CMSC 430: Introduction to Compilers

Functional

Thomas Gilray (3:15-4:30p, 4161AVW)

Javran Cheng (12:30-1:45p, 4103AVW)

cs.umd.edu/class/fall2017/cmsc430

Why take compilers?

Sapir-Whorf Hypothesis

At least true for programming languages…

Why take compilers?

- Likewise, learning how the compiler thinks *makes you a better programmer.*
- Can make smarter use of existing compilers.
- Will apply many of the same principles elsewhere.
- You may eventually work on a smaller language, or perhaps even a large one.
- Avoid introducing something indecent into the world…

Such as equality semantics in PHP…

problem, boole?

Loose comparisons with ==

	TRUE	FALSE	1	0	-1	"1"	"0"	"-1"	NULL	array()	"php"	""
TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
1	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
0	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE
-1	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
"1"	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
"0"	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
"-1"	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
NULL	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
array()	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
"php"	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
""	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE

Lexing

↓

Parsing

↓

Macros/preprocessing

↓

Intermediate representation (IR/IL)

↓

Smaller IR

↓

Tiny IR (admin bindings)

↓

Static single assignment (SSA)

↓

Register allocation

↓

Code emission (asm)

Lexing

↓

Parsing

↓

Macros/preprocessing

↓

Intermediate representation (IR/IL)

↓

Smaller IR

↓

Tiny IR (admin bindings)

↓

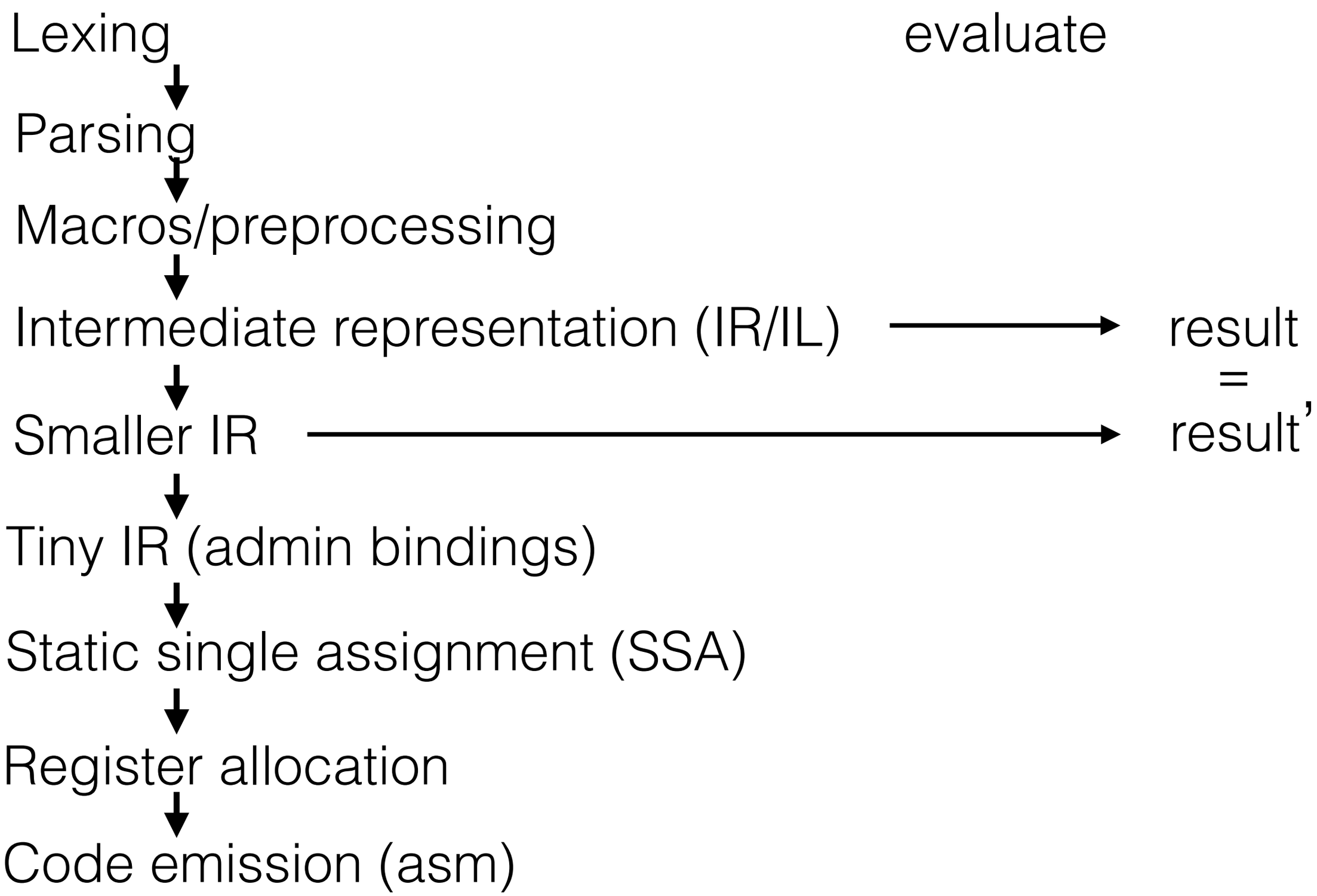
Static single assignment (SSA)

