Compilers.

David Luposchainsky

2018-03-09

Compiler

A compiler is a translation engine between two formal languages.

Compiler

A compiler is a translation engine between two formal languages.

Input

```
-- My fancy program
main = putStrLn "Hello, world!"
```

Compiler

A compiler is a translation engine between two formal languages.

Input

```
-- My fancy program
main = putStrLn "Hello, world!"
```

Output

```
# xxd a.out
...
00002ea0: c508 4a8b 742b 084a 8974 2808 4983 c508  ..J.t+.J.t(.I...
00002eb0: 4e8b 442b 084e 8944 2808 4983 c508 4a8b  N.D+.N.D(.I...J.
00002ec0: 542b 084a 8954 2808 4983 c508 4e8b 4c2b  T+.J.T(.I...N.L+
00002ed0: 084e 894c 2808 4983 c508 4e8b 542b 084e  .N.L(.I...N.T+.N
00002ee0: 8954 2808 4983 c508 4d39 dd0f 84ef 0a00  .T(.I...M9......
00002ef0: 004e 8b64 2b08 4e89 6428 0849 8b7c 1d10  .N.d+.N.d(.I.|..
00002f00: 4a89 7c28 104d 8b74 1d18 4e89 7428 1849  J.|(.M.t..N.t(.I
00002f10: 8b4c 1d20 4a89 4c28 2049 8b74 1d28 4a89  .L. J.L( I.t.(J.
...
```

Magic!

Magic!

