Intermediate forms: A-Normal Form

Matt Might
University of Utah
www.ucombinator.org
matt.might.net

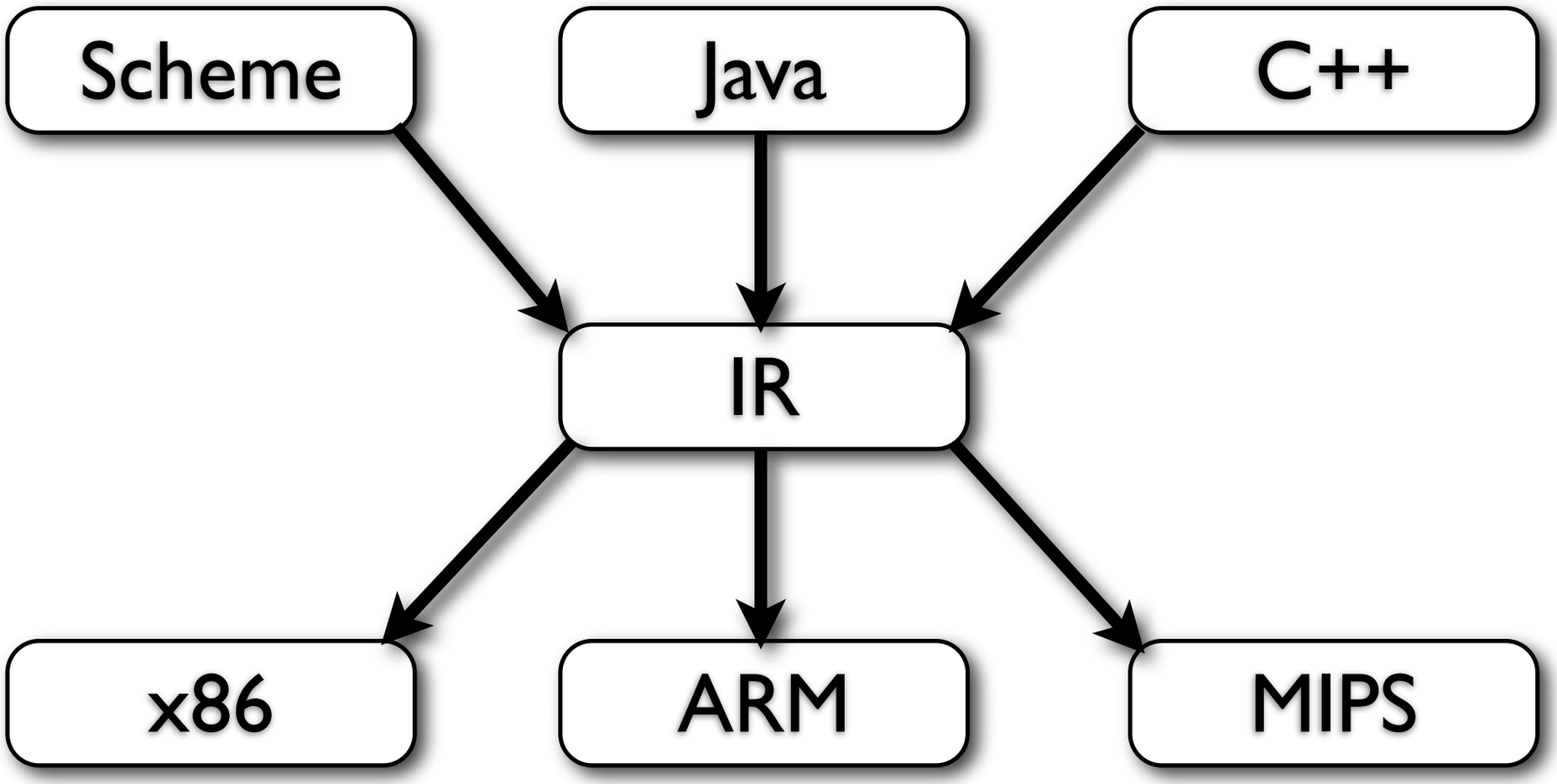
Terminology

- Intermediate language
- Intermediate representation
- Intermediate form

Intermediate form

An **intermediate language** is a target language that sits above the output language.