↓

Register allocation

↓

Code emission (asm)


Lexing
evaluate
Parsing
Macros/preprocessing
Intermediate representation (IR/IL)
result
=
Smaller IR
result'
Tiny IR (admin bindings)
Static single assignment (SSA)
Register allocation
Code emission (asm)

Lexing

↓

Parsing

↓

Macros/preprocessing

↓

Intermediate representation (IR/IL)

↓

Smaller IR

↓

Tiny IR (admin bindings)

Assignment conversion

Static single assignment (SSA)

Continuation-passing style conversion

Register allocation

Closure conversion

Code emission (asm)

The big idea:

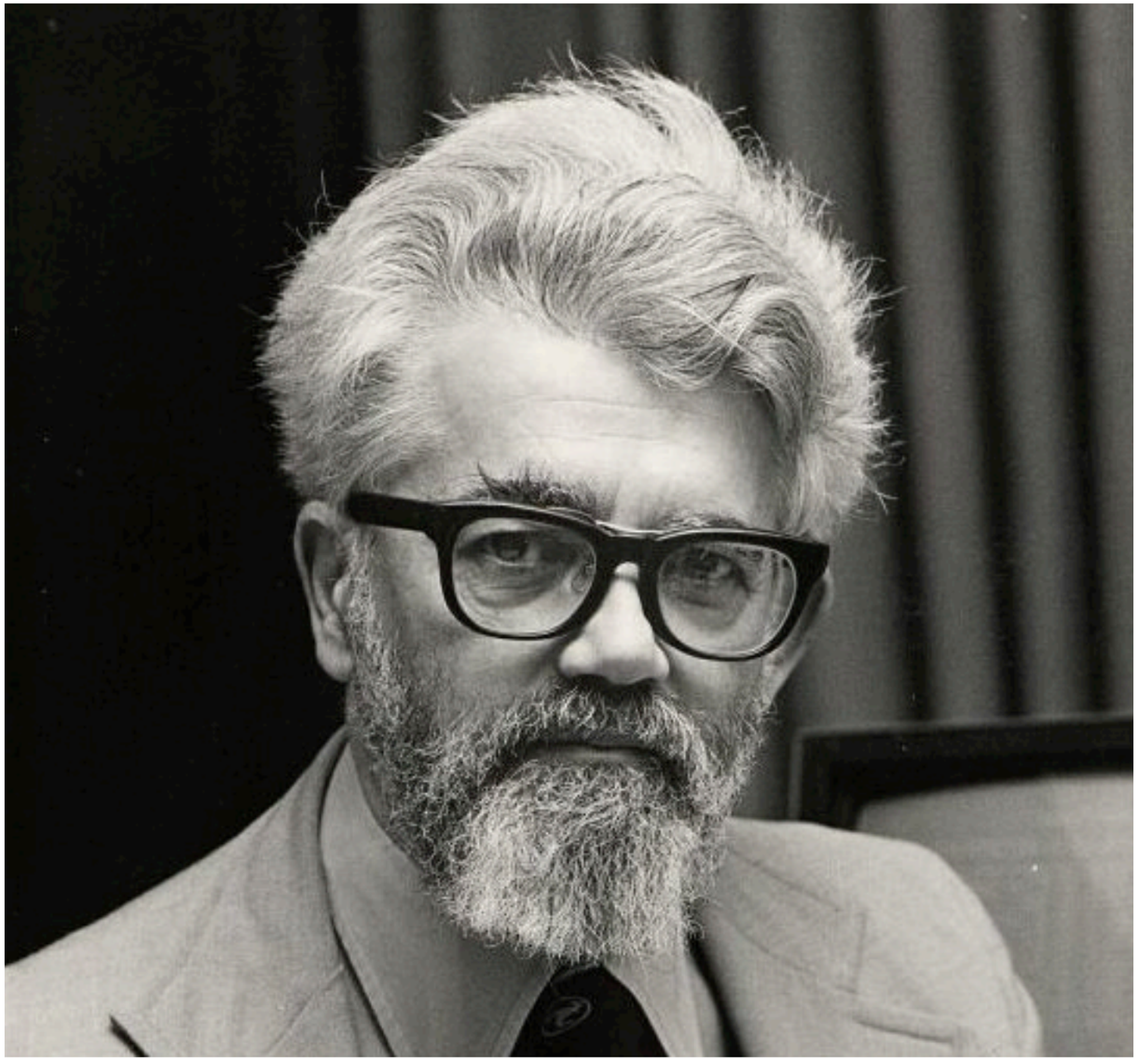
Compile λ well

Then compile *almost* everything into λ

λ-calculus

Alonzo Church

LISP



John McCarthy

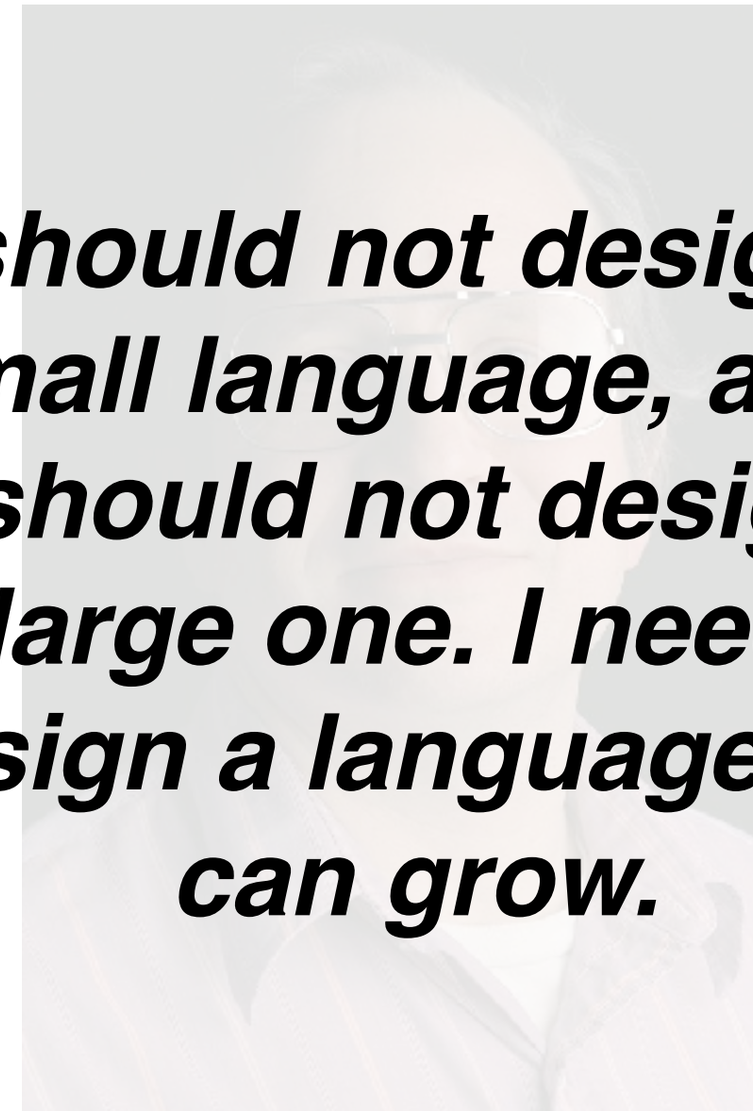
Scheme

Guy Steele

Gerald Jay Sussman

Scheme

Guy Steele



Gerald Jay Sussman

“***I should not design a small language, and I should not design a large one. I need to design a language that can grow.***”

Growing a language. (1998)

PLT Scheme -> Racket

Matthias Felleisen

(et al.)