...not really

Small pieces

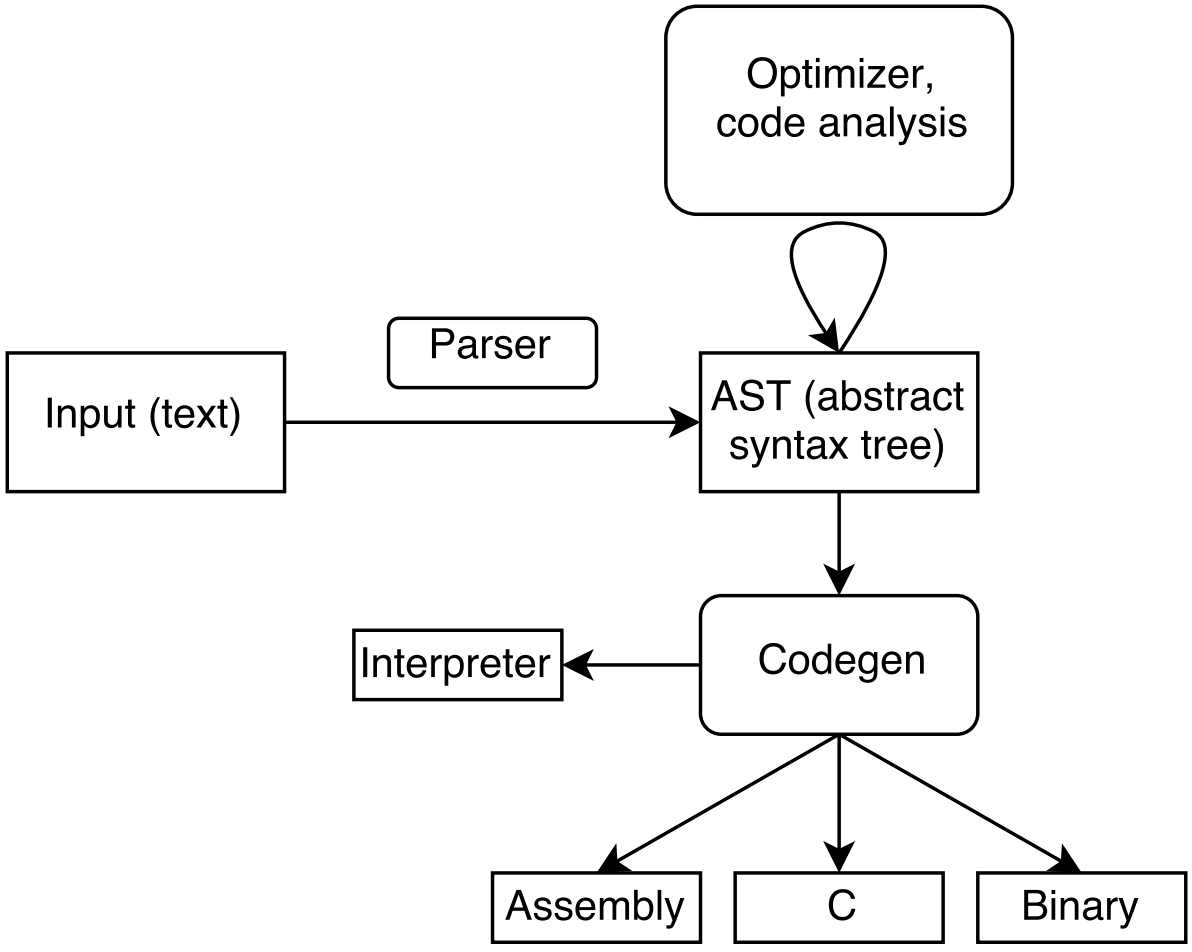

Figure 1: Compiler pipeline

Parsing

```
    def main():
        text = "hello world"
        print(text)
```

