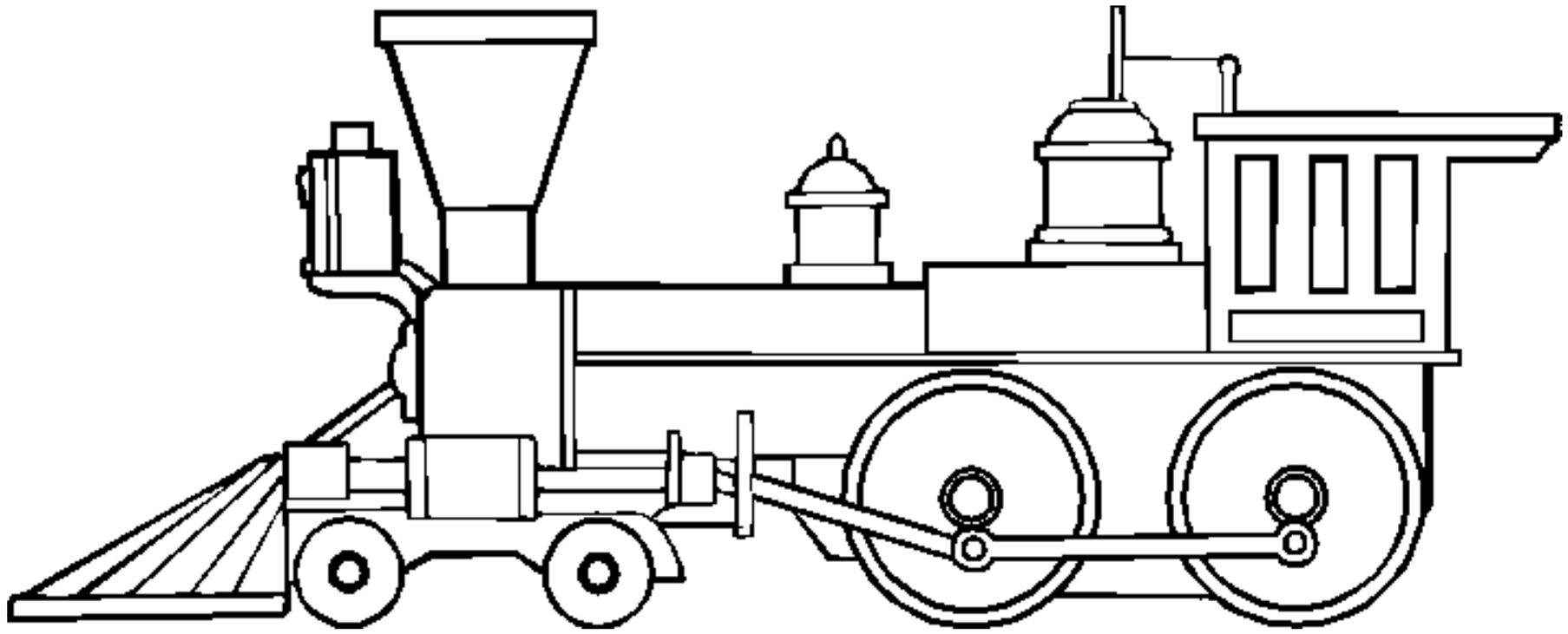
Matt Might
University of Utah
matt.might.net



CESK machines

No class Thursday!

Today

- Project 3: Due next Monday
- Finish up mini-lang walkthrough
- Explain and implement CESK

