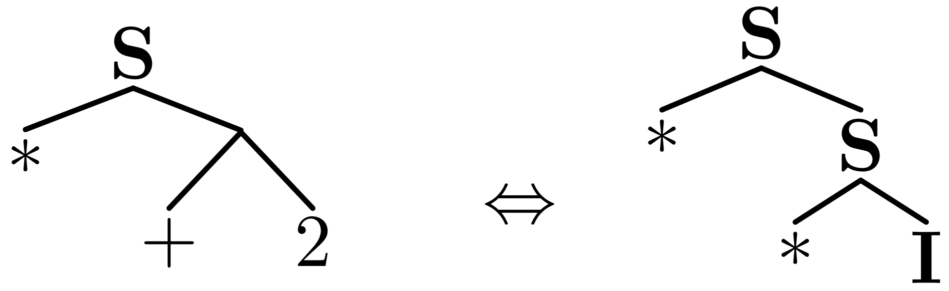


$[x \mapsto x(x+2)]$

Program transformations (MCMC moves)

Switching: Change content, preserve empirical semantics

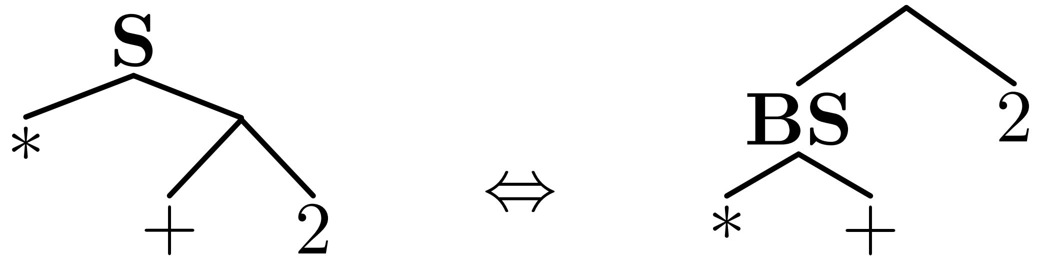
Data: $\{(2, 8)\}$



$[x \mapsto x(x+2)]$ $\quad$ $[x \mapsto x^3]$

Purpose: change generalization

Refactoring: Change form, preserve total semantics

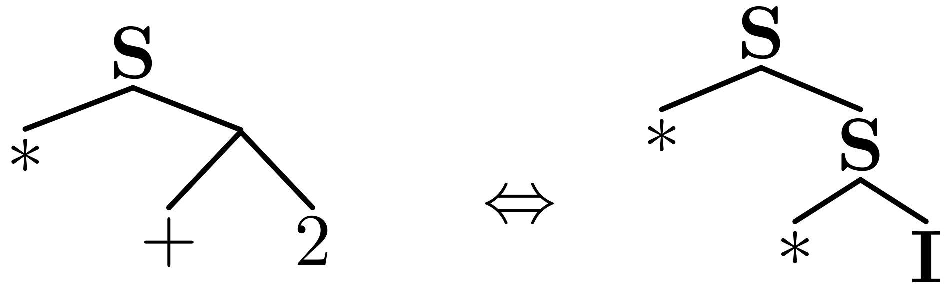


$[x \mapsto x(x+2)]$ $\quad$ $[x \mapsto x(x+2)]$

Program transformations (MCMC moves)

Switching: Change content, preserve empirical semantics

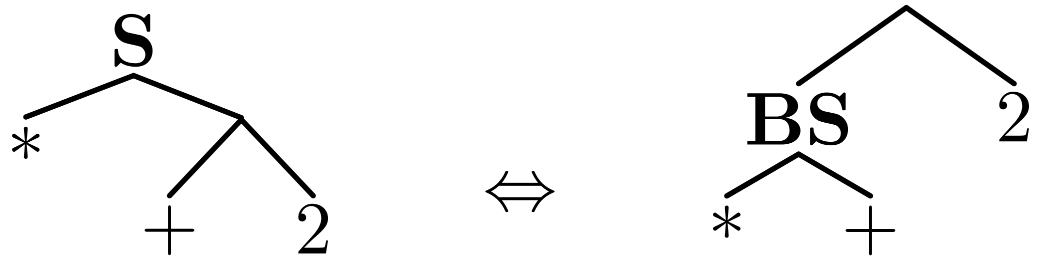
Data: $\{(2, 8)\}$



$[x \mapsto x(x+2)]$ $\quad$ $[x \mapsto x^3]$

Purpose: change generalization

Refactoring: Change form, preserve total semantics



$[x \mapsto x(x+2)]$ $\quad$ $[x \mapsto x(x+2)]$

Purpose: expose different subprograms for sharing

Text Editing Experiments

Setup:

