PART II:

PROGRAMMING LANGUAGES

WRONG idea: Python (or any other general purpose PL)

- too large and general
- complicated syntax
- lots of program aliasing
+ lots of bad training data online

DSL: Domain Specific Language

- Designed for a specific class of tasks
- Defined by a family of primitives to be combined

$\Longrightarrow$ FUNCTIONAL PROGRAMMING

also called "component-based synthesis"

EXAMPLE DSL

list of primitives :

their types

sort : $list(int) \longrightarrow list(int)$

map : $(t_0 \longrightarrow t_1) \longrightarrow list(t_0) \longrightarrow list(t_1)$

succ : $int \longrightarrow int$

⋮

TYPES

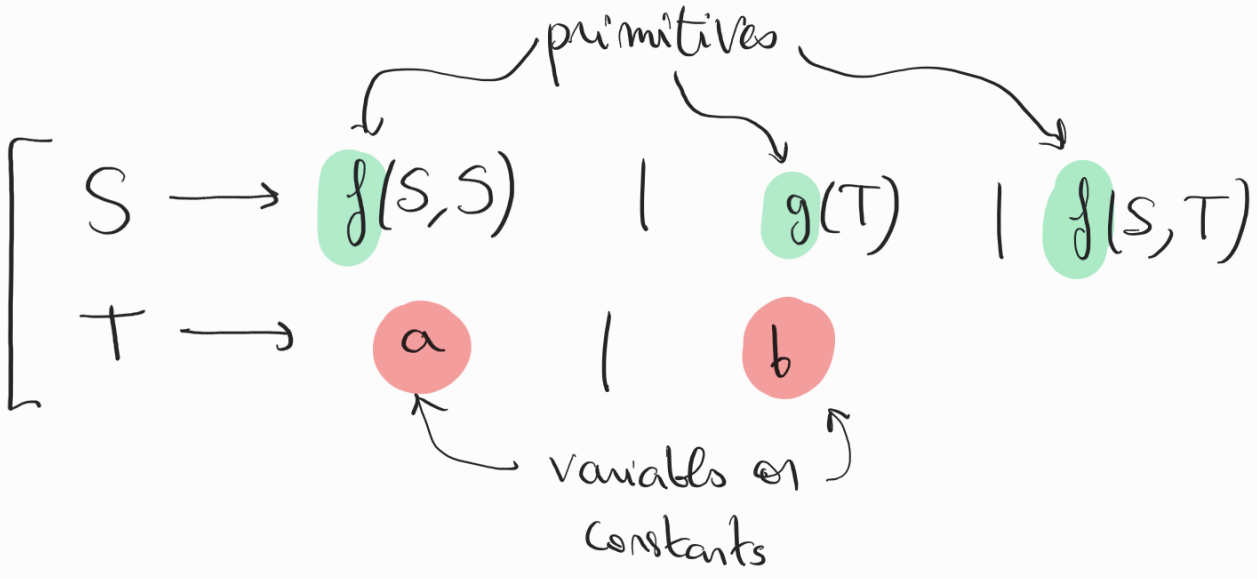
atomic types : int, bool, string, ...

generic types : $t_0, t_1, \ldots$

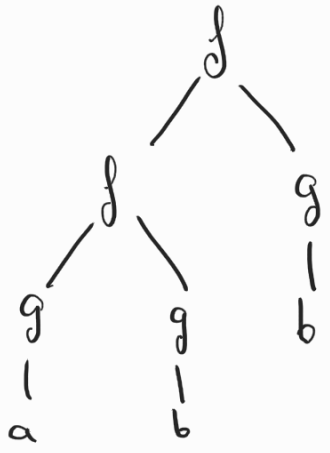
constructor : arrows, list, set, ...

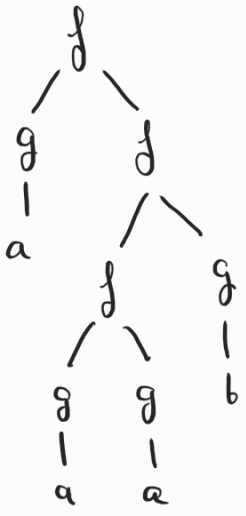
(the treatment of polymorphic types is an interesting one not discussed here.)

