Bytecode Interpreters

Concepts of Programming Languages

CAS CS 320

Outline

- Course evaluations
- Discuss **stack-based languages** and **stack machines**
- Look briefly at how to compile **variables** and **functions**
- Finish up the dang course

Course evaluations

Overview

Abstract/Virtual Machines

Abstract/Virtual Machines

A **virtual machine** is a computational abstraction, like a Turing machine (but usually *easier to implement*)

Abstract/Virtual Machines

A **virtual machine** is a computational abstraction, like a Turing machine (but usually *easier to implement*)

Virtual machines are typically implemented as **bytecode interpreters**, where "programs" are streams of bytes and a command is represented as a byte (plus possibly some extra data)

Abstract/Virtual Machines

A **virtual machine** is a computational abstraction, like a Turing machine (but usually *easier to implement*)

Virtual machines are typically implemented as **bytecode interpreters**, where "programs" are streams of bytes and a command is represented as a byte (plus possibly some extra data)

One common abstraction is **stack machines**

Benefits of Virtual (Stack) Machines

state program

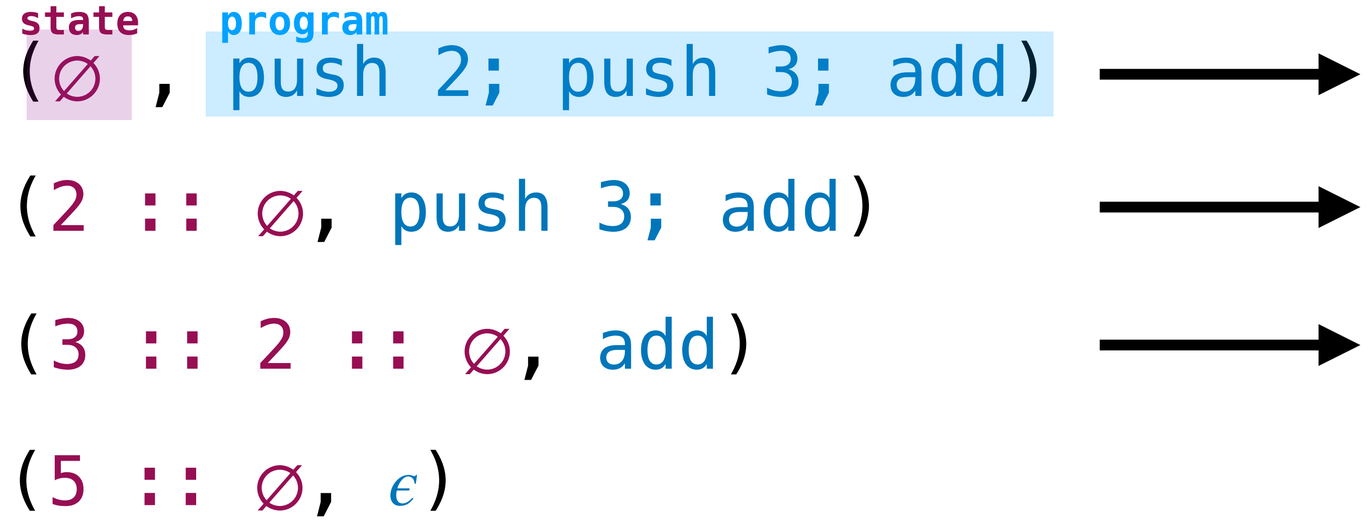
(∅ , push 2; push 3; add) ⟶

(2 :: ∅, push 3; add) ⟶

(3 :: 2 :: ∅, add) ⟶

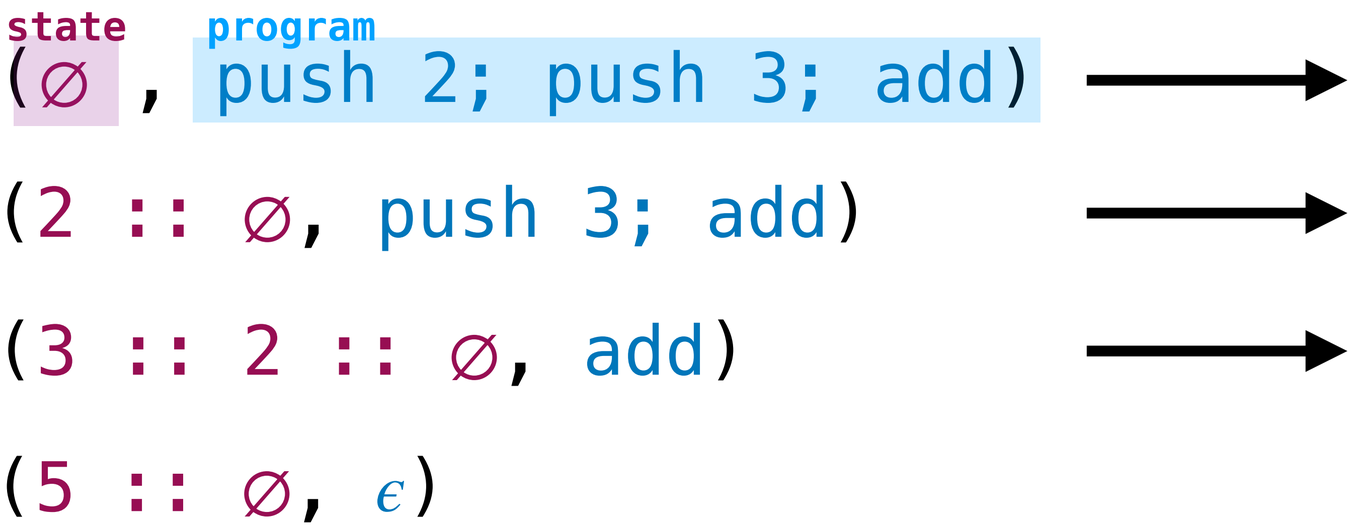
(5 :: ∅, ϵ)

Benefits of Virtual (Stack) Machines



Simplicity: Stacks aren't too complicated

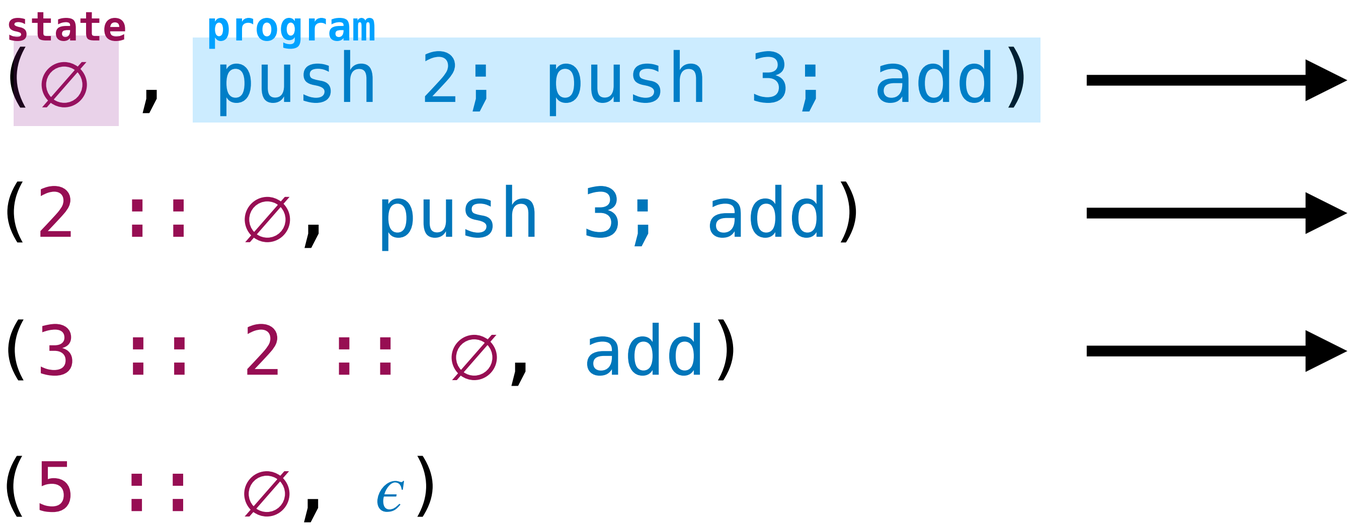
Benefits of Virtual (Stack) Machines



Simplicity: Stacks aren't too complicated

Portability: Any OS should be able to handle a stream of bytes, so the machine dependent part of our programming language can be simplified

Benefits of Virtual (Stack) Machines

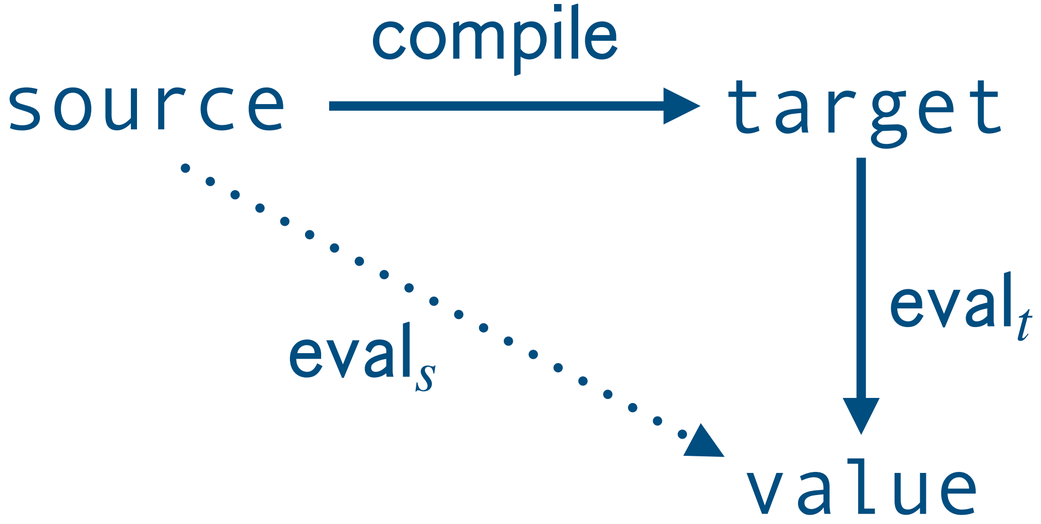


Simplicity: Stacks aren't too complicated

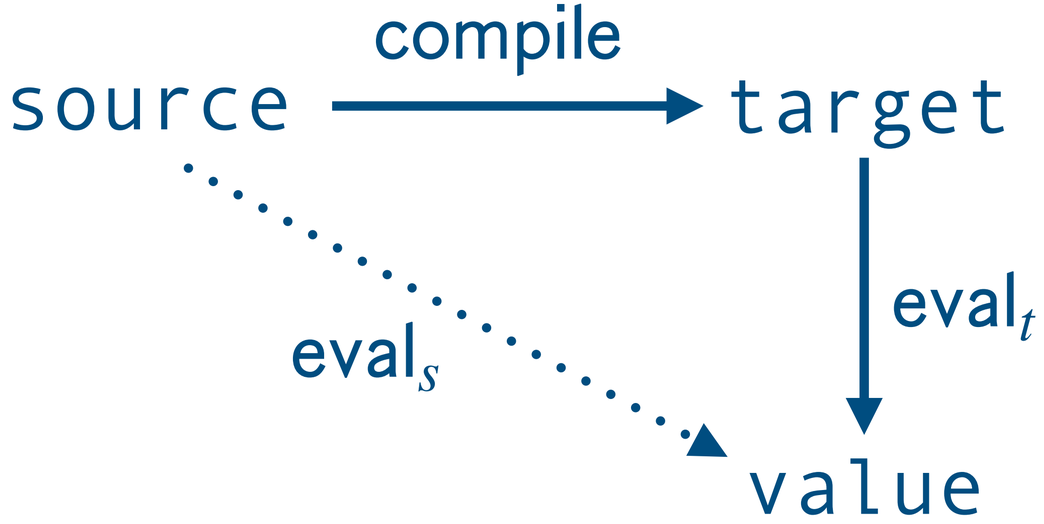
Portability: Any OS should be able to handle a stream of bytes, so the machine dependent part of our programming language can be simplified

