OCaml Types

int

bool

int * int — tuple of two ints

int → int — fn from one int to another

int → int → int — fn taking two ints giving int

OR fn taking one int giving fn that takes an int and gives an int

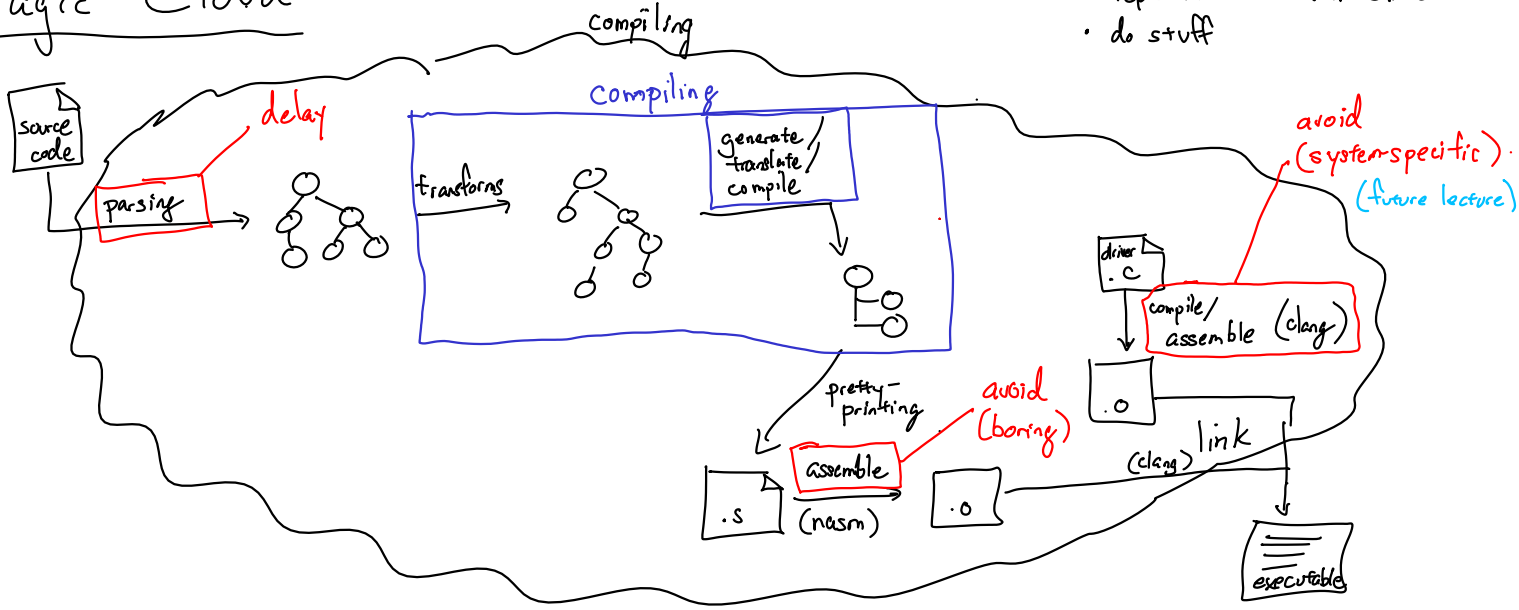
currying

(int → int) → int — fn taking in another fn from int to int and giving int

int * int → int — fn taking pair of ints giving int

Magic Cloud

- robust
- represent data with structure
- do stuff



Auklet

Step 1 Syntax: (EBNF)

```
<expr> ::= 0 | 1 | -1 | 2 | -2 | ...
         | <expr> + <expr> | <expr> - <expr> | <expr> * <expr>
         | after (<expr>) | before (<expr>)
         | (<expr>)
```

concrete syntax

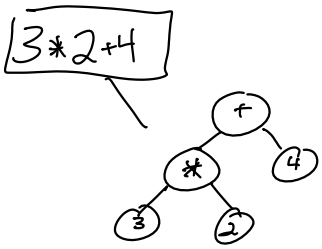
Step 2 Semantics:

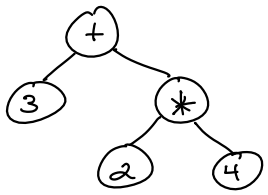
```
2 + 1  ⟹  3
after(3)  ⟹  4
before(2 - 1)  ⟹  0
3+2*4  ⟹  11
after (+)  ⟵ parse error
before(7) + 2
```

```
type expr =
  | EInt of int
  | EPlus of expr * expr
  | EMinus of expr * expr
  | ETimes of expr * expr
  ⋮
```

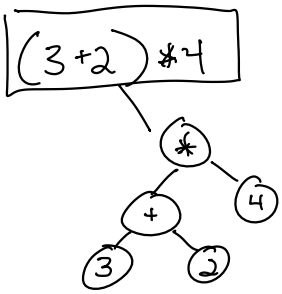
abstract syntax

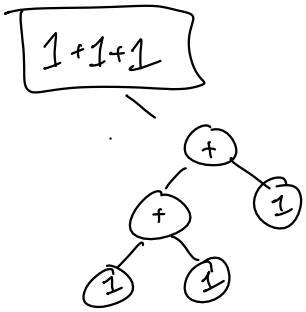
EPlus (EInt 3, ETimes (EInt 2, EInt 4))





abstract syntax trees (AST)






```
type instruction =
  | AsmMov arg * arg
```

Intel

```
mov rax, 4
```

AT&T

```
mov $4, %rax
```

```
type arg =
  | ArgReg of register
  | ArgConst of int
```

AsmMov(ArgReg RAX, ArgConst 4)

```
let compile (e : expr) : instruction list =
  match e with
  | EInt n ->
    [AsmMov (ArgRegister RAX, ArgConst n)]
  | EAfter e' ->
    compile e' @ [AsmAdd (ArgRegister RAX, ArgConstant 1)].
```

← returns a list of assembly instructions that puts the result of the expr into the RAX register

mov rax, n