Reduction

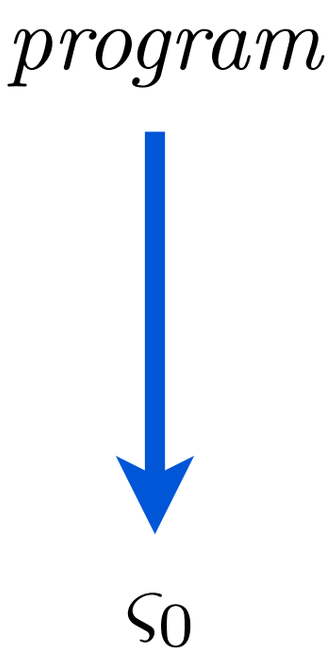
$$program \Rightarrow program'$$

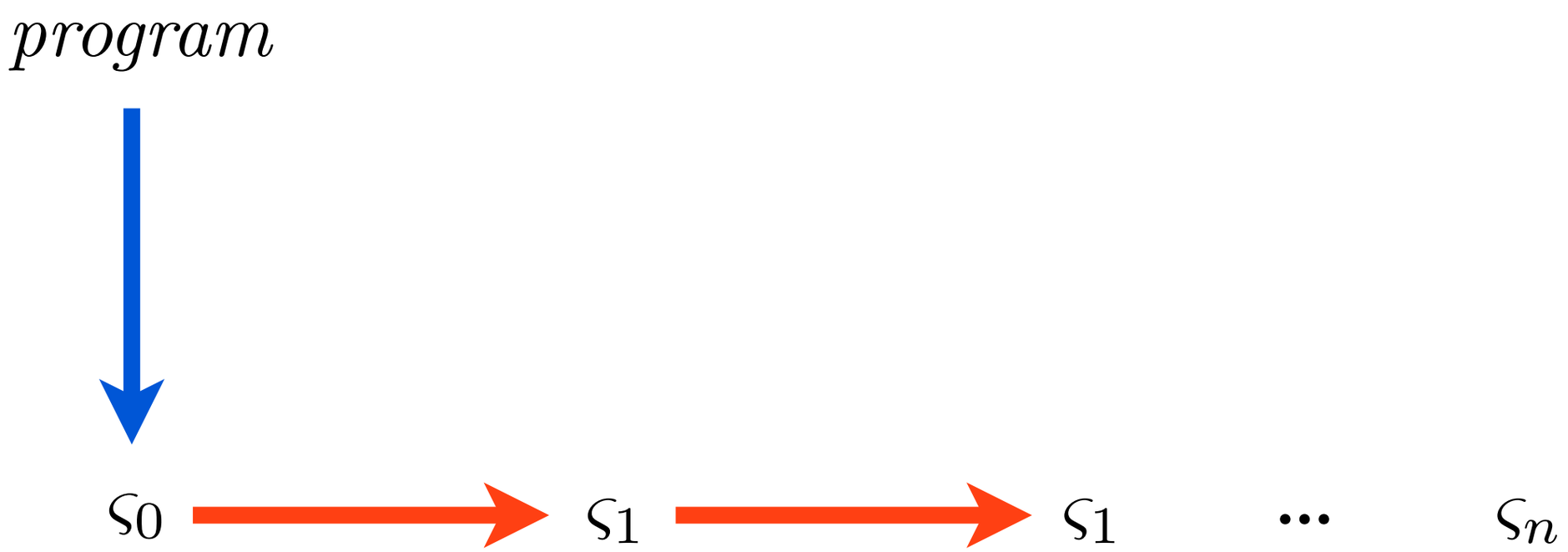
(+ 3 (+ 4 5)) => (+ 3 9)