Purpose



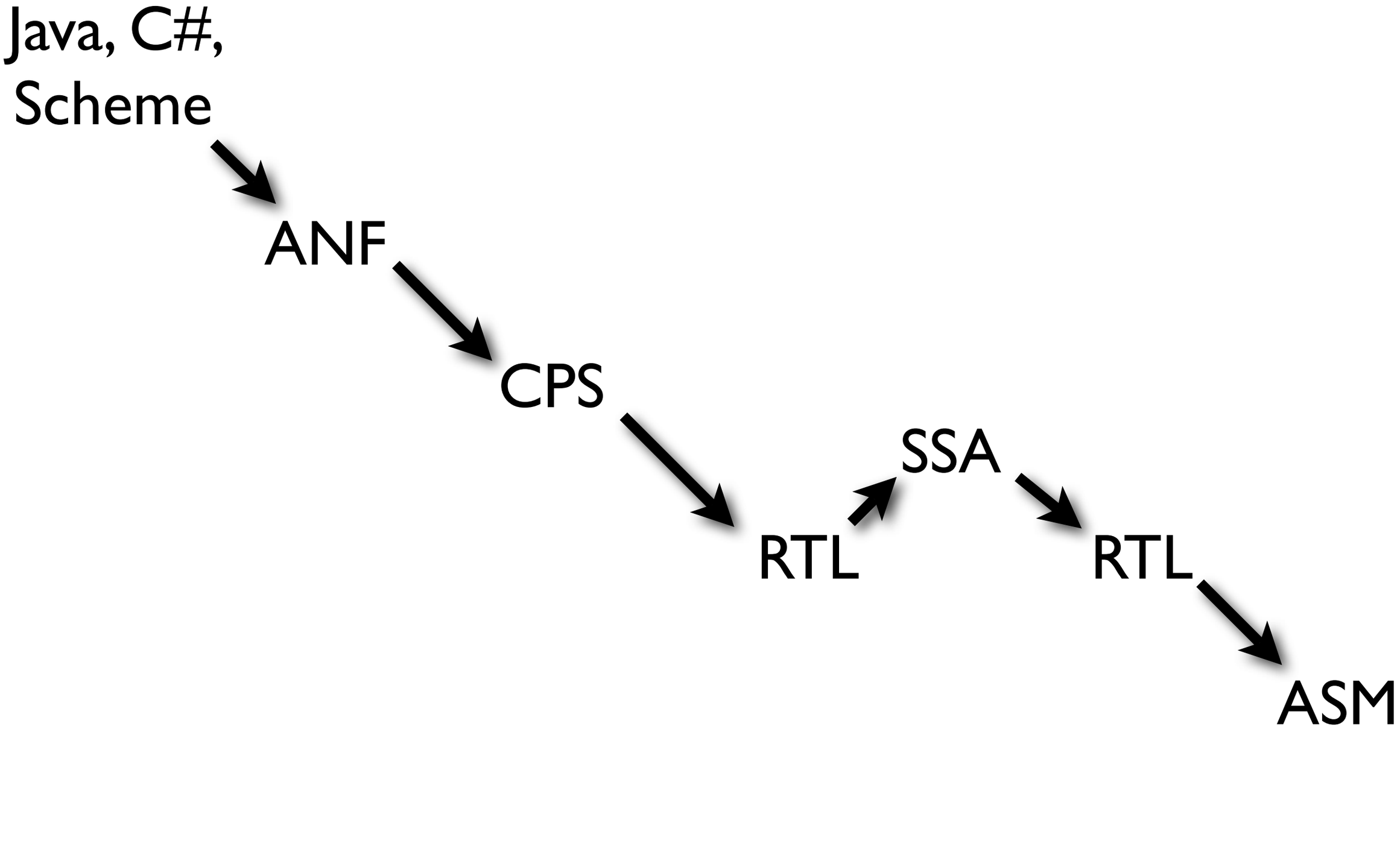
Nano-pass compilation

60+ intermediate forms!

History

- Register-transfer languages (1950s)
- Continuation-passing style (1970s)
- Static single assignment (1980s)
- A-normal form (1990s)
- Hybridization? (2010s)