Parsing

```
def main():
    text = "hello world"
    print(text)
```

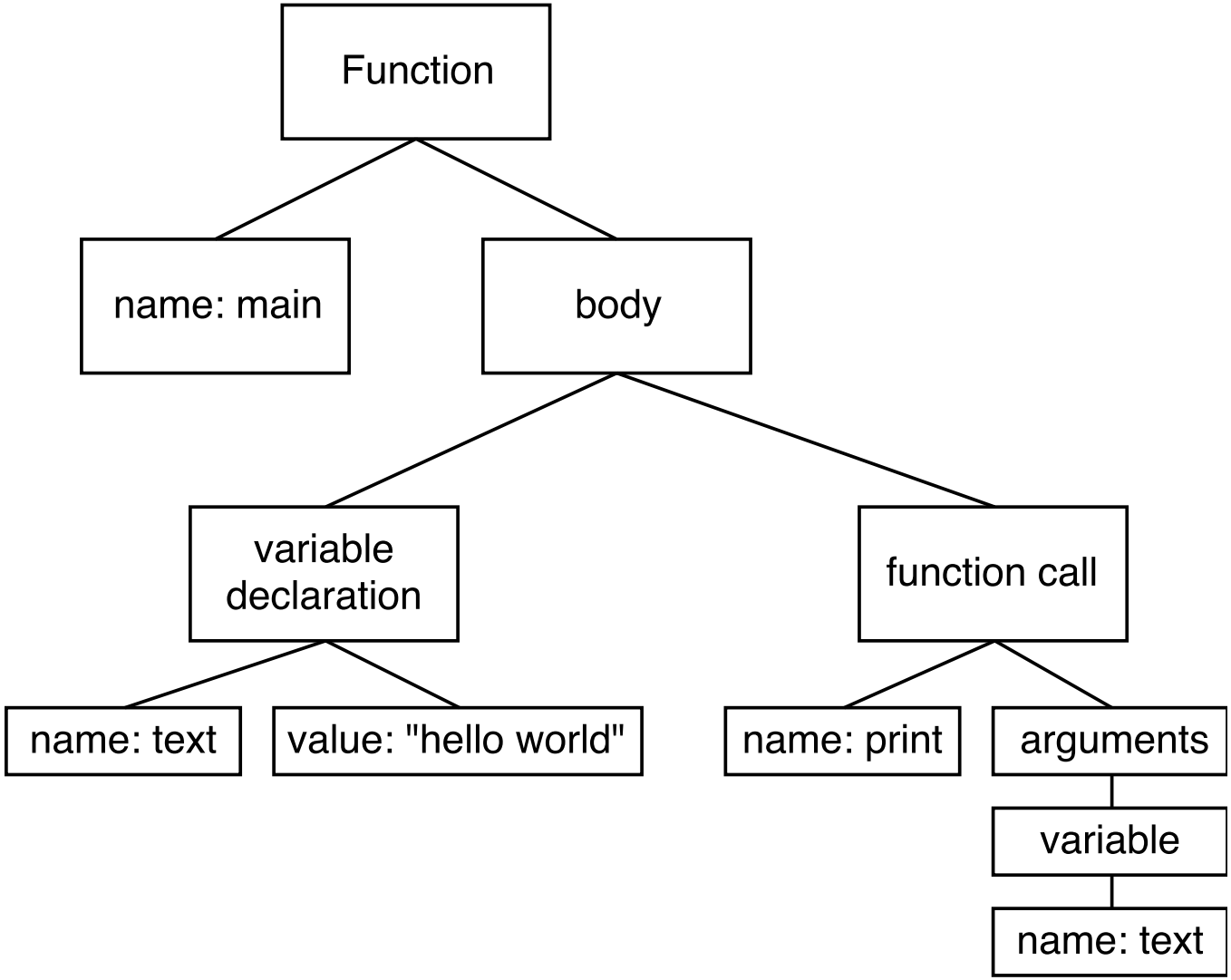

Figure 2: Abstract syntax tree

Program analysis: arity checking

```
def add(x,y):
    x + y

main = print(add(3,4,5))
```

Program analysis: variable bindings

```
def main():
    text = "hello world"
    print(foobar)
```

Program analysis: type checking

```
def isEmpty(list):
    length(list) == 0
main = print(isEmpty(4))
```

Program analysis: type checking

```
def isEmpty(list):
    length(list) == 0
main = print(isEmpty(4))

# Foo extends Bar
main = print((Bar) someFoo)
```

Program analysis: type checking

```
def isEmpty(list):
    length(list) == 0
main = print(isEmpty(4))

# Foo extends Bar
main = print((Bar) someFoo)

main = print(2 == "hello")
```

Program analysis: dead code

```
def main():
    text = "hello world"
    print(text)
    return
    throw "oh noes"
```

Optimization: inlining

```
def main():
    text = "hello world"
    print(text)
```

Optimization: inlining

```
def main():
    text = "hello world"
    print(text)

def main2():
    print("hello world") # Allocation omitted
```

Optimization: dependency analysis/floating in

```
def main():
    x = readLine
    y = expensive()
    if x == "secret"
        then print("hello world")
        else print(y)
```

Optimization: dependency analysis/floating in

```
def main():
    x = readLine
    y = expensive()
    if x == "secret"
        then print("hello world")
        else print(y)

def main2():
    x = readLine
    if x == "secret"
        then print("hello world")
        else
            y = expensive()
            print(y)
```

Optimization: CSE

```
def main():
    print(expensive())
    print(expensive())
```

Optimization: CSE

```
def main():
    print(expensive())
    print(expensive())

def main2():
    # Trading time for memory
    temp = expensive()
    print(temp)
    print(temp)
```

Optimization: constant folding, propagation

```
def main():
    x = 5
    print(1 + 2 + 3 + 4 + x)
```

Optimization: constant folding, propagation

```
def main():
    x = 5
    print(1 + 2 + 3 + 4 + x)

def main2():
    print(15)
```

Optimization: loop unrolling

```
def main():
    for i = 1...3:
        print(i)
```

Optimization: loop unrolling

```
def main():
    for i = 1...3:
        print(i)

def main2():
    print(1) # loop variable avoided
    print(2)
    print(3)
```

Optimization: loop fusion

```
def main():
    x = y = 0
    for i = 1...3:
        x++
    for j = 1..3:
        y++
    print(x + y)
```

Optimization: loop fusion

```
def main():
    x = y = 0
    for i = 1...3:
        x++
    for j = 1..3:
        y++
    print(x + y)

def main2():
    x = y = 0
    for i = 1..3
        x++
        y++
    print(x + y)
```

Optimization: bit twiddling

```
def abs(x):
    if x < 0
        then -x
        else x

main =
    x <- readLine
    if abs(x) >= 0
        then print "…"
        else print "Is this dead?"
```