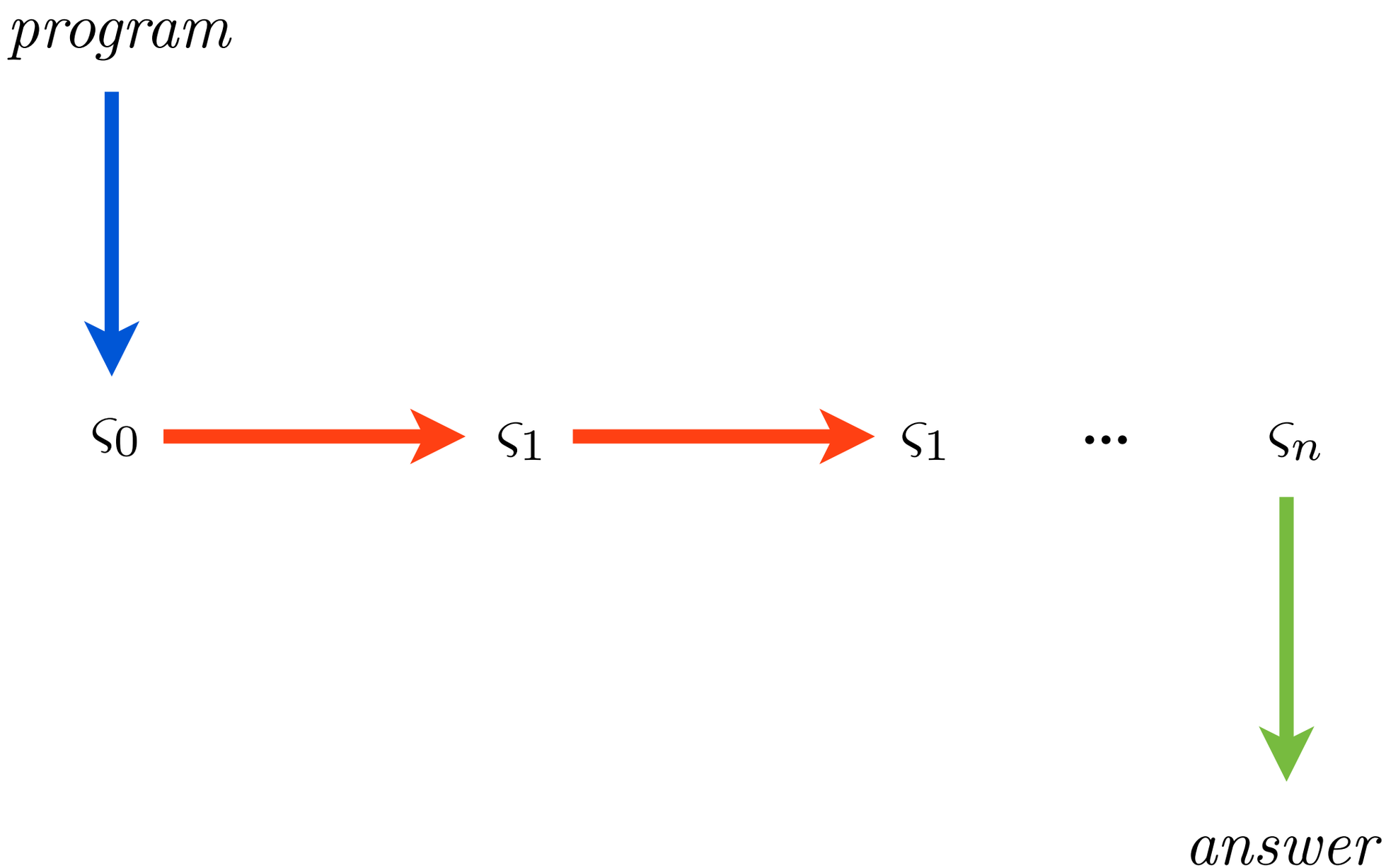
Mini-lang

Small-step machines

program


program
ς_0


program
ς_0
ς_1
ς_1
...
ς_n


program
ς_0
ς_1
ς_1
...
ς_n
answer

CESK machines

Why CESK?

- Mutation
- Higher-order
- Exceptions
- Coroutines
- Continuations
- Threads

What's in a state?

CESK

Control

Environment

Store

Kontinuation

Control

Environment

Store

Continuation

Program counter

Registers

Heap

Stack


x86

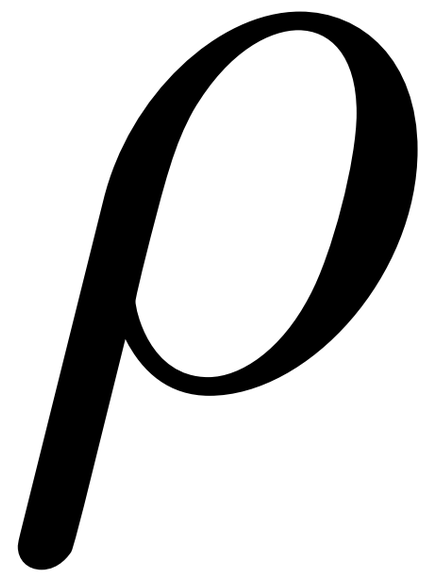

λ

Control

$$
\begin{array}{lll}
exp & ::= & var \\
 & | & (exp\ exp) \\
 & | & (\lambda\ (var)\ exp)
\end{array}
$$