A pathway



Lesser-known

- Store-passing style
- Monadic languages

Standard IRs

- CIL
- LLVM
- GIMPLE
- C--
- JIMPLE/SHIMPLE

A-Normal Form

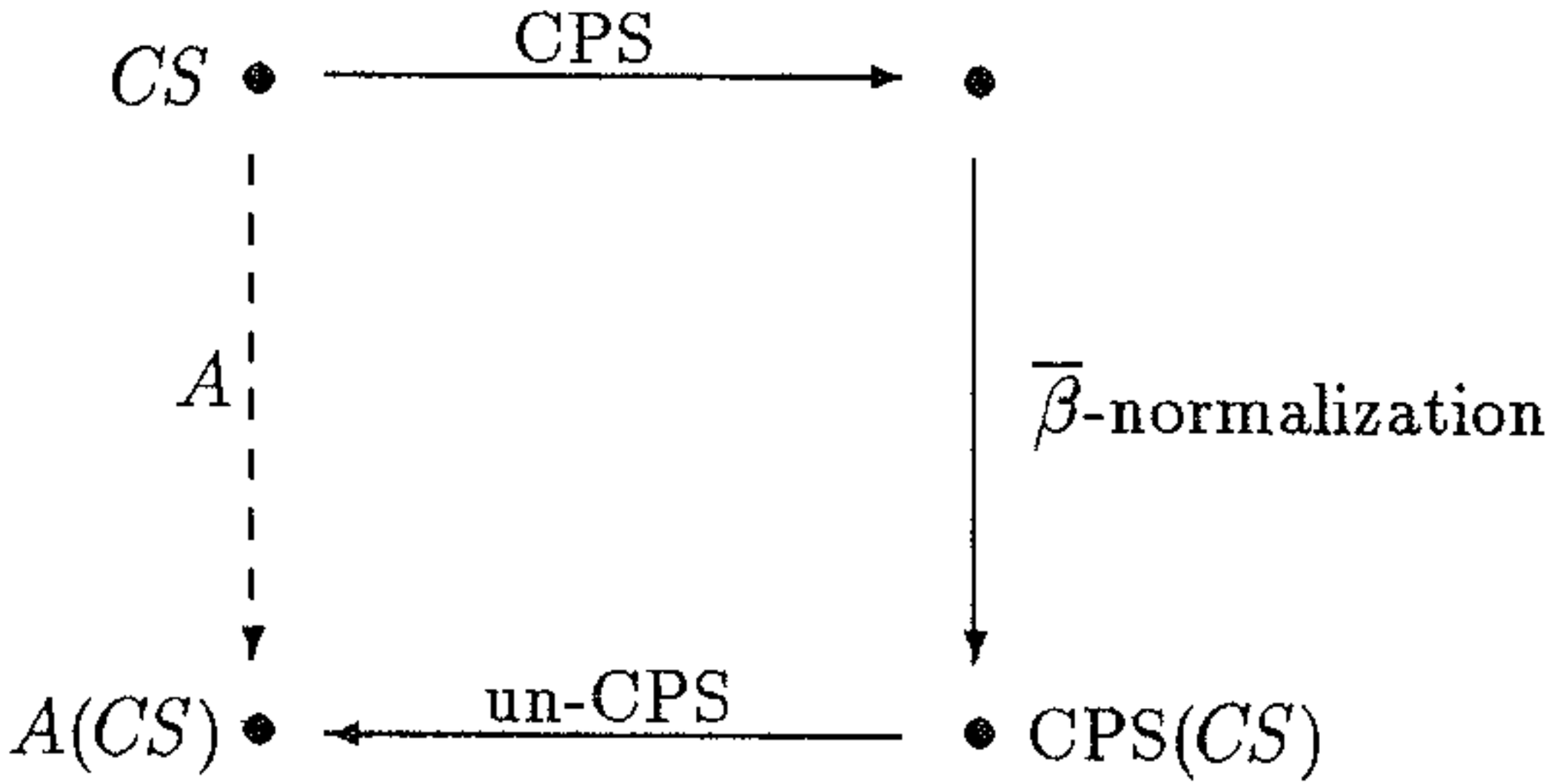
All arguments to functions are atomic.

Flanagan et al.,
"The Essence of Compiling with Continuations."

A-Normal Form

$$
\begin{array}{rcl}
M & ::= & V \\
 & & \mid (\textbf{let}\ (x\ V)\ M) \\
 & & \mid (\textbf{if0}\ V\ M\ M) \\
 & & \mid (V\ V_1\ \ldots\ V_n) \\
 & & \mid (\textbf{let}\ (x\ (V\ V_1\ \ldots\ V_n))\ M) \\
 & & \mid (O\ V_1\ \ldots\ V_n) \\
 & & \mid (\textbf{let}\ (x\ (O\ V_1\ \ldots\ V_n))\ M) \\
 & & \\
V & ::= & c \mid x \mid (\lambda x_1 \ldots x_n.M)
\end{array}
$$

Discovering ANF



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

Correspondence

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

```
celsius: mov F,   a0
         div t1,  5, 9
         sub t2,  F, 32
         mul res, t1, t2
         ret res
```

Example

```
(define (fact n)
 (if (zero? n)
     1
     (* n (fact (- n 1)))))
```

```
(define (fact n)
 (let ((zero? (= n 0)))
  (if zero?
      1
      (let ((n-1 (- n 1)))
        (let ((|n-1|! (fact n-1)))
          (* n |n-1|!))))))
```

Example

```
fact: mov   n, a0
      cmp   t1, n, 0
      bz    t1, body
      ret   1
body: sub   t2, n, 1
      call  fact
      mov   t3, rv
      mul   t4, n, t3
      ret   t4
```

```
(define (fact n)
 (let ((zero? (= n 0)))
  (if zero?
      1
      (let ((n-1 (- n 1)))
        (let ((|n-1|! (fact n-1)))
          (* n |n-1|!)))))))
```

Example (Java)

```
return foo.bar(x.f,qux()) ;
```

```
F v0 = x.f ;
Q q0 = qux() ;
return foo.bar(v0,q0) ;
```