Efficiency (sort of): They can be implemented in low-level languages, and so will generally be faster than the interpreters we build in this course (though not as fast as natively compiled code)

Compilation (High Level)

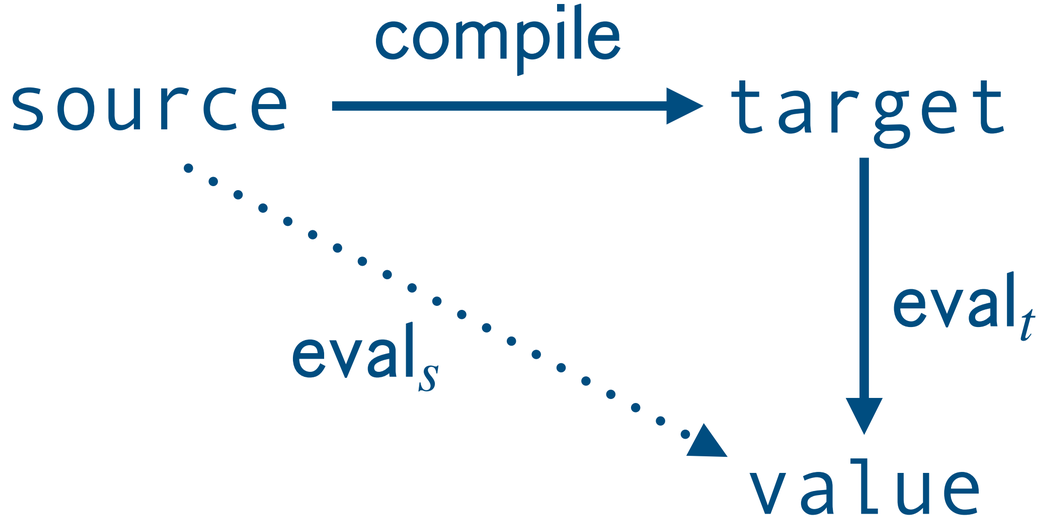


Compilation (High Level)



Compilation is the process of translating a program in one language to another, maintaining semantic behavior

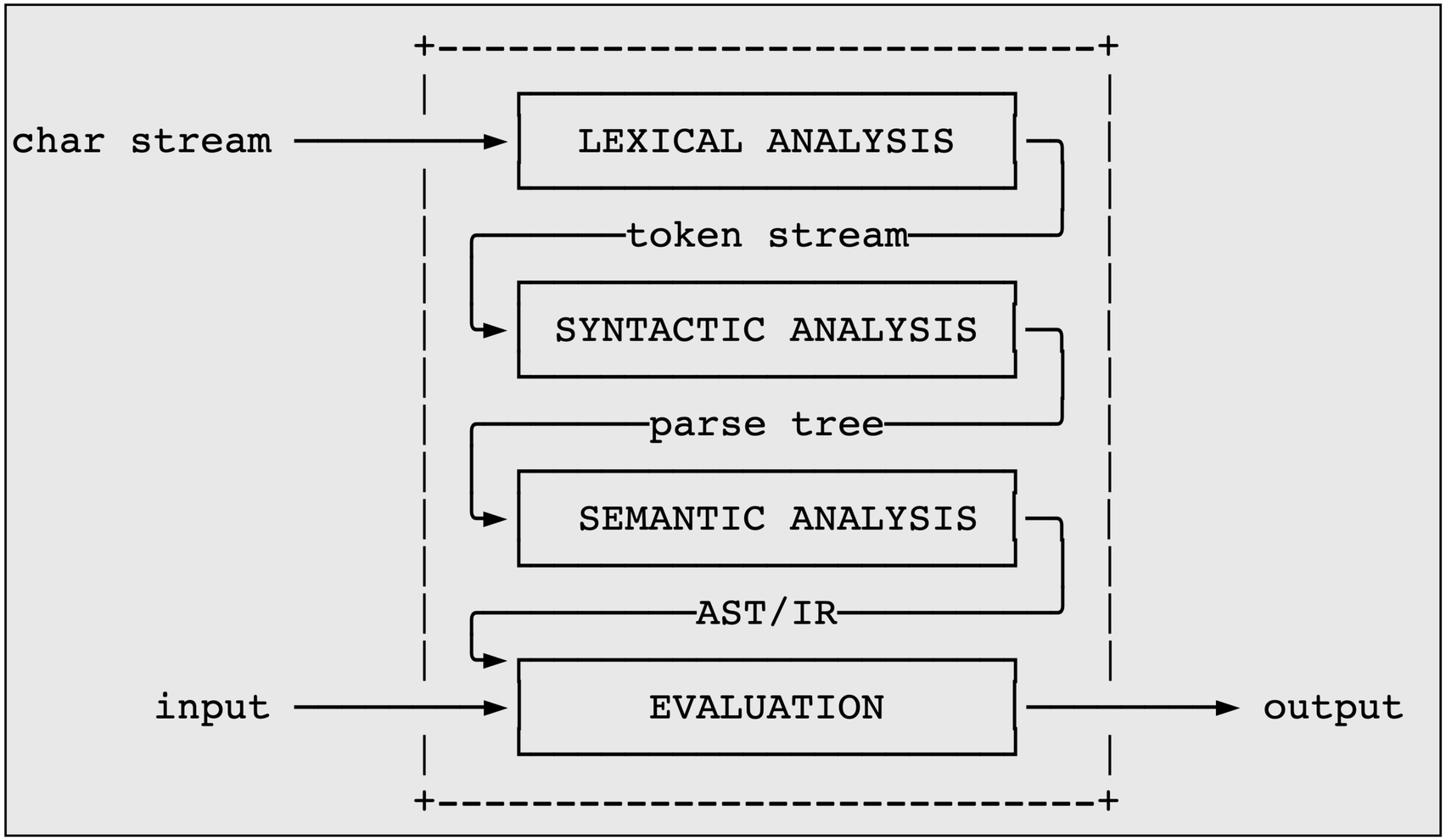
Compilation (High Level)



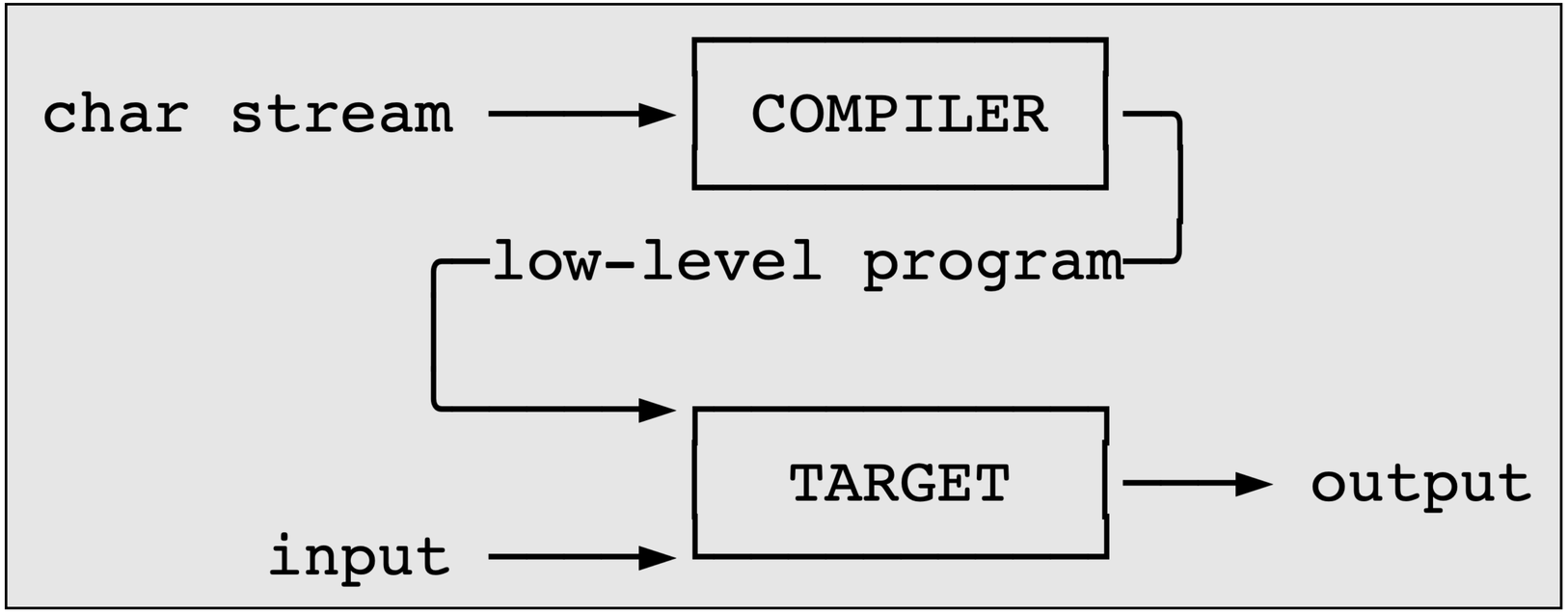
Compilation is the process of translating a program in one language to another, maintaining semantic behavior

Compilation can be a part of interpretation as well, like with **bytecode interpretation**