Matthew Flatt

Symbolic expressions (s-expr)

(. . .)

```
(tag children ...)
```

Textual encoding for lists/trees

```
<tag>
  children ...
</tag>
```

```
<tag id="child" ...>
  children ...
</tag>
```

```
(tag ([id child] ...)
     children ...)
```

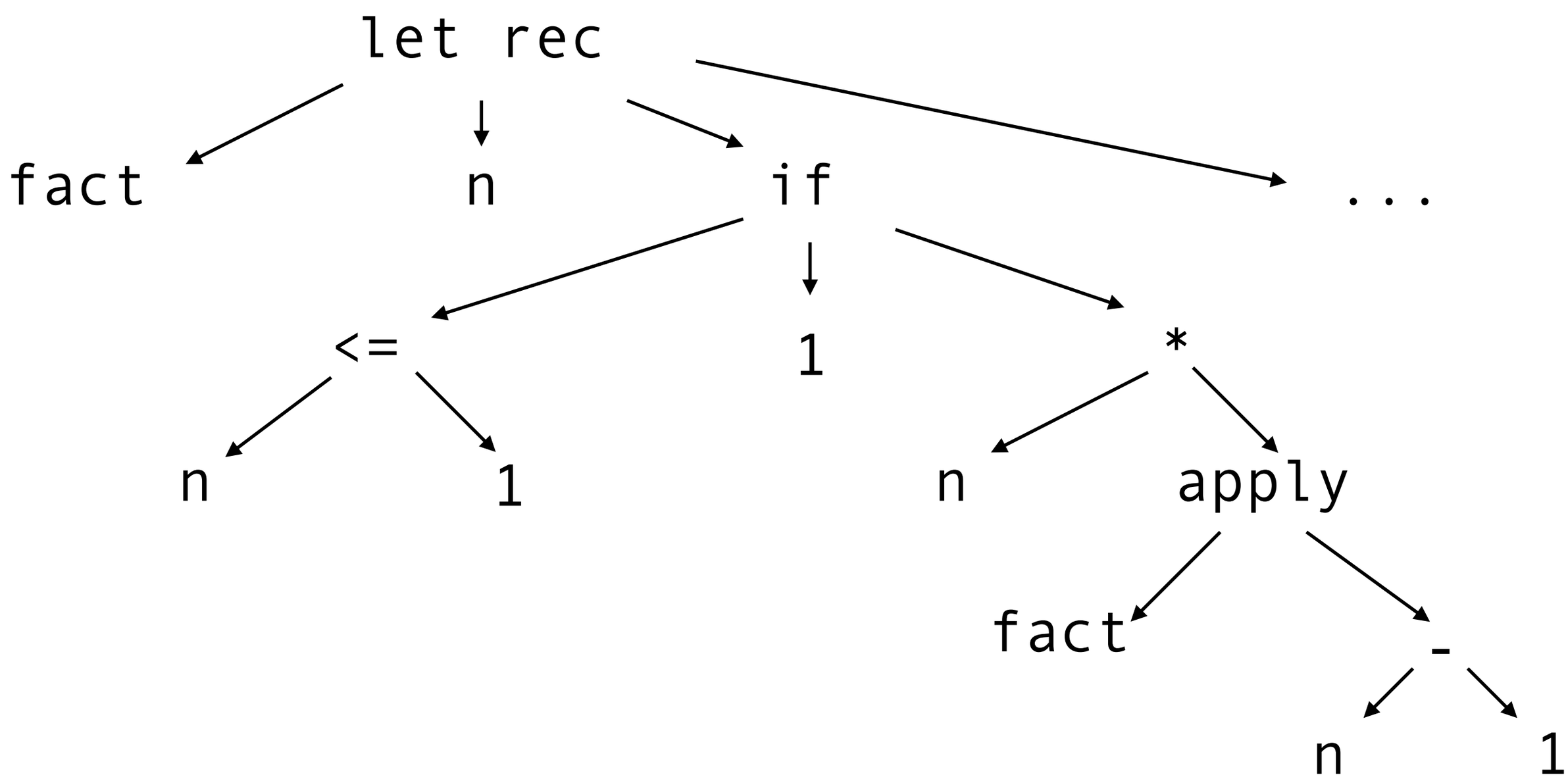
```
(xexpr->xml
 `(tag ([id child] ...)
    children ...))
```