Dataset of [Lau et al., 2003]

$K = 24$ tasks

Each task: train on 2–5 examples, test on $\approxeq 13$ examples

10 random trials

Text Editing Experiments

Setup:

Dataset of [Lau et al., 2003]
$K = 24$ tasks
Each task: train on 2–5 examples, test on $\approxeq 13$ examples
10 random trials

Example task:

Cardinals 5, Pirates 2.

$$\Downarrow$$

GameScore[winner 'Cardinals'; loser 'Pirates'; scores [5, 2]].

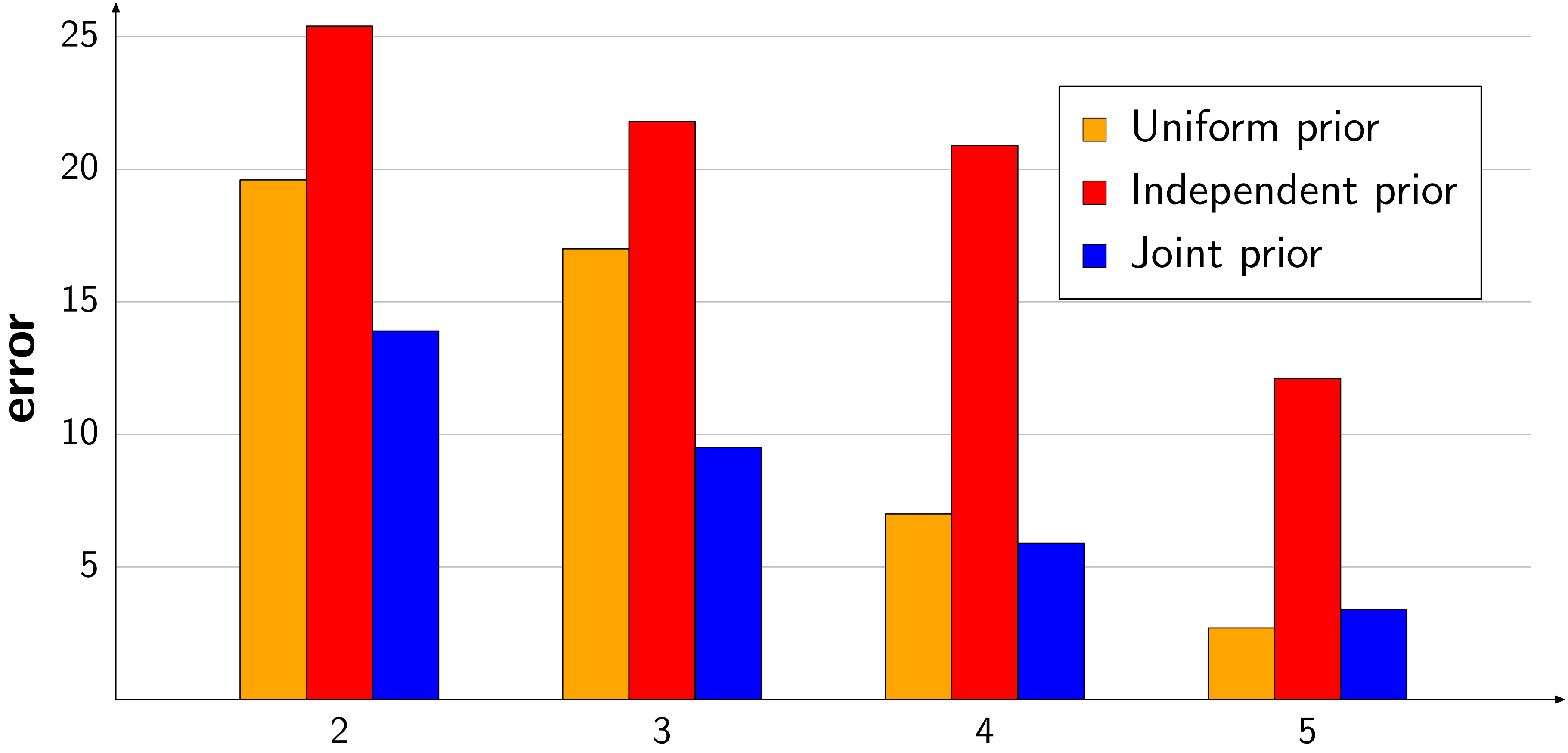
Experimental Results

🟧 Uniform prior

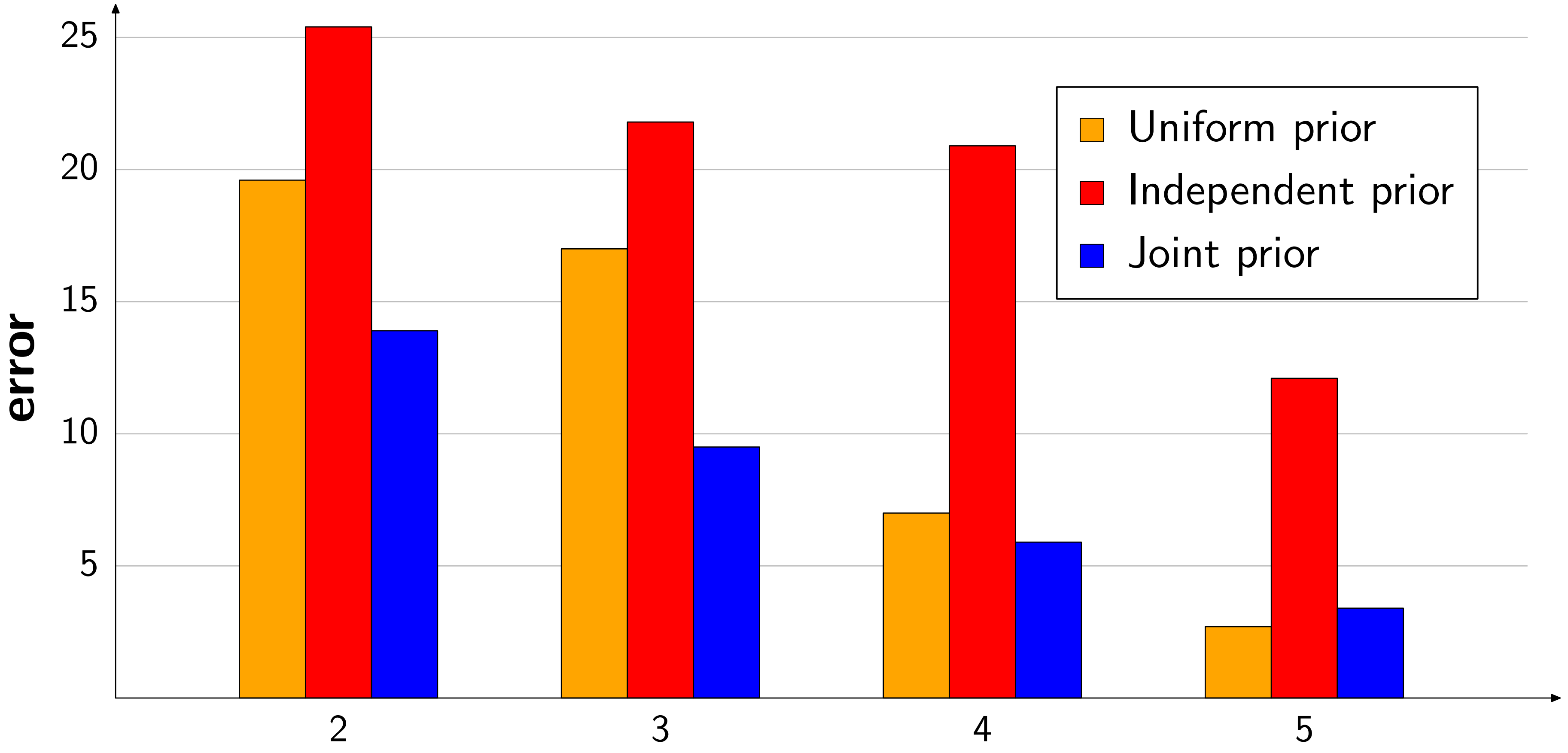