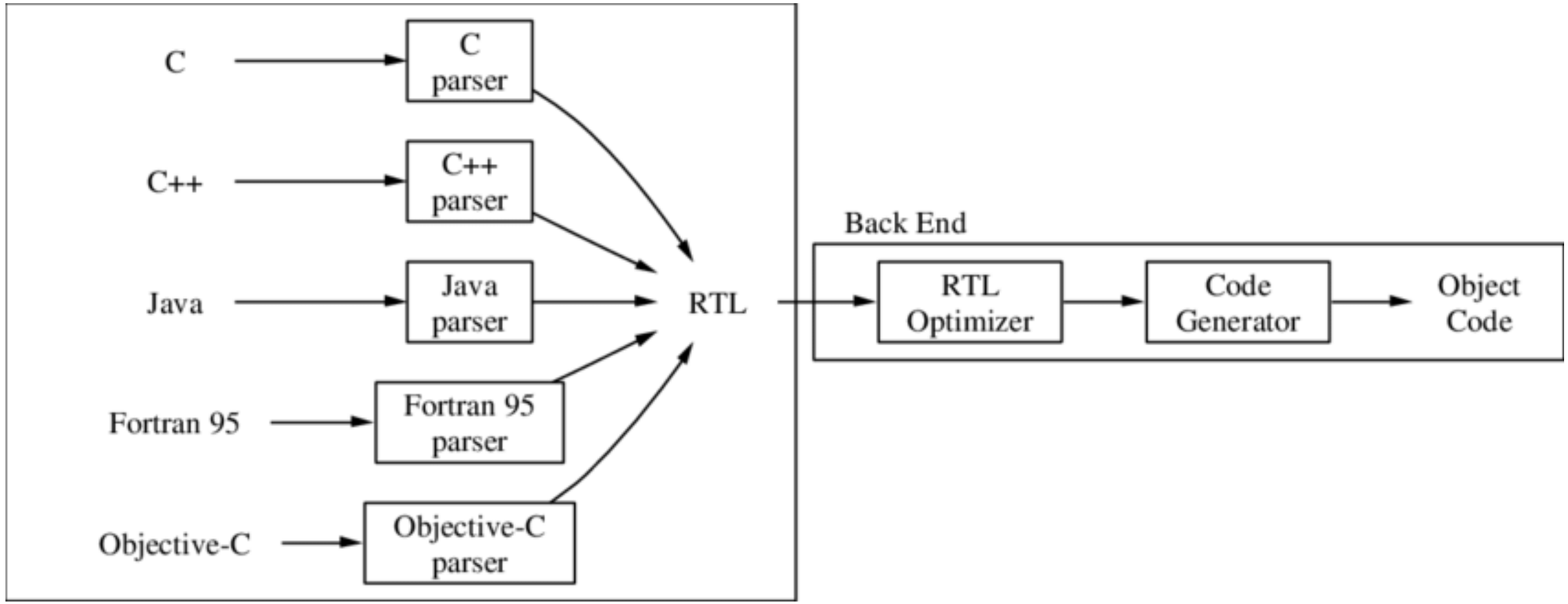
Recall: The Interpretation Pipeline



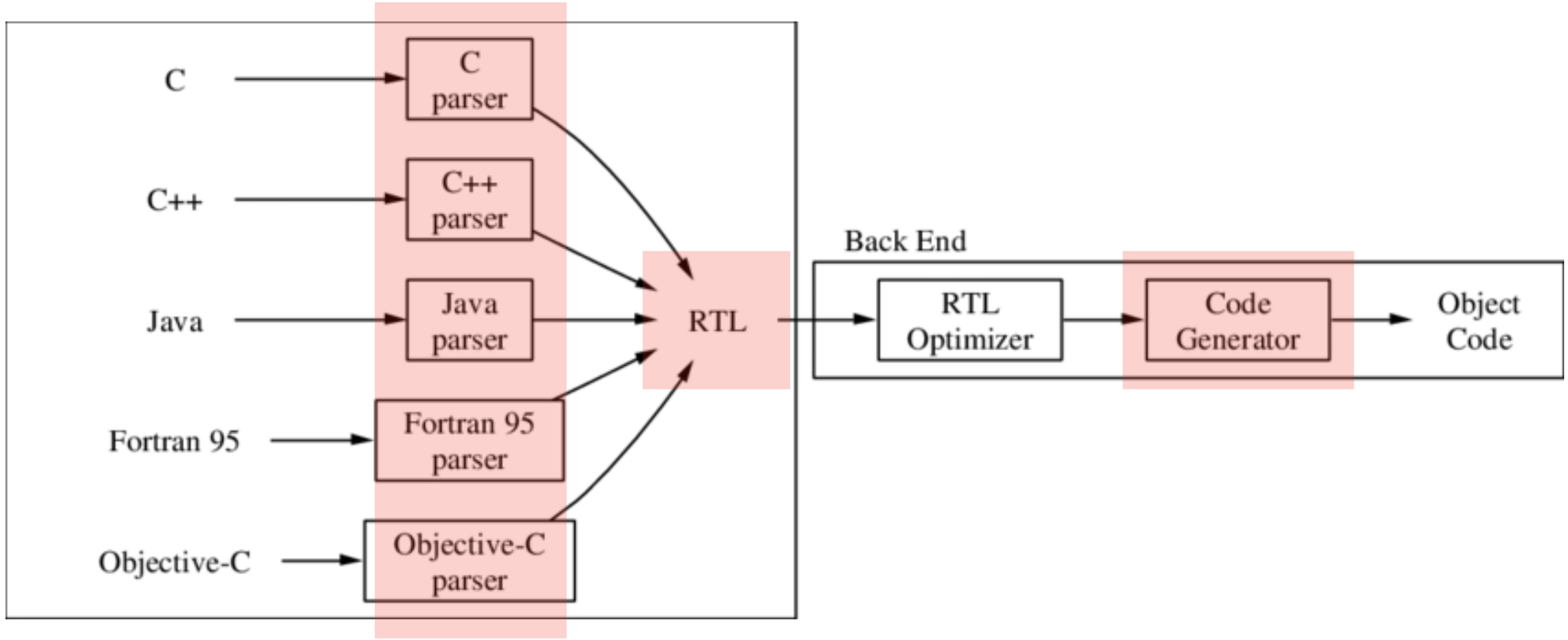
The Compiler Pipeline



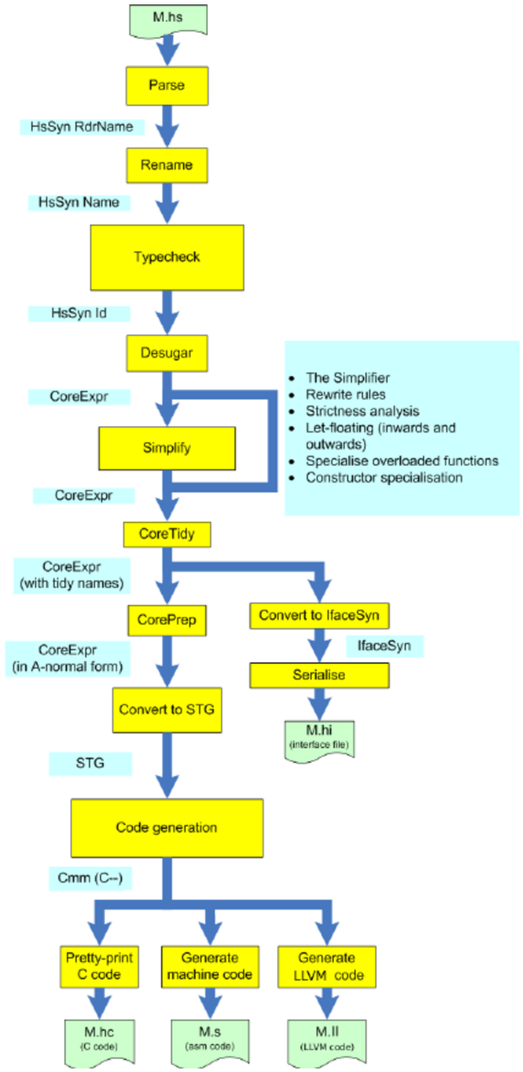
Example Pipelines: gcc



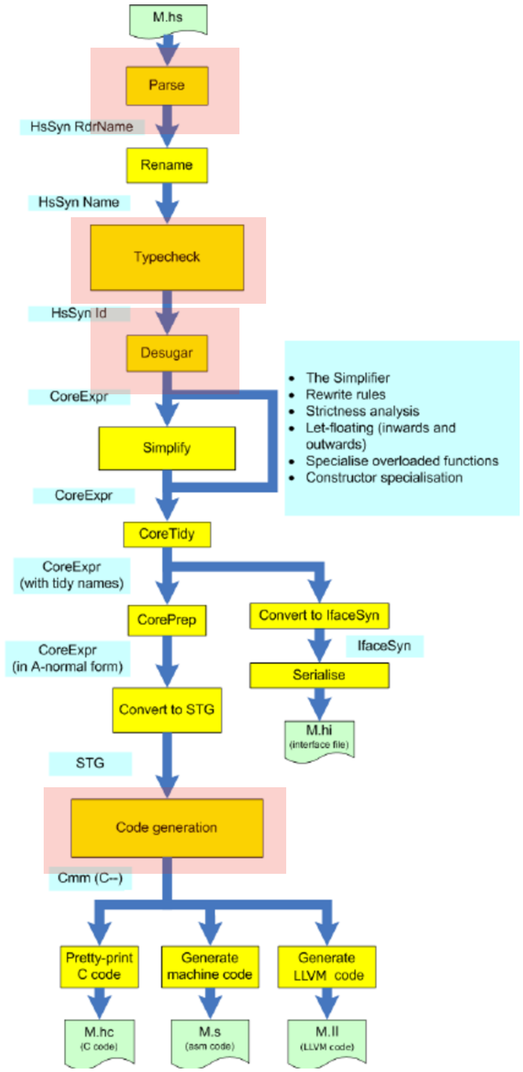
Example Pipelines: gcc



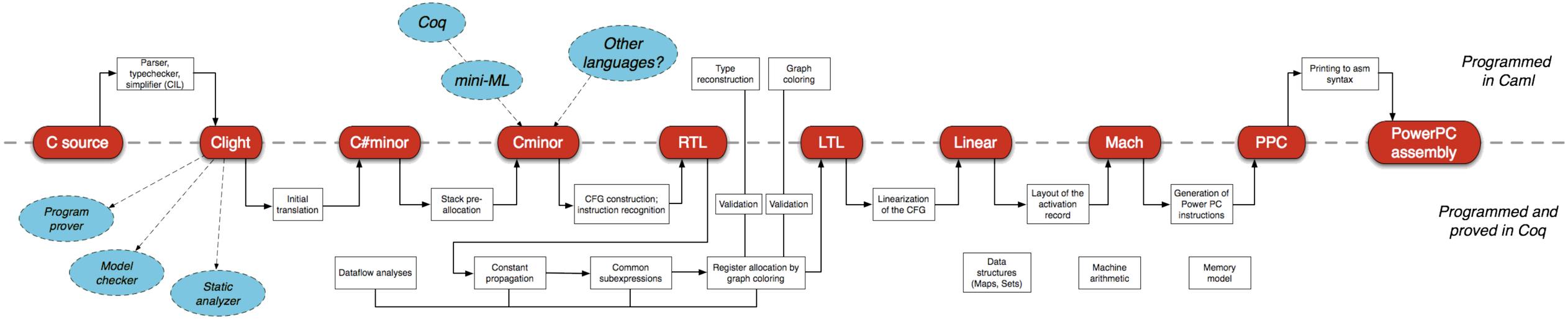
Example Pipelines: GHC (Haskell)



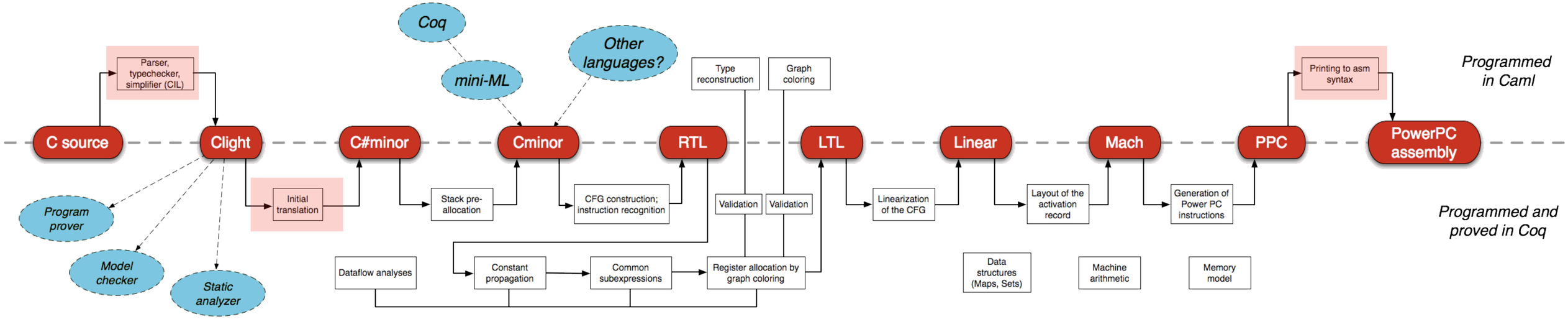
Example Pipelines: GHC (Haskell)



Example Pipelines: CompCert (C)



Example Pipelines: CompCert (C)



Stack-Based Arithmetic

Stack-Based Arithmetic (Syntax)

$$
\begin{aligned}
\texttt{<prog>} &::= \{\texttt{<com>}\} \\
\texttt{<com>} &::= \texttt{ADD} \mid \texttt{SUB} \mid \texttt{MUL} \mid \texttt{DIV} \mid \texttt{PUSH}\ \texttt{<num>} \\
\texttt{<num>} &::= \mathbb{Z}
\end{aligned}
$$

Stack-Based Arithmetic (Semantics)

$$\langle \mathcal{S}, P \rangle$$

A **value** is an integer ($\mathbb{Z}$)

A **configuration** is made up of a stack ($\mathcal{S}$) of values and a program (P) given by **<prog>**

Stack-Based Arithmetic (Semantics)

$$\frac{}{\langle m :: n :: \mathcal{S}, \texttt{ADD}\ P\rangle \longrightarrow \langle (m+n) :: \mathcal{S}, P\rangle}\,(\texttt{add})$$

$$\frac{}{\langle m :: n :: \mathcal{S}, \texttt{SUB}\ P\rangle \longrightarrow \langle (m-n) :: \mathcal{S}, P\rangle}\,(\texttt{sub})$$

$$\frac{}{\langle m :: n :: \mathcal{S}, \texttt{MUL}\ P\rangle \longrightarrow \langle (m\times n) :: \mathcal{S}, P\rangle}\,(\texttt{mul})$$

$$\frac{n \neq 0}{\langle m :: n :: \mathcal{S}, \texttt{DIV}\ P\rangle \longrightarrow \langle (m/n) :: \mathcal{S}, P\rangle}\,(\texttt{div})$$

$$\frac{}{\langle \mathcal{S}, \texttt{PUSH}\ n\ P\rangle \longrightarrow \langle n :: \mathcal{S}, P\rangle}\,(\texttt{push})$$