CONTEXT FREE GRAMMARS (for trees)



Example trees generated by CFG:





$\mathcal{L}(G)$: set of trees generated by the CFG G

List of primitives

Add: int → int → int
Double: int → int
Halve: int → int
IfThenElse: bool → int → int → int
Even: int → bool
Equal: int → int → bool
0: int **False**: bool
1: int **True**: bool

type request:
int → int

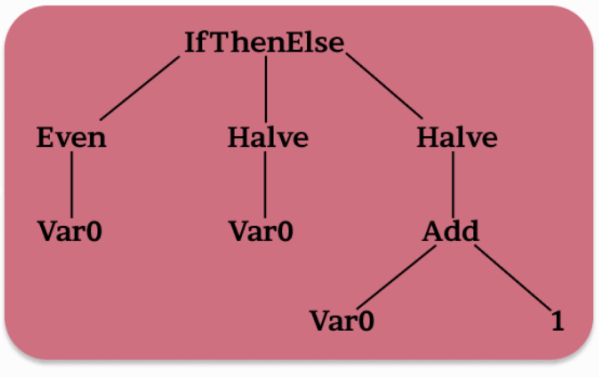
Compilation

Grammar

int → **Add**(int, int)
int → **Double**(int)
int → **Halve**(int)
int → **IfThenElse**(bool, int, int)
int → **0**
int → **1**
int → **Var0**
bool → **Even**(int)
bool → **Equal**(int, int)
bool → **True**
bool → **False**

Usual (Python-style) syntax

```
if Even(var0):
    Halve(var0)
else:
    Halve(Add var0 1)
```



Equivalent AST representation

DSL

Sort : list(int) → list(int)

map : $(t_0 \to t_1) \to list(t_0) \to list(t_1)$

succ : int → int

⋮

CFG G

list(int) → Sort (list(int))

list(int) → map (int → int , list(int))

list(int) → input

int → int → succ ()

⋮

$\mathcal{L}(G)$ = all correctly typed programs

AFTER COMPILATION :

We have a CFG representing the class of programs of interest

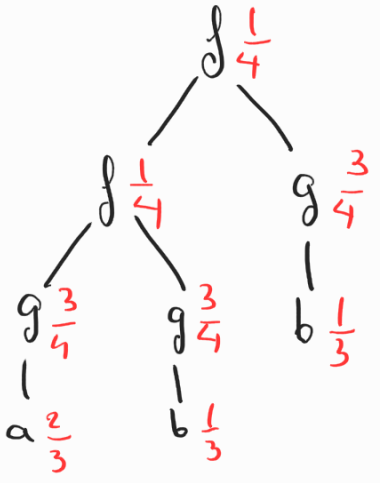
How does that connect to ML part ?

PROBABILISTIC CONTEXT FREE GRAMMARS

$$\text{CFG} \begin{cases} S \longrightarrow f(S,S) + g(T) \\ T \longrightarrow a + b \end{cases}$$

$$\text{PCFG} \begin{cases} S \longrightarrow \frac{1}{4} f(S,S) + \frac{3}{4} g(T) \\ T \longrightarrow \frac{2}{3} a + \frac{1}{3} b \end{cases}$$

Example :

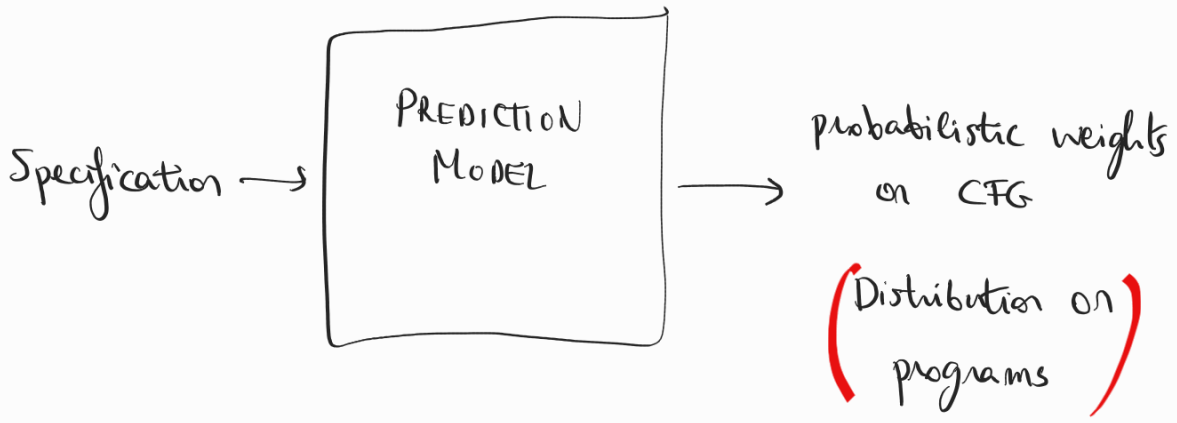


probability :

$$= \frac{1}{4} \cdot \frac{1}{4} \cdot \frac{3}{4} \cdot \frac{2}{3} \cdot \dots$$

A PCFG induces a distribution over trees $\simeq$ programs

BACK TO ML: PREDICTIONS



STATE OF AFFAIRS

- we construct a CFG representing the class of programs
- we learn probabilistic weights to obtain a PCFG representing a distribution $\mathcal{D}$ over programs

$$\mathbb{P}(\text{program} \mid \text{examples})$$

- next (and last) step: SEARCH THROUGH $\mathcal{D}$

NEURAL GUIDED PROGRAM SYNTHESIS