Code generation

```
# High level
for i = 1..3:
    print(i)
print(123)
```

Code generation

```
# High level
for i = 1..3:
    print(i)
print(123)
```

```
; Low level
load            %eax 1          ; 1 in register EAX
load            %ebx 3          ; 3 in register EBX
loop:
add             %sp  %sp  1     ; Increase stack pointer…
load            %sp  %eax       ; …to push EAX (= 1) onto stack
call print                      ; Print top of stack (and popping it)
less            %tmp %eax %ebx  ; Check exit condition
add             %eax %eax 1     ; Increase loop counter
if0             %tmp loop       ; Loop if exit condition isn't met
add             %sp  %sp  1     ; Increase stack pointer…
load            %sp  123        ; …to push 123 onto the stack
call print                      ; Print top of stack (and popping it)
```

Code generation

Translate to processor opcodes

```
0x00              0x01 0x01
0x00              0x05 0x03
; loop:
0x02              0x02 0x02 0x01
0x00              0x02 0x01
0x07 <link: print>
0x03              0x04 0x01 0x05
0x02              0x01 0x01 0x01
0x06              0x04 0x05
0x02              0x02 0x02 0x01
0x00              0x02 0x7b
0x07 <link: print>
```

Linking

Put program alongside its libraries into a single file

```
0x0a00                  0x0a01 0x0a01
0x0a00                  0x0a05 0x0a03
0x0a02                  0x0a02 0x0a02 0x0a01
0x0a00                  0x0a02 0x0a01
0x0a07 0x03d4
0x0a03                  0x0a04 0x0a01 0x0a05
0x0a02                  0x0a01 0x0a01 0x0a01
0x0a06                  0x0a04 0x0a05
0x0a02                  0x0a02 0x0a02 0x0a01
0x0a00                  0x0a02 0x0a7b
0x0a07 0x03d4
```

Linking

Make it wholly unreadable

```
0a00 0a01 0a01 0a00 0a05 0a03 0a02 0a02
0a02 0a01 0a00 0a02 0a01 0a07 03d4 0a03
0a04 0a01 0a05 0a02 0a01 0a01 0a01 0a06
0a04 0a05 0a02 0a02 0a02 0a01 0a00 0a02
0a7b 0a07 03d4
```

Linking

Make it wholly unreadable

```
0a00 0a01 0a01 0a00 0a05 0a03 0a02 0a02
0a02 0a01 0a00 0a02 0a01 0a07 03d4 0a03
0a04 0a01 0a05 0a02 0a01 0a01 0a01 0a06
0a04 0a05 0a02 0a02 0a02 0a01 0a00 0a02
0a7b 0a07 03d4
```

```
...
00002ea0: c508 4a8b 742b 084a 8974 2808 4983 c508  ..J.t+.J.t(.I...
00002eb0: 4e8b 442b 084e 8944 2808 4983 c508 4a8b  N.D+.N.D(.I...J.
00002ec0: 542b 084a 8954 2808 4983 c508 4e8b 4c2b  T+.J.T(.I...N.L+
00002ed0: 084e 894c 2808 4983 c508 4e8b 542b 084e  .N.L(.I...N.T+.N
00002ee0: 8954 2808 4983 c508 4d39 dd0f 84ef 0a00  .T(.I...M9......
00002ef0: 004e 8b64 2b08 4e89 6428 0849 8b7c 1d10  .N.d+.N.d(.I.|..
00002f00: 4a89 7c28 104d 8b74 1d18 4e89 7428 1849  J.|(.M.t..N.t(.I
00002f10: 8b4c 1d20 4a89 4c28 2049 8b74 1d28 4a89  .L. J.L( I.t.(J.
...
```

Linking: add runtime

- Where is the main function?
- What to do after the program ends?
- Garbage collection
- Memory management

More advanced features

- JIT compilation
- Complex runtimes
- Complex parsers
- Smart GC
- Smart recompilation
- Hot swapping

```
> echo "main = putStrLn \"Hello, world!\"" > Hello.hs

> ghc -O2 Hello.hs
[1 of 1] Compiling Main (Hello.hs, Hello.o)
Linking Hello ...

> ./Hello
Hello, world!
```