Example (Evaluation)

$\langle\ \varnothing\ ,$ `PUSH 2 PUSH 3 SUB PUSH 4 MUL` $\rangle \longrightarrow$

Example (Evaluation)

$\langle\ \varnothing\ ,$ `PUSH 2 PUSH 3 SUB PUSH 4 MUL` $\rangle \longrightarrow$

$\langle\ 2 :: \varnothing\ ,$ `PUSH 3 SUB PUSH 4 MUL` $\rangle \longrightarrow$

Example (Evaluation)

$\langle\ \varnothing\ ,$ **PUSH 2 PUSH 3 SUB PUSH 4 MUL** $\rangle \longrightarrow$

$\langle\ 2 :: \varnothing\ ,$ **PUSH 3 SUB PUSH 4 MUL** $\rangle \longrightarrow$

$\langle\ 3 :: 2 :: \varnothing\ ,$ **SUB PUSH 4 MUL** $\rangle \longrightarrow$

Example (Evaluation)

$\langle\ \varnothing\ ,$ `PUSH 2 PUSH 3 SUB PUSH 4 MUL` $\rangle \longrightarrow$

$\langle\ 2 :: \varnothing\ ,$ `PUSH 3 SUB PUSH 4 MUL` $\rangle \longrightarrow$

$\langle\ 3 :: 2 :: \varnothing\ ,$ `SUB PUSH 4 MUL` $\rangle \longrightarrow$

$\langle\ -1 :: \varnothing\ ,$ `PUSH 4 MUL` $\rangle \longrightarrow$

Example (Evaluation)

$\langle\ \varnothing\ ,\ \texttt{PUSH 2 PUSH 3 SUB PUSH 4 MUL}\ \rangle \longrightarrow$

$\langle\ 2 :: \varnothing\ ,\ \texttt{PUSH 3 SUB PUSH 4 MUL}\ \rangle \longrightarrow$

$\langle\ 3 :: 2 :: \varnothing\ ,\ \texttt{SUB PUSH 4 MUL}\ \rangle \longrightarrow$

$\langle\ -1 :: \varnothing\ ,\ \texttt{PUSH 4 MUL}\ \rangle \longrightarrow$

$\langle\ 4 :: -1 :: \varnothing\ ,\ \texttt{MUL}\ \rangle \longrightarrow$

Example (Evaluation)

$\langle\ \varnothing\ ,\ \texttt{PUSH 2 PUSH 3 SUB PUSH 4 MUL}\ \rangle \longrightarrow$

$\langle\ 2 :: \varnothing\ ,\ \texttt{PUSH 3 SUB PUSH 4 MUL}\ \rangle \longrightarrow$

$\langle\ 3 :: 2 :: \varnothing\ ,\ \texttt{SUB PUSH 4 MUL}\ \rangle \longrightarrow$

$\langle\ -1 :: \varnothing\ ,\ \texttt{PUSH 4 MUL}\ \rangle \longrightarrow$

$\langle\ 4 :: -1 :: \varnothing\ ,\ \texttt{MUL}\ \rangle \longrightarrow$

$\langle\ -4 :: \varnothing\ ,\ \epsilon\ \rangle \longrightarrow$

Example (Evaluation)

$\langle\ \varnothing\ ,\ \texttt{PUSH 2 PUSH 3 SUB PUSH 4 MUL}\ \rangle \longrightarrow$

$\langle\ 2 :: \varnothing\ ,\ \texttt{PUSH 3 SUB PUSH 4 MUL}\ \rangle \longrightarrow$

$\langle\ 3 :: 2 :: \varnothing\ ,\ \texttt{SUB PUSH 4 MUL}\ \rangle \longrightarrow$

$\langle\ -1 :: \varnothing\ ,\ \texttt{PUSH 4 MUL}\ \rangle \longrightarrow$

$\langle\ 4 :: -1 :: \varnothing\ ,\ \texttt{MUL}\ \rangle \longrightarrow$

$\langle\ -4 :: \varnothing\ ,\ \epsilon\ \rangle \longrightarrow\ -4$

Compiling Arithmetic Expressions

$$\texttt{n} \Longrightarrow \texttt{PUSH n}$$

$$e_1 \texttt{ + } e_2 \Longrightarrow \mathscr{C}(e_2)\ \mathscr{C}(e_1)\ \texttt{ADD}$$

$$e_1 \texttt{ - } e_2 \Longrightarrow \mathscr{C}(e_2)\ \mathscr{C}(e_1)\ \texttt{SUB}$$

$$e_1 \texttt{ * } e_2 \Longrightarrow \mathscr{C}(e_2)\ \mathscr{C}(e_1)\ \texttt{MUL}$$

$$e_1 \texttt{ / } e_2 \Longrightarrow \mathscr{C}(e_2)\ \mathscr{C}(e_1)\ \texttt{DIV}$$

We need a procedure $\mathscr{C}$ for converting an arithmetic expression into a stack program. *Note the order!*

Example (Compilation)

$\mathscr{C}$(`4 * (2 - 3)`)

Example (Compilation)

$\mathscr{C}$(`4 * (2 - 3)`)

$\Longrightarrow$ $\mathscr{C}$(`4`) $\mathscr{C}$(`2 - 3`) `MUL`

Example (Compilation)

$\mathscr{C}$(`4 * (2 - 3)`)

$\Longrightarrow$ $\mathscr{C}$(`4`) $\mathscr{C}$(`2 - 3`) `MUL`

$\Longrightarrow$ `PUSH 4` $\mathscr{C}$(`2 - 3`) `MUL`

Example (Compilation)

$\mathscr{C}$(4 * (2 - 3))

$\Longrightarrow$ $\mathscr{C}$(4) $\mathscr{C}$(2 - 3) MUL

$\Longrightarrow$ PUSH 4 $\mathscr{C}$(2 - 3) MUL

$\Longrightarrow$ PUSH 4 $\mathscr{C}$(2) $\mathscr{C}$(3) SUB MUL

Example (Compilation)

$\mathscr{C}$(4 * (2 - 3))

$\Longrightarrow$ $\mathscr{C}$(4) $\mathscr{C}$(2 - 3) MUL

$\Longrightarrow$ PUSH 4 $\mathscr{C}$(2 - 3) MUL

$\Longrightarrow$ PUSH 4 $\mathscr{C}$(2) $\mathscr{C}$(3) SUB MUL

$\Longrightarrow$ PUSH 4 PUSH 2 $\mathscr{C}$(3) SUB MUL

Example (Compilation)

$\mathscr{C}$(4 * (2 - 3))

$\Longrightarrow$ $\mathscr{C}$(4) $\mathscr{C}$(2 - 3) MUL

$\Longrightarrow$ PUSH 4 $\mathscr{C}$(2 - 3) MUL

$\Longrightarrow$ PUSH 4 $\mathscr{C}$(2) $\mathscr{C}$(3) SUB MUL

$\Longrightarrow$ PUSH 4 PUSH 2 $\mathscr{C}$(3) SUB MUL

$\Longrightarrow$ PUSH 4 PUSH 2 PUSH 2 SUB MUL

Variables

Variables (Syntax)

$$
\begin{aligned}
\texttt{<prog>} &::= \{\texttt{<com>}\} \\
\texttt{<com>} &::= \texttt{ADD} \mid \texttt{SUB} \mid \texttt{MUL} \mid \texttt{DIV} \mid \texttt{PUSH}\ \texttt{<num>} \\
&\quad \mid \texttt{ASSIGN}\ \texttt{<var>} \mid \texttt{LOOKUP}\ \texttt{<var>} \\
\texttt{<num>} &::= \mathbb{Z} \\
\texttt{<var>} &::= \mathbb{I}
\end{aligned}
$$

Variables (Semantics)

$$\langle \mathcal{S}, \mathcal{E}, P \rangle$$

A **value** is an integer ($\mathbb{Z}$)

A **configuration** is made up of a stack $\mathcal{S}$ of values, an environment $\mathcal{E}$ (mapping of identifiers to values), and a program P given by **<prog>**

Variables (Semantics)

$$\frac{}{\langle m :: n :: \mathcal{S}, \mathcal{E}, \texttt{ADD}\ P\rangle \longrightarrow \langle (m+n) :: \mathcal{S}, \mathcal{E}, P\rangle}\,(\texttt{add}) \qquad \frac{}{\langle m :: n :: \mathcal{S}, \mathcal{E}, \texttt{SUB}\ P\rangle \longrightarrow \langle (m-n) :: \mathcal{S}, \mathcal{E}, P\rangle}\,(\texttt{sub})$$

$$\frac{}{\langle m :: n :: \mathcal{S}, \mathcal{E}, \texttt{MUL}\ P\rangle \longrightarrow \langle (m \times n) :: \mathcal{S}, \mathcal{E}, P\rangle}\,(\texttt{mul}) \qquad \frac{n \neq 0}{\langle m :: n :: \mathcal{S}, \mathcal{E}, \texttt{DIV}\ P\rangle \longrightarrow \langle (m/n) :: \mathcal{S}, \mathcal{E}, P\rangle}\,(\texttt{div})$$

$$\frac{}{\langle \mathcal{S}, \mathcal{E}, \texttt{PUSH}\ n\ P\rangle \longrightarrow \langle n :: \mathcal{S}, \mathcal{E}, P\rangle}\,(\texttt{push})$$

$$\frac{}{\langle n :: \mathcal{S}, \mathcal{E}, \texttt{ASSIGN}\ x\ P\rangle \longrightarrow \langle \mathcal{S}, \mathcal{E}[x \mapsto n], P\rangle}\,(\texttt{asn}) \qquad \frac{}{\langle n :: \mathcal{S}, \mathcal{E}, \texttt{LOOKUP}\ x\ P\rangle \longrightarrow \langle \mathcal{E}(x) :: \mathcal{S}, \mathcal{E}, P\rangle}\,(\texttt{lkp})$$

Variables (Semantics)

basically the same

$$\frac{}{\langle m :: n :: \mathcal{S}, \mathcal{E}, \texttt{ADD}\ P\rangle \longrightarrow \langle (m+n) :: \mathcal{S}, \mathcal{E}, P\rangle}\,(\texttt{add}) \qquad \frac{}{\langle m :: n :: \mathcal{S}, \mathcal{E}, \texttt{SUB}\ P\rangle \longrightarrow \langle (m-n) :: \mathcal{S}, \mathcal{E}, P\rangle}\,(\texttt{sub})$$

$$\frac{}{\langle m :: n :: \mathcal{S}, \mathcal{E}, \texttt{MUL}\ P\rangle \longrightarrow \langle (m \times n) :: \mathcal{S}, \mathcal{E}, P\rangle}\,(\texttt{mul}) \qquad \frac{n \neq 0}{\langle m :: n :: \mathcal{S}, \mathcal{E}, \texttt{DIV}\ P\rangle \longrightarrow \langle (m/n) :: \mathcal{S}, \mathcal{E}, P\rangle}\,(\texttt{div})$$

$$\frac{}{\langle \mathcal{S}, \mathcal{E}, \texttt{PUSH}\ n\ P\rangle \longrightarrow \langle n :: \mathcal{S}, \mathcal{E}, P\rangle}\,(\texttt{push})$$

$$\frac{}{\langle n :: \mathcal{S}, \mathcal{E}, \texttt{ASSIGN}\ x\ P\rangle \longrightarrow \langle \mathcal{S}, \mathcal{E}[x \mapsto n], P\rangle}\,(\texttt{asn}) \qquad \frac{}{\langle n :: \mathcal{S}, \mathcal{E}, \texttt{LOOKUP}\ x\ P\rangle \longrightarrow \langle \mathcal{E}(x) :: \mathcal{S}, \mathcal{E}, P\rangle}\,(\texttt{lkp})$$