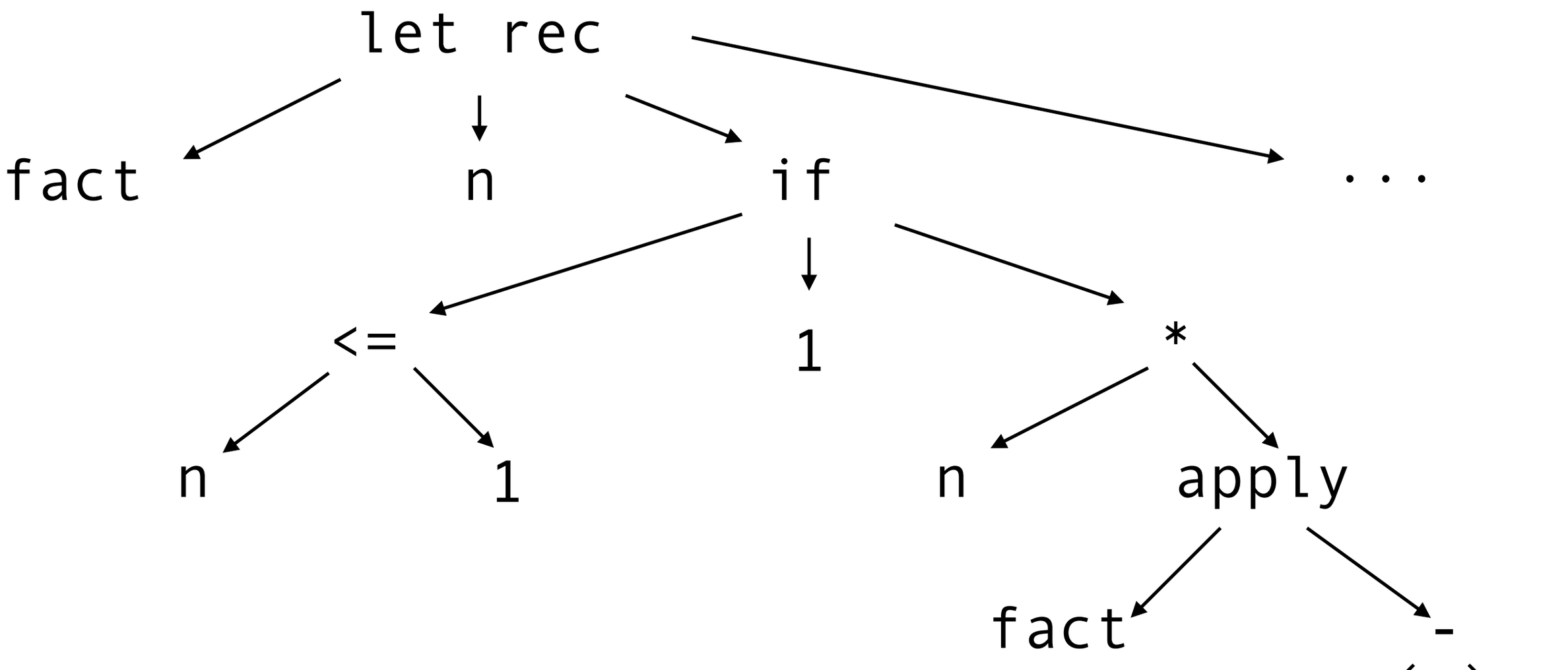
↓

```
"<tag id=\"child\" ...>
  children ...
</tag>"
```

```
let rec fact n =
  if n <= 1
  then 1
  else n * fact (n - 1)
...
```

```
let rec fact n =
  if n <= 1
  then 1
  else n * fact (n - 1)
...
```






```
(letrec ([fact
          (λ (n)
             (if (<= n 1)
                 1
                 (* n (fact (- n 1))))))])
  ...)
```

This week.

- Download and install Racket 6.10
- Try out DrRacket IDE and the cmd-line Racket REPL.
- Try examples and start learning the language. It will simply take some time using the language to learn it.
- Warm-up assignment 0 is online now. Due Mon, 9/4.
- Use submit.cs.umd.edu to submit. Start early.

Let’s try some Racket.