🟥 Independent prior

🟦 Joint prior

Experimental Results



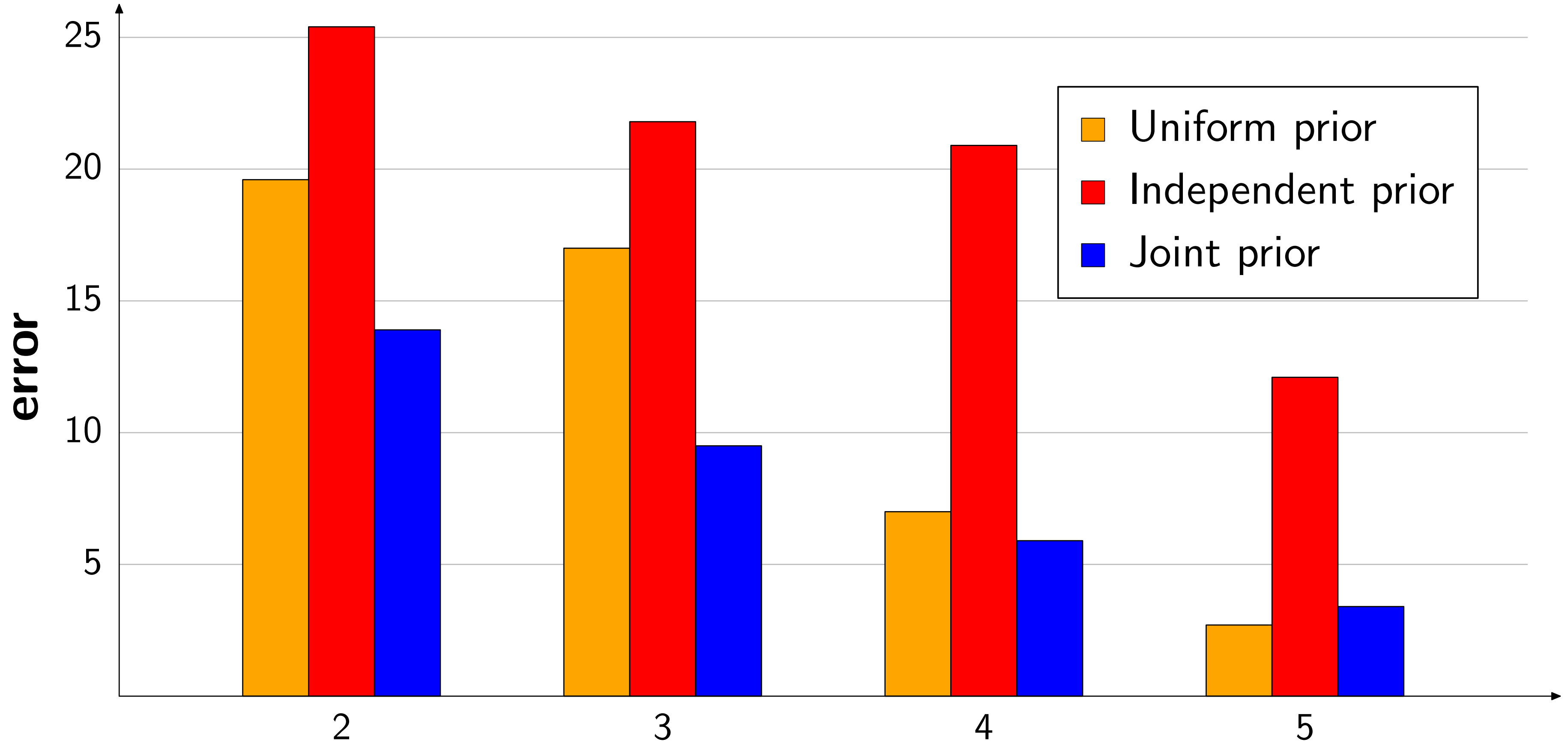
Experimental Results



Observations:

- Independent prior is even worse than uniform prior

Experimental Results



Observations:

- Independent prior is even worse than uniform prior
- Joint prior (multi-task learning) is effective

Summary

$$X \Rightarrow \boxed{} \Rightarrow Y$$

Summary

$$X \Rightarrow \boxed{\text{program}} \Rightarrow Y$$

Summary

$$X \Rightarrow \boxed{\text{program}} \Rightarrow Y$$

Key challenge: learn programs from few examples

Summary

$$X \Rightarrow \boxed{\text{program}} \Rightarrow Y$$

Key challenge: learn programs from few examples

Main idea: share subprograms across multiple tasks

Summary

$$X \Rightarrow \boxed{\text{program}} \Rightarrow Y$$

Key challenge: learn programs from few examples

Main idea: share subprograms across multiple tasks

Tools:

- Combinatory logic: expose subprograms to be shared
- Adaptor grammars: encourage sharing of subprograms
- Metropolis-Hastings: proposals are program transformations