Variables (Semantics)

basically the same

$$\frac{}{\langle m :: n :: \mathcal{S}, \mathcal{E}, \texttt{ADD}\ P\rangle \longrightarrow \langle (m+n) :: \mathcal{S}, \mathcal{E}, P\rangle}\ (\texttt{add}) \qquad \frac{}{\langle m :: n :: \mathcal{S}, \mathcal{E}, \texttt{SUB}\ P\rangle \longrightarrow \langle (m-n) :: \mathcal{S}, \mathcal{E}, P\rangle}\ (\texttt{sub})$$

$$\frac{}{\langle m :: n :: \mathcal{S}, \mathcal{E}, \texttt{MUL}\ P\rangle \longrightarrow \langle (m \times n) :: \mathcal{S}, \mathcal{E}, P\rangle}\ (\texttt{mul}) \qquad \frac{n \neq 0}{\langle m :: n :: \mathcal{S}, \mathcal{E}, \texttt{DIV}\ P\rangle \longrightarrow \langle (m/n) :: \mathcal{S}, \mathcal{E}, P\rangle}\ (\texttt{div})$$

$$\frac{}{\langle \mathcal{S}, \mathcal{E}, \texttt{PUSH}\ n\ P\rangle \longrightarrow \langle n :: \mathcal{S}, \mathcal{E}, P\rangle}\ (\texttt{push})$$

new rules

$$\frac{}{\langle n :: \mathcal{S}, \mathcal{E}, \texttt{ASSIGN}\ x\ P\rangle \longrightarrow \langle \mathcal{S}, \mathcal{E}[x \mapsto n], P\rangle}\ (\texttt{asn}) \qquad \frac{}{\langle n :: \mathcal{S}, \mathcal{E}, \texttt{LOOKUP}\ x\ P\rangle \longrightarrow \langle \mathcal{E}(x) :: \mathcal{S}, \mathcal{E}, P\rangle}\ (\texttt{lkp})$$

Example (Evaluation)

$\langle \varnothing , \varnothing ,$ `PUSH 2 ASSIGN x PUSH 3 ASSIGN y LOOKUP x LOOKUP y ADD` $\rangle \longrightarrow$

Example (Evaluation)

$\langle\ \varnothing\ ,\ \varnothing\ ,$ **PUSH 2 ASSIGN x PUSH 3 ASSIGN y LOOKUP x LOOKUP y ADD** $\rangle \longrightarrow$

$\langle\ 2 :: \varnothing\ ,\ \varnothing\ ,$ **ASSIGN x PUSH 3 ASSIGN y LOOKUP x LOOKUP y ADD** $\rangle \longrightarrow$

Example (Evaluation)

$\langle\ \varnothing\ ,\ \varnothing\ ,$ **PUSH 2 ASSIGN x PUSH 3 ASSIGN y LOOKUP x LOOKUP y ADD** $\rangle \longrightarrow$

$\langle\ 2 :: \varnothing\ ,\ \varnothing\ ,$ **ASSIGN x PUSH 3 ASSIGN y LOOKUP x LOOKUP y ADD** $\rangle \longrightarrow$

$\langle\ \varnothing\ ,\ \{x \mapsto 2\}\ ,$ **PUSH 3 ASSIGN y LOOKUP x LOOKUP y ADD** $\rangle \longrightarrow$

Example (Evaluation)

$\langle\ \varnothing\ ,\ \varnothing\ ,$ **PUSH 2 ASSIGN x PUSH 3 ASSIGN y LOOKUP x LOOKUP y ADD** $\rangle \longrightarrow$

$\langle\ 2 :: \varnothing\ ,\ \varnothing\ ,$ **ASSIGN x PUSH 3 ASSIGN y LOOKUP x LOOKUP y ADD** $\rangle \longrightarrow$

$\langle\ \varnothing\ ,\ \{x \mapsto 2\}\ ,$ **PUSH 3 ASSIGN y LOOKUP x LOOKUP y ADD** $\rangle \longrightarrow$

$\langle\ 3 :: \varnothing\ ,\ \{x \mapsto 2\}\ ,$ **ASSIGN y LOOKUP x LOOKUP y ADD** $\rangle \longrightarrow$

Example (Evaluation)

$\langle\ \varnothing\ ,\ \varnothing\ ,$ **PUSH 2 ASSIGN x PUSH 3 ASSIGN y LOOKUP x LOOKUP y ADD** $\rangle \longrightarrow$

$\langle\ 2 :: \varnothing\ ,\ \varnothing\ ,$ **ASSIGN x PUSH 3 ASSIGN y LOOKUP x LOOKUP y ADD** $\rangle \longrightarrow$

$\langle\ \varnothing\ ,\ \{x \mapsto 2\}\ ,$ **PUSH 3 ASSIGN y LOOKUP x LOOKUP y ADD** $\rangle \longrightarrow$

$\langle\ 3 :: \varnothing\ ,\ \{x \mapsto 2\}\ ,$ **ASSIGN y LOOKUP x LOOKUP y ADD** $\rangle \longrightarrow$

$\langle\ \varnothing\ ,\ \{x \mapsto 2\ ,\ y \mapsto 3\}\ ,$ **LOOKUP x LOOKUP y ADD** $\rangle \longrightarrow$

Example (Evaluation)

$\langle\ \varnothing\ ,\ \varnothing\ ,$ **PUSH 2 ASSIGN x PUSH 3 ASSIGN y LOOKUP x LOOKUP y ADD** $\rangle \longrightarrow$

$\langle\ 2 :: \varnothing\ ,\ \varnothing\ ,$ **ASSIGN x PUSH 3 ASSIGN y LOOKUP x LOOKUP y ADD** $\rangle \longrightarrow$

$\langle\ \varnothing\ ,\ \{x \mapsto 2\}\ ,$ **PUSH 3 ASSIGN y LOOKUP x LOOKUP y ADD** $\rangle \longrightarrow$

$\langle\ 3 :: \varnothing\ ,\ \{x \mapsto 2\}\ ,$ **ASSIGN y LOOKUP x LOOKUP y ADD** $\rangle \longrightarrow$

$\langle\ \varnothing\ ,\ \{x \mapsto 2\ ,\ y \mapsto 3\}\ ,$ **LOOKUP x LOOKUP y ADD** $\rangle \longrightarrow$

$\langle\ 2 :: \varnothing\ ,\ \{x \mapsto 2\ ,\ y \mapsto 3\}\ ,$ **LOOKUP y ADD** $\rangle \longrightarrow$

Example (Evaluation)

$\langle\ \varnothing\ ,\ \varnothing\ ,$ **PUSH 2 ASSIGN x PUSH 3 ASSIGN y LOOKUP x LOOKUP y ADD** $\rangle \longrightarrow$

$\langle\ 2 :: \varnothing\ ,\ \varnothing\ ,$ **ASSIGN x PUSH 3 ASSIGN y LOOKUP x LOOKUP y ADD** $\rangle \longrightarrow$

$\langle\ \varnothing\ ,\ \{x \mapsto 2\}\ ,$ **PUSH 3 ASSIGN y LOOKUP x LOOKUP y ADD** $\rangle \longrightarrow$

$\langle\ 3 :: \varnothing\ ,\ \{x \mapsto 2\}\ ,$ **ASSIGN y LOOKUP x LOOKUP y ADD** $\rangle \longrightarrow$

$\langle\ \varnothing\ ,\ \{x \mapsto 2\ ,\ y \mapsto 3\}\ ,$ **LOOKUP x LOOKUP y ADD** $\rangle \longrightarrow$

$\langle\ 2 :: \varnothing\ ,\ \{x \mapsto 2\ ,\ y \mapsto 3\}\ ,$ **LOOKUP y ADD** $\rangle \longrightarrow$

$\langle\ 3 :: 2 :: \varnothing\ ,\ \{x \mapsto 2\ ,\ y \mapsto 3\}\ ,$ **ADD** $\rangle \longrightarrow$

Example (Evaluation)

$\langle\ \varnothing\ ,\ \varnothing\ ,$ **PUSH 2 ASSIGN x PUSH 3 ASSIGN y LOOKUP x LOOKUP y ADD** $\rangle \longrightarrow$

$\langle\ 2 :: \varnothing\ ,\ \varnothing\ ,$ **ASSIGN x PUSH 3 ASSIGN y LOOKUP x LOOKUP y ADD** $\rangle \longrightarrow$

$\langle\ \varnothing\ ,\ \{x \mapsto 2\}\ ,$ **PUSH 3 ASSIGN y LOOKUP x LOOKUP y ADD** $\rangle \longrightarrow$

$\langle\ 3 :: \varnothing\ ,\ \{x \mapsto 2\}\ ,$ **ASSIGN y LOOKUP x LOOKUP y ADD** $\rangle \longrightarrow$

$\langle\ \varnothing\ ,\ \{x \mapsto 2\ ,\ y \mapsto 3\}\ ,$ **LOOKUP x LOOKUP y ADD** $\rangle \longrightarrow$

$\langle\ 2 :: \varnothing\ ,\ \{x \mapsto 2\ ,\ y \mapsto 3\}\ ,$ **LOOKUP y ADD** $\rangle \longrightarrow$

$\langle\ 3 :: 2 :: \varnothing\ ,\ \{x \mapsto 2\ ,\ y \mapsto 3\}\ ,$ **ADD** $\rangle \longrightarrow$

$\langle\ 5 :: \varnothing\ ,\ \{x \mapsto 2\ ,\ y \mapsto 3\}\ ,\ \epsilon\ \rangle \longrightarrow$

Example (Evaluation)

$\langle\ \varnothing\ ,\ \varnothing\ ,\ \texttt{PUSH 2 ASSIGN x PUSH 3 ASSIGN y LOOKUP x LOOKUP y ADD}\ \rangle \longrightarrow$

$\langle\ 2 :: \varnothing\ ,\ \varnothing\ ,\ \texttt{ASSIGN x PUSH 3 ASSIGN y LOOKUP x LOOKUP y ADD}\ \rangle \longrightarrow$

$\langle\ \varnothing\ ,\ \{x \mapsto 2\}\ ,\ \texttt{PUSH 3 ASSIGN y LOOKUP x LOOKUP y ADD}\ \rangle \longrightarrow$

$\langle\ 3 :: \varnothing\ ,\ \{x \mapsto 2\}\ ,\ \texttt{ASSIGN y LOOKUP x LOOKUP y ADD}\ \rangle \longrightarrow$

$\langle\ \varnothing\ ,\ \{x \mapsto 2\ ,\ y \mapsto 3\}\ ,\ \texttt{LOOKUP x LOOKUP y ADD}\ \rangle \longrightarrow$

$\langle\ 2 :: \varnothing\ ,\ \{x \mapsto 2\ ,\ y \mapsto 3\}\ ,\ \texttt{LOOKUP y ADD}\ \rangle \longrightarrow$

$\langle\ 3 :: 2 :: \varnothing\ ,\ \{x \mapsto 2\ ,\ y \mapsto 3\}\ ,\ \texttt{ADD}\ \rangle \longrightarrow$

$\langle\ 5 :: \varnothing\ ,\ \{x \mapsto 2\ ,\ y \mapsto 3\}\ ,\ \epsilon\ \rangle \longrightarrow\ 5$

Compiling Let-Expressions (Attempt)

$$\texttt{x} \quad \Longrightarrow \quad \texttt{LOOKUP}\ x$$

$$\texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2 \quad \Longrightarrow \quad \mathscr{C}(e_1)\ \texttt{ASSIGN}\ x\ \mathscr{C}(e_2)$$

Compiling Let-Expressions (Attempt)

$$\texttt{x} \quad \Longrightarrow \quad \texttt{LOOKUP}\ x$$

$$\texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2 \quad \Longrightarrow \quad \mathscr{C}(e_1)\ \texttt{ASSIGN}\ x\ \mathscr{C}(e_2)$$