Environment

$$Env = \mathsf{Var} \rightharpoonup Addr$$



Store

$$Store = Addr \rightharpoonup Clo$$

$$\sigma$$

$$Clo = \text{Lambda} \times Env$$

```
(lambda (x) z)
```

```
(lambda (x) z)
```

```
[z => 3]
```

Continuation

$$\kappa \in Kont ::= \mathbf{arg}(e, \rho, \kappa)$$
$$\mid \quad \mathbf{fun}(clo, \kappa)$$
$$\mid \quad \mathbf{halt}$$

$$\kappa \in Kont = Frame^*$$

$$\phi \in Frame ::= \mathbf{arg}(e, \rho)$$

$$\mid \quad \mathbf{fun}(clo)$$

CESK in Racket

Yuck.

Language

Machine

Language

Machine

Language

Machine

Language

Machine

A-Normal Form (ANF)

A-Normal Form

All arguments to functions are atomic.

Flanagan, Sabry, Duba, Felleisen.
"The Essence of Compiling with Continuations."

Atomic?

- No side effects
- No control effects
- Must terminate
- Must not error

$$
\begin{array}{rcl}
M & ::= & V \\
 & & \mid (\textbf{let}\ (x\ V)\ M) \\
 & & \mid (\textbf{if0}\ V\ M\ M) \\
 & & \mid (V\ V_1\ \ldots\ V_n) \\
 & & \mid (\textbf{let}\ (x\ (V\ V_1\ \ldots\ V_n))\ M) \\
 & & \mid (O\ V_1\ \ldots\ V_n) \\
 & & \mid (\textbf{let}\ (x\ (O\ V_1\ \ldots\ V_n))\ M)
\end{array}
$$

$$
V ::= c \mid x \mid (\lambda x_1 \ldots x_n.M)
$$

A-Normalization

```
(define normalize-term (lambda (M) (normalize M (lambda (x) x))))

(define normalize
  (lambda (M k)
    (match M
      [`(lambda ,params ,body) (k `(lambda ,params ,(normalize-term body)))]
      [`(let (,x ,M1) ,M2) (normalize M1 (lambda (N1) `(let (,x ,N1) ,(normalize M2 k))))]
      [`(if0 ,M1 ,M2 ,M3) (normalize-name M1 (lambda (t) (k `(if0 ,t ,(normalize-term M2) ,(normalize-term M3)))))]
      [`(,Fn . ,M*) (if (PrimOp? Fn)
                        (normalize-name* M* (lambda (t*) (k `(,Fn . ,t*))))
                        (normalize-name Fn (lambda (t) (normalize-name* M* (lambda (t*) (k `(,t . ,t*)))))))]
      [V (k V)])))

(define normalize-name
  (lambda (M k)
    (normalize M (lambda (N) (if (Value? N) (k N) (let ([t (newvar)]) `(let (,t ,N) ,(k t))))))))

(define normalize-name*
  (lambda (M* k)
    (if (null? M*)
        (k '())
        (normalize-name (car M*) (lambda (t) (normalize-name* (cdr M*) (lambda (t*) (k `(,t . ,t*)))))))))
```

Example

```
(define (celsius F)
  (* (/ 5 9) (- F 32)))
```

Example

```
(define (celsius F)
  (let ([t1 (/ 5 9)])
    (let ([t2 (- F 32)])
      (* t1 t2))))
```

Benefits

- Order of evaluation is specified
- Evaluation broken into small steps

CESK for ANF in Racket