Except this isn't quite right

Example

$\mathscr{C}$(`let y = 1 in let x = let y = 2 in y in y`) $\Longrightarrow$

Example

$\mathscr{C}$(`let y = 1 in let x = let y = 2 in y in y`) $\Longrightarrow$

$\mathscr{C}$(`1`) `ASSIGN y` $\mathscr{C}$(`let x = let y = 2 in y in y`) $\Longrightarrow$

Example

$\mathscr{C}$(let y = 1 in let x = let y = 2 in y in y) $\Longrightarrow$

$\mathscr{C}$(1) ASSIGN y $\mathscr{C}$(let x = let y = 2 in y in y) $\Longrightarrow$

PUSH 1 ASSIGN y $\mathscr{C}$(let x = let y = 2 in y in y) $\Longrightarrow$

Example

$\mathscr{C}$(let y = 1 in let x = let y = 2 in y in y) $\Longrightarrow$

$\mathscr{C}$(1) ASSIGN y $\mathscr{C}$(let x = let y = 2 in y in y) $\Longrightarrow$

PUSH 1 ASSIGN y $\mathscr{C}$(let x = let y = 2 in y in y) $\Longrightarrow$

PUSH 1 ASSIGN y $\mathscr{C}$(let y = 2 in y) ASSIGN x $\mathscr{C}$(y) $\Longrightarrow$

Example

$\mathscr{C}$(let y = 1 in let x = let y = 2 in y in y) $\Longrightarrow$

$\mathscr{C}$(1) ASSIGN y $\mathscr{C}$(let x = let y = 2 in y in y) $\Longrightarrow$

PUSH 1 ASSIGN y $\mathscr{C}$(let x = let y = 2 in y in y) $\Longrightarrow$

PUSH 1 ASSIGN y $\mathscr{C}$(let y = 2 in y) ASSIGN x $\mathscr{C}$(y) $\Longrightarrow$

PUSH 1 ASSIGN y $\mathscr{C}$(2) ASSIGN y $\mathscr{C}$(y) ASSIGN x $\mathscr{C}$(y) $\Longrightarrow$

Example

$\mathscr{C}$(let y = 1 in let x = let y = 2 in y in y) $\Longrightarrow$

$\mathscr{C}$(1) ASSIGN y $\mathscr{C}$(let x = let y = 2 in y in y) $\Longrightarrow$

PUSH 1 ASSIGN y $\mathscr{C}$(let x = let y = 2 in y in y) $\Longrightarrow$

PUSH 1 ASSIGN y $\mathscr{C}$(let y = 2 in y) ASSIGN x $\mathscr{C}$(y) $\Longrightarrow$

PUSH 1 ASSIGN y $\mathscr{C}$(2) ASSIGN y $\mathscr{C}$(y) ASSIGN x $\mathscr{C}$(y) $\Longrightarrow$

PUSH 1 ASSIGN y PUSH 2 ASSIGN y $\mathscr{C}$(y) ASSIGN x $\mathscr{C}$(y) $\Longrightarrow$

Example

$\mathscr{C}$(let y = 1 in let x = let y = 2 in y in y) $\Longrightarrow$

$\mathscr{C}$(1) ASSIGN y $\mathscr{C}$(let x = let y = 2 in y in y) $\Longrightarrow$

PUSH 1 ASSIGN y $\mathscr{C}$(let x = let y = 2 in y in y) $\Longrightarrow$

PUSH 1 ASSIGN y $\mathscr{C}$(let y = 2 in y) ASSIGN x $\mathscr{C}$(y) $\Longrightarrow$

PUSH 1 ASSIGN y $\mathscr{C}$(2) ASSIGN y $\mathscr{C}$(y) ASSIGN x $\mathscr{C}$(y) $\Longrightarrow$

PUSH 1 ASSIGN y PUSH 2 ASSIGN y $\mathscr{C}$(y) ASSIGN x $\mathscr{C}$(y) $\Longrightarrow$

PUSH 1 ASSIGN y PUSH 2 ASSIGN y LOOKUP y ASSIGN x $\mathscr{C}$(y) $\Longrightarrow$

Example

$\mathscr{C}$(let y = 1 in let x = let y = 2 in y in y) $\Longrightarrow$

$\mathscr{C}$(1) ASSIGN y $\mathscr{C}$(let x = let y = 2 in y in y) $\Longrightarrow$

PUSH 1 ASSIGN y $\mathscr{C}$(let x = let y = 2 in y in y) $\Longrightarrow$

PUSH 1 ASSIGN y $\mathscr{C}$(let y = 2 in y) ASSIGN x $\mathscr{C}$(y) $\Longrightarrow$

PUSH 1 ASSIGN y $\mathscr{C}$(2) ASSIGN y $\mathscr{C}$(y) ASSIGN x $\mathscr{C}$(y) $\Longrightarrow$

PUSH 1 ASSIGN y PUSH 2 ASSIGN y $\mathscr{C}$(y) ASSIGN x $\mathscr{C}$(y) $\Longrightarrow$

PUSH 1 ASSIGN y PUSH 2 ASSIGN y LOOKUP y ASSIGN x $\mathscr{C}$(y) $\Longrightarrow$

PUSH 1 ASSIGN y PUSH 2 ASSIGN y LOOKUP y ASSIGN x LOOKUP y

Example

$\langle \varnothing , \varnothing ,$ `PUSH 1 ASSIGN y PUSH 2 ASSIGN y LOOKUP y ASSIGN x LOOKUP y` $\rangle \longrightarrow$

Example

$\langle\ \varnothing\ ,\ \varnothing\ ,$ `PUSH 1 ASSIGN y PUSH 2 ASSIGN y LOOKUP y ASSIGN x LOOKUP y` $\rangle \longrightarrow$

$\langle\ 1 :: \varnothing\ ,\ \varnothing\ ,$ `ASSIGN y PUSH 2 ASSIGN y LOOKUP y ASSIGN x LOOKUP y` $\rangle \longrightarrow$

Example

$\langle\ \varnothing\ ,\ \varnothing\ ,$ **PUSH 1 ASSIGN y PUSH 2 ASSIGN y LOOKUP y ASSIGN x LOOKUP y** $\rangle \longrightarrow$

$\langle\ 1 :: \varnothing\ ,\ \varnothing\ ,$ **ASSIGN y PUSH 2 ASSIGN y LOOKUP y ASSIGN x LOOKUP y** $\rangle \longrightarrow$

$\langle\ \varnothing\ ,\ \{y \mapsto 1\}\ ,$ **PUSH 2 ASSIGN y LOOKUP y ASSIGN x LOOKUP y** $\rangle \longrightarrow$

Example

$\langle\ \varnothing\ ,\ \varnothing\ ,$ `PUSH 1 ASSIGN y PUSH 2 ASSIGN y LOOKUP y ASSIGN x LOOKUP y` $\rangle \longrightarrow$

$\langle\ 1 :: \varnothing\ ,\ \varnothing\ ,$ `ASSIGN y PUSH 2 ASSIGN y LOOKUP y ASSIGN x LOOKUP y` $\rangle \longrightarrow$

$\langle\ \varnothing\ ,\ \{y \mapsto 1\}\ ,$ `PUSH 2 ASSIGN y LOOKUP y ASSIGN x LOOKUP y` $\rangle \longrightarrow$

$\langle\ 2 :: \varnothing\ ,\ \{y \mapsto 1\}\ ,$ `ASSIGN y LOOKUP y ASSIGN x LOOKUP y` $\rangle \longrightarrow$

Example

$\langle\ \varnothing\ ,\ \varnothing\ ,$ **PUSH 1 ASSIGN y PUSH 2 ASSIGN y LOOKUP y ASSIGN x LOOKUP y** $\rangle \longrightarrow$

$\langle\ 1 :: \varnothing\ ,\ \varnothing\ ,$ **ASSIGN y PUSH 2 ASSIGN y LOOKUP y ASSIGN x LOOKUP y** $\rangle \longrightarrow$

$\langle\ \varnothing\ ,\ \{y \mapsto 1\}\ ,$ **PUSH 2 ASSIGN y LOOKUP y ASSIGN x LOOKUP y** $\rangle \longrightarrow$

$\langle\ 2 :: \varnothing\ ,\ \{y \mapsto 1\}\ ,$ **ASSIGN y LOOKUP y ASSIGN x LOOKUP y** $\rangle \longrightarrow$

$\langle\ \varnothing\ ,\ \{y \mapsto 2\}\ ,$ **LOOKUP Y ASSIGN x LOOKUP y** $\rangle \longrightarrow$

Example

$\langle\ \varnothing\ ,\ \varnothing\ ,$ **PUSH 1 ASSIGN y PUSH 2 ASSIGN y LOOKUP y ASSIGN x LOOKUP y** $\rangle \longrightarrow$

$\langle\ 1 :: \varnothing\ ,\ \varnothing\ ,$ **ASSIGN y PUSH 2 ASSIGN y LOOKUP y ASSIGN x LOOKUP y** $\rangle \longrightarrow$

$\langle\ \varnothing\ ,\ \{y \mapsto 1\}\ ,$ **PUSH 2 ASSIGN y LOOKUP y ASSIGN x LOOKUP y** $\rangle \longrightarrow$

$\langle\ 2 :: \varnothing\ ,\ \{y \mapsto 1\}\ ,$ **ASSIGN y LOOKUP y ASSIGN x LOOKUP y** $\rangle \longrightarrow$

$\langle\ \varnothing\ ,\ \{y \mapsto 2\}\ ,$ **LOOKUP Y ASSIGN x LOOKUP y** $\rangle \longrightarrow$

$\langle\ 2 :: \varnothing\ ,\ \{y \mapsto 2\}\ ,$ **ASSIGN x LOOKUP y** $\rangle \longrightarrow$

Example

$\langle\ \varnothing\ ,\ \varnothing\ ,$ **PUSH 1 ASSIGN y PUSH 2 ASSIGN y LOOKUP y ASSIGN x LOOKUP y** $\rangle \longrightarrow$

$\langle\ 1 :: \varnothing\ ,\ \varnothing\ ,$ **ASSIGN y PUSH 2 ASSIGN y LOOKUP y ASSIGN x LOOKUP y** $\rangle \longrightarrow$

$\langle\ \varnothing\ ,\ \{y \mapsto 1\}\ ,$ **PUSH 2 ASSIGN y LOOKUP y ASSIGN x LOOKUP y** $\rangle \longrightarrow$

$\langle\ 2 :: \varnothing\ ,\ \{y \mapsto 1\}\ ,$ **ASSIGN y LOOKUP y ASSIGN x LOOKUP y** $\rangle \longrightarrow$

$\langle\ \varnothing\ ,\ \{y \mapsto 2\}\ ,$ **LOOKUP Y ASSIGN x LOOKUP y** $\rangle \longrightarrow$

$\langle\ 2 :: \varnothing\ ,\ \{y \mapsto 2\}\ ,$ **ASSIGN x LOOKUP y** $\rangle \longrightarrow$

$\langle\ \varnothing\ ,\ \{y \mapsto 2\ ,\ x \mapsto 2\}\ ,$ **LOOKUP y** $\rangle \longrightarrow$

Example

$\langle\ \varnothing\ ,\ \varnothing\ ,$ **PUSH 1 ASSIGN y PUSH 2 ASSIGN y LOOKUP y ASSIGN x LOOKUP y** $\rangle \longrightarrow$

$\langle\ 1 :: \varnothing\ ,\ \varnothing\ ,$ **ASSIGN y PUSH 2 ASSIGN y LOOKUP y ASSIGN x LOOKUP y** $\rangle \longrightarrow$

$\langle\ \varnothing\ ,\ \{y \mapsto 1\}\ ,$ **PUSH 2 ASSIGN y LOOKUP y ASSIGN x LOOKUP y** $\rangle \longrightarrow$

$\langle\ 2 :: \varnothing\ ,\ \{y \mapsto 1\}\ ,$ **ASSIGN y LOOKUP y ASSIGN x LOOKUP y** $\rangle \longrightarrow$

$\langle\ \varnothing\ ,\ \{y \mapsto 2\}\ ,$ **LOOKUP Y ASSIGN x LOOKUP y** $\rangle \longrightarrow$

$\langle\ 2 :: \varnothing\ ,\ \{y \mapsto 2\}\ ,$ **ASSIGN x LOOKUP y** $\rangle \longrightarrow$

$\langle\ \varnothing\ ,\ \{y \mapsto 2\ ,\ x \mapsto 2\}\ ,$ **LOOKUP y** $\rangle \longrightarrow$

$\langle\ 2 :: \varnothing\ ,\ \{y \mapsto 2\ ,\ x \mapsto 2\}\ ,\ \epsilon\ \rangle \longrightarrow$

Example

$\langle\ \varnothing\ ,\ \varnothing\ ,$ **PUSH 1 ASSIGN y PUSH 2 ASSIGN y LOOKUP y ASSIGN x LOOKUP y** $\rangle \longrightarrow$

$\langle\ 1 :: \varnothing\ ,\ \varnothing\ ,$ **ASSIGN y PUSH 2 ASSIGN y LOOKUP y ASSIGN x LOOKUP y** $\rangle \longrightarrow$

$\langle\ \varnothing\ ,\ \{y \mapsto 1\}\ ,$ **PUSH 2 ASSIGN y LOOKUP y ASSIGN x LOOKUP y** $\rangle \longrightarrow$

$\langle\ 2 :: \varnothing\ ,\ \{y \mapsto 1\}\ ,$ **ASSIGN y LOOKUP y ASSIGN x LOOKUP y** $\rangle \longrightarrow$

$\langle\ \varnothing\ ,\ \{y \mapsto 2\}\ ,$ **LOOKUP Y ASSIGN x LOOKUP y** $\rangle \longrightarrow$

$\langle\ 2 :: \varnothing\ ,\ \{y \mapsto 2\}\ ,$ **ASSIGN x LOOKUP y** $\rangle \longrightarrow$

$\langle\ \varnothing\ ,\ \{y \mapsto 2\ ,\ x \mapsto 2\}\ ,$ **LOOKUP y** $\rangle \longrightarrow$

$\langle\ 2 :: \varnothing\ ,\ \{y \mapsto 2\ ,\ x \mapsto 2\}\ ,\ \epsilon\ \rangle \longrightarrow\ 2$

Scoping

```
let y = 1 in
let x = let y = 2 in y in
y
```

$\nrightarrow 2$

Scoping

```
let y = 1 in
let x = let y = 2 in y in
y
```

$\nrightarrow 2$

The language we've just described is only good for compiling from languages with **dynamic scoping**

Scoping

```
let y = 1 in
let x = let y = 2 in y in
y
```

$\nrightarrow 2$

The language we've just described is only good for compiling from languages with **dynamic scoping**

We can use closures to deal with lexical scoping

Functions

The Rough Picture

```
let k = fun x -> fun y -> x in
let a = k 2 in
a 3
```

Compilation is just a big sequence of transformations

The Rough Picture

```
(fun k ->
   let a = k 2 in
   a 3)
(fun x -> fun y -> x)
```

We can simulate let expressions with functions
(we did this in lab)

The Rough Picture

```
(fun k ->
   (fun a -> a 3)
   (k 2))
(fun x -> fun y -> x)
```

and again...

The Rough Picture

```
[(fun k ->
   (fun a -> a 3)
   (k 2))
 (fun x -> fun y -> x)]
```

think of [expr] as as compile(expr)

The Rough Picture

```
[(fun x -> fun y -> x)]
[(fun k -> (fun a -> a 3) (k 2))]
CALL
```

We introduce as CALL command to call functions
Note the order, function/argument will go on a stack

The Rough Picture

```
FUN ? X
   [fun y -> x]
   RETURN
[(fun k -> (fun a -> a 3) (k 2))]
CALL
```

We introduce a FUN command to define functions and a RETURN command to return from functions

The Rough Picture

```
FUN ? X
  FUN ? Y
    [x]
    RETURN
  RETURN
[(fun k -> (fun a -> a 3) (k 2))]
CALL
```

and again...

The Rough Picture

```
FUN 4 X
  FUN 2 Y
    LOOKUP X
    RETURN
  RETURN
[(fun k -> (fun a -> a 3) (k 2))]
CALL
```

The familiar LOOKUP command...

And functions let us know how many commands they have

The Rough Picture

```
FUN 4 X
  FUN 2 Y
    LOOKUP X
    RETURN
  RETURN
FUN ? K
  [(fun a -> a 3) (k 2)]
  RETURN
CALL
```

and we can keep going...

The Rough Picture

```
FUN 4 X
  FUN 2 Y
    LOOKUP X
    RETURN
  RETURN
FUN ? K
  [k 2]
  [fun a -> a 3]
  CALL
  RETURN
CALL
```

The Rough Picture

```
FUN 4 X
  FUN 2 Y
    LOOKUP X
    RETURN
  RETURN
FUN ? K
  [2]
  [k]
  CALL
  [fun a -> a 3]
  CALL
  RETURN
CALL
```

The Rough Picture

```
FUN 4 X
  FUN 2 Y
    LOOKUP X
    RETURN
  RETURN
FUN ? K
  PUSH 2
  [k]
  CALL
  [fun a -> a 3]
  CALL
  RETURN
CALL
```

The Rough Picture

```
FUN 4 X
  FUN 2 Y
    LOOKUP X
    RETURN
  RETURN
FUN ? K
  PUSH 2
  LOOKUP K
  CALL
  [fun a -> a 3]
  CALL
  RETURN
CALL
```

The Rough Picture

```
FUN 4 X
  FUN 2 Y
    LOOKUP X
    RETURN
  RETURN
FUN ? K
  PUSH 2
  LOOKUP K
  CALL
  FUN ? A
    [a 3]
    RETURN
  CALL
  RETURN
CALL
```

The Rough Picture

```
FUN 4 X
  FUN 2 Y
    LOOKUP X
    RETURN
  RETURN
FUN ? K
  PUSH 2
  LOOKUP K
  CALL
  FUN ? A
    [3]
    [a]
    CALL
    RETURN
  CALL
  RETURN
CALL
```

The Rough Picture

```
FUN 4 X
  FUN 2 Y
    LOOKUP X
    RETURN
  RETURN
FUN ? K
  PUSH 2
  LOOKUP K
  CALL
  FUN ? A
    PUSH 3
    [a]
    CALL
    RETURN
  CALL
  RETURN
CALL
```

The Rough Picture

```
FUN 4 X
  FUN 2 Y
    LOOKUP X
    RETURN
  RETURN
FUN 10 K
  PUSH 2
  LOOKUP K
  CALL
  FUN 4 A
    PUSH 3
    LOOKUP A
    CALL
    RETURN
  CALL
  RETURN
CALL
```

The Rough Picture

```
let k = fun x -> fun y -> x in
let a = k 2 in
a 3
```

```
FUN 4 X
  FUN 2 Y
    LOOKUP X
    RETURN
  RETURN
FUN 10 K
  PUSH 2
  LOOKUP K
  CALL
  FUN 4 A
    PUSH 3
    LOOKUP A
    CALL
    RETURN
  CALL
  RETURN
CALL
```

Compilation is just a big sequence of transformations

The Rough Picture

```
let k = fun x -> fun y -> x in
let a = k 2 in
a 3
```

```
7 4
7 2
10 1
9
9
7 10
0 2
10 0
8
7 4
0 3
10 0
8
9
8
9
8
```

Byte-code interpretation additionally maps each command to a byte value

Syntax

```
<prog> ::= { <com> }
<com>  ::= PUSH <int>
         | ADD | SUB | MUL | DIV
         | FUN <ident> <int> | CALL | RETURN
         | LOOKUP
```

Semantics (Configurations)

$$\langle \mathcal{S}, \mathcal{E}, P \rangle$$

A **value** is an integer ($\mathbb{Z}$) or a closure ($\mathbb{C}$) of the form $(\mathcal{E}, x, P)$

A **configuration** is made up of a stack $\mathcal{S}$ of values, an environment $\mathcal{E}$ (mapping of identifiers to values) and a program P given by **<prog>**

Semantics (Functions)

$$\frac{}{\langle \mathcal{S}, \mathcal{E}, \mathsf{FUN}\ x\ n\ P \rangle \longrightarrow \langle (\mathcal{E}, x, P[1..n]) :: \mathcal{S}, \mathcal{E}'[x \mapsto v], P[n+1..] \rangle}\ (\texttt{fun})$$

Function definitions carry a **parameter name** and an **offset,** which we use to construct the closure

Semantics (Continuation Passing)

$$\frac{}{\langle (\mathscr{E}', x, P') :: v :: \mathscr{S}, \mathscr{E}, \texttt{CALL}\ P \rangle \longrightarrow \langle (\mathscr{E}, \_, P) :: \mathscr{S}, \mathscr{E}'[x \mapsto v], P' \rangle}\ (\texttt{call})$$

$$\frac{}{\langle v :: (\mathscr{E}', \_, P') :: \mathscr{S}, \mathscr{E}, \texttt{RETURN}\ P \rangle \longrightarrow \langle v :: \mathscr{S}, \mathscr{E}', P' \rangle}\ (\texttt{ret})$$

One challenge: when we call a function, where to we "return" to?

Answer: We put the information on the stack itself in a closure!

Example

Stack:

Env:

```
FUN 4 X
  FUN 2 Y
    LOOKUP X
    RETURN
  RETURN
FUN 10 K
  PUSH 2
  LOOKUP K
  CALL
  FUN 4 A
    PUSH 3
    LOOKUP A
    CALL
    RETURN
  CALL
  RETURN
CALL
```

Example

Stack:

```
∅
x
FUN 2 Y
  LOOKUP X
  RETURN
RETURN
```

Env:

```
FUN 10 K
  PUSH 2
  LOOKUP K
  CALL
  FUN 4 A
    PUSH 3
    LOOKUP A
    CALL
    RETURN
  CALL
  RETURN
CALL
```

Example

Stack:

```
∅
X
FUN 2 Y
  LOOKUP X
  RETURN
RETURN
```

```
∅
K
PUSH 2
LOOKUP K
CALL
FUN 4 A
  PUSH 3
  LOOKUP A
  CALL
  RETURN
CALL
RETURN
```

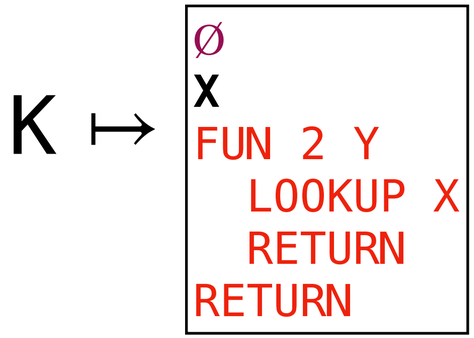
Env:

CALL

Example

Stack:

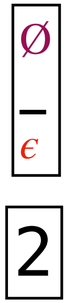
Env:



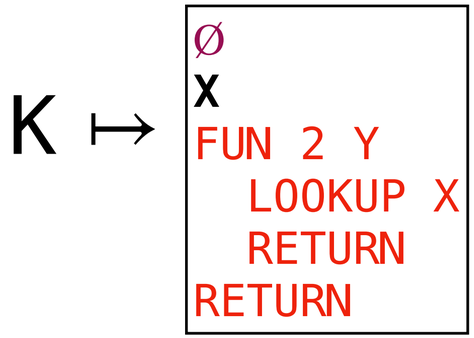

```
PUSH 2
LOOKUP K
CALL
FUN 4 A
  PUSH 3
  LOOKUP A
  CALL
  RETURN
CALL
RETURN
```

Example

Stack:



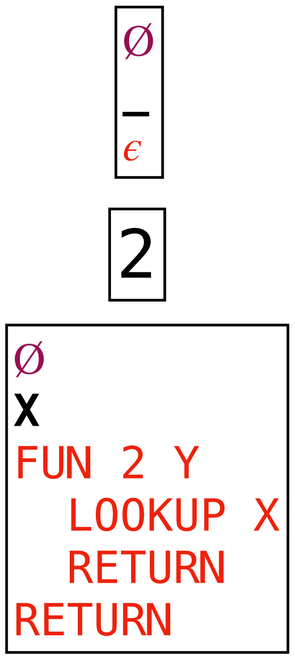
Env:



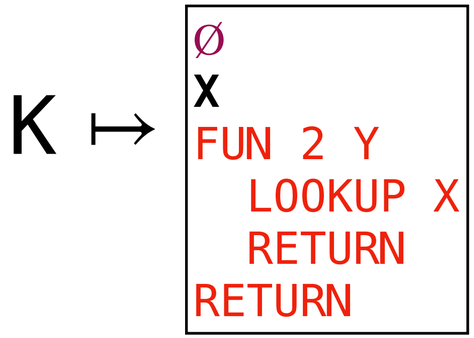

```
LOOKUP K
CALL
FUN 4 A
  PUSH 3
  LOOKUP A
  CALL
  RETURN
CALL
RETURN
```

Example

Stack:



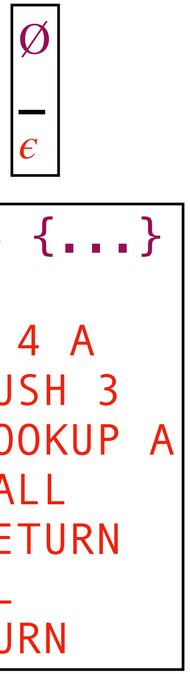
Env:




```
CALL
FUN 4 A
  PUSH 3
  LOOKUP A
  CALL
  RETURN
CALL
RETURN
```

Example

Stack:



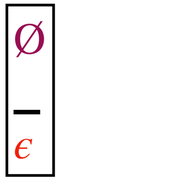
Env:

X ↦ 2

```
FUN 2 Y
  LOOKUP X
  RETURN
RETURN
```

Example

Stack:




```
K ↦ {...}
–
FUN 4 A
  PUSH 3
  LOOKUP A
  CALL
  RETURN
CALL
RETURN
```

```
X ↦ 2
Y
LOOKUP X
RETURN
```

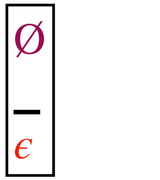
Env:

X ↦ 2

RETURN

Example

Stack:




```
X ↦ 2
Y
LOOKUP X
RETURN
```

Env:

K ↦

```
∅
X
FUN 2 Y
  LOOKUP X
  RETURN
RETURN
```

```
FUN 4 A
  PUSH 3
  LOOKUP A
  CALL
  RETURN
CALL
RETURN
```

Example

Stack:


```
X ↦ 2
Y
LOOKUP X
RETURN
```

```
K ↦ {...}
A
PUSH 3
LOOKUP A
CALL
RETURN
```

Env:

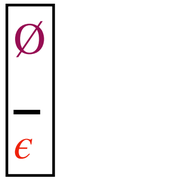
K ↦

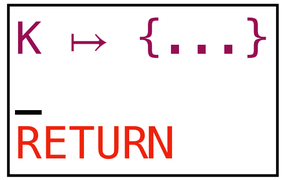
```
Ø
X
FUN 2 Y
  LOOKUP X
  RETURN
RETURN
```

```
CALL
RETURN
```

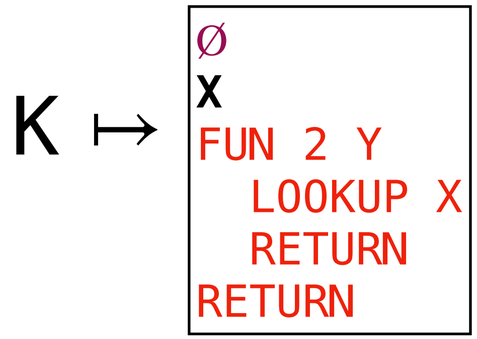
Example

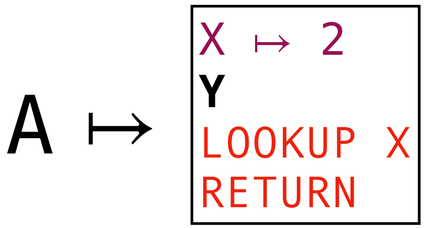
Stack:





Env:

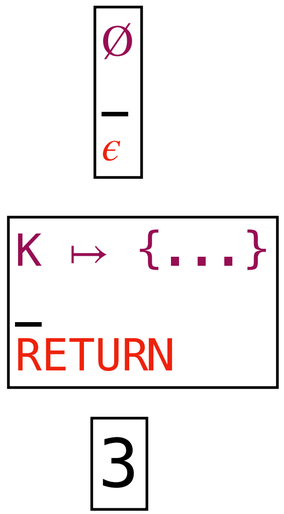




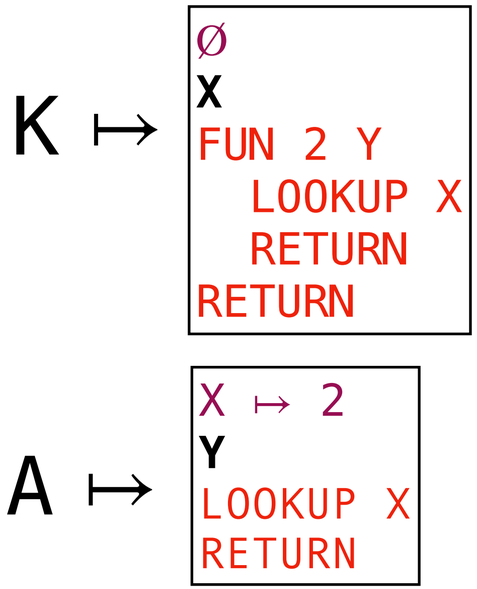

```
PUSH 3
LOOKUP A
CALL
RETURN
```

Example

Stack:



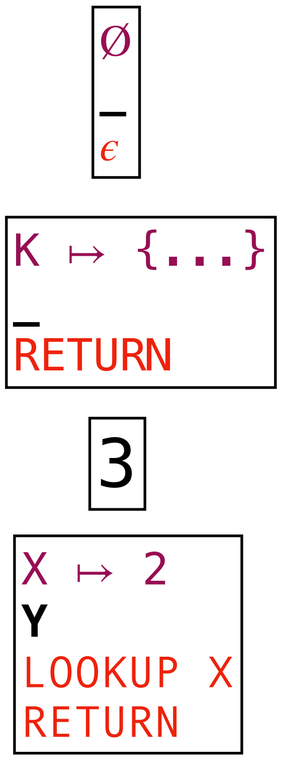
Env:



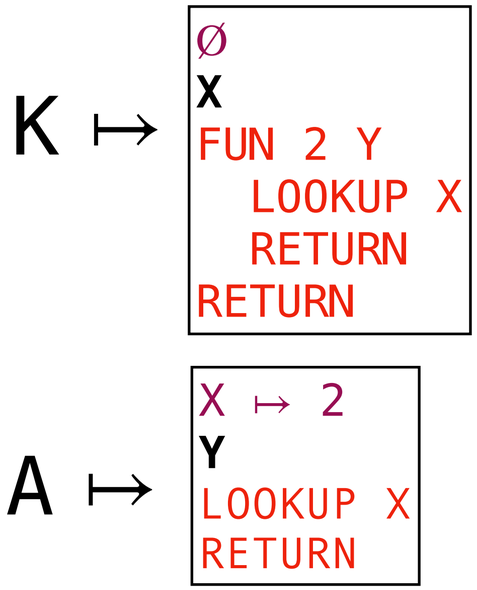

```
LOOKUP A
CALL
RETURN
```

Example

Stack:



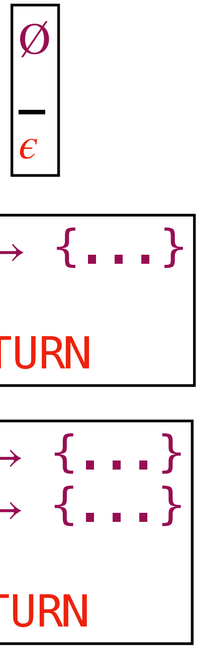
Env:



CALL
RETURN

Example

Stack:



Env:

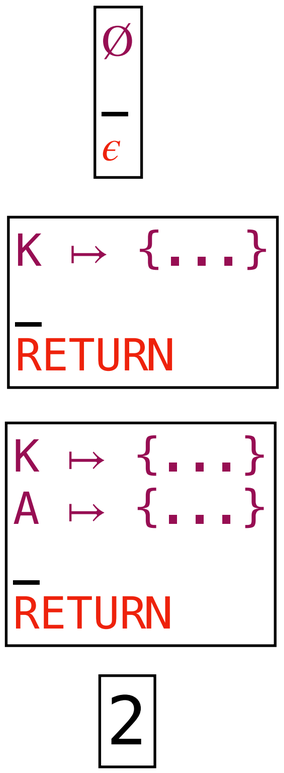
X ↦ 2

Y ↦ 3

```
LOOKUP X
RETURN
```

Example

Stack:



Env:

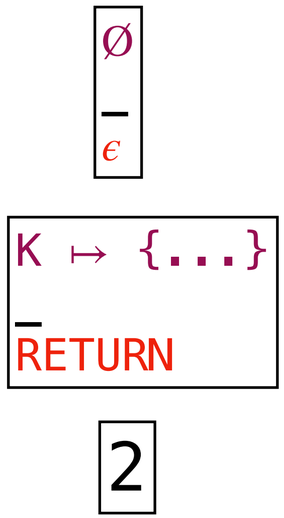
X ↦ 2

Y ↦ 3

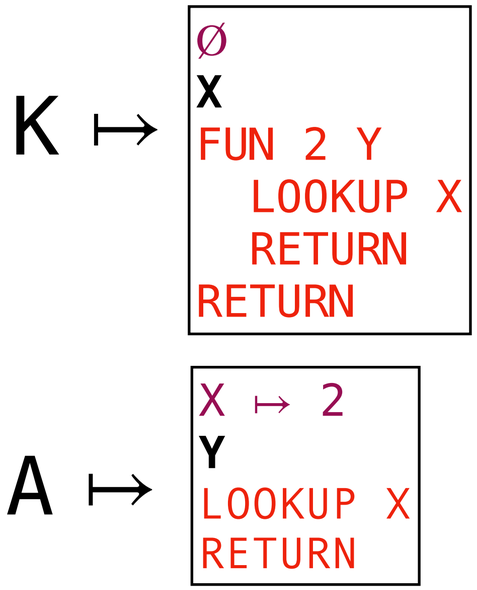
RETURN

Example

Stack:



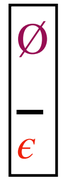
Env:



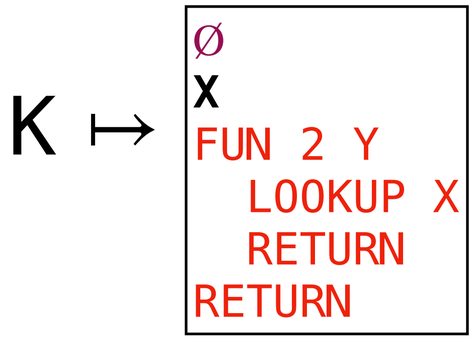
RETURN

Example

Stack:



Env:



RETURN

Example

Stack:

2

Env:

ϵ

demo

What's next?

More OCaml:

» Modules, functional data structures, mutability

» GADTS, effects, parallelism

» applications in ML, linear algebra, scientific computing

More PL:

» Take Professor Xi's compilers course next semester

» Learn Haskell, Elm, Scala, Rust

More Math/Type Theory:

» Go learn about logic with Professor Das next semester!

» Category theory (functors, monads, comonads), Logic, Type theory

» (I do a lot of independent studies)

More Computers:

» Compilers, Linkers, LLVM

» Formal verification

» embedded systems programming, tensor program compilation

fin

(thanks everyone)