Reg Ex

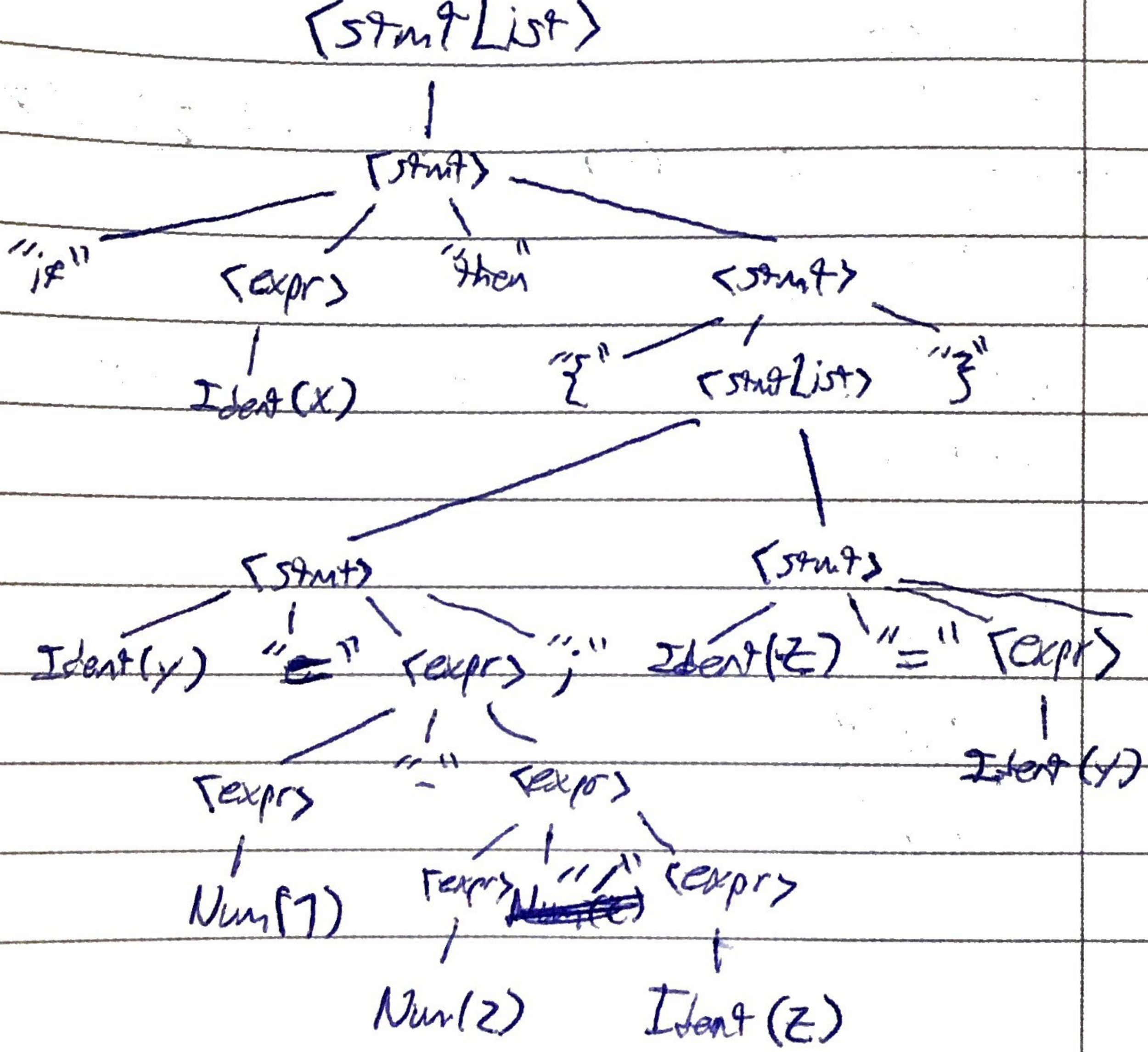
$a\ a^*\ b^*\ (c\,|\,\varepsilon)\ d$

Grammar

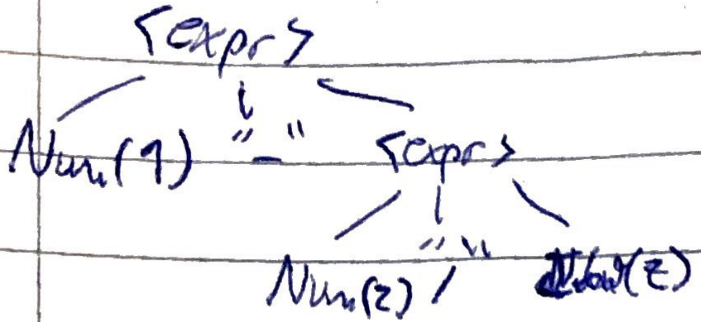
```
expr -> "(" expr List ")"
      | Num
List -> ";" expr list
      | ""
```

Parse Trees / ASTs

1.

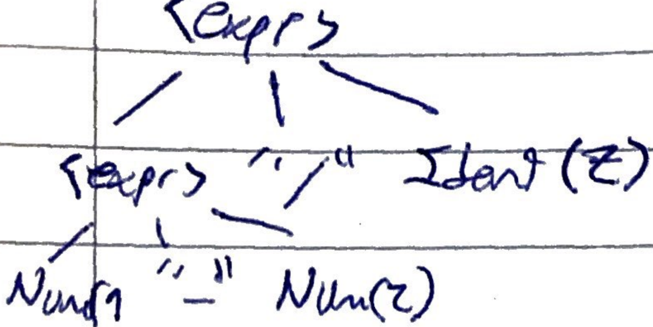


2. Unambiguous. Mostly it's well defined, but the expression 1 - 2 /z could be parsed as



aka: 1 - (2/z)

or



aka: (1-2) / z

which would change the order of evaluation.

3. a) ";"

b) "{"

c) "}"

Static vs. Dynamic Type Checking

Statically typed languages do type-checking at compile time, whereas dynamically typed languages do it at runtime.

Statically typed language allow us to see type errors before we even compile, and have the added benefit of ~~tooling~~ increased readability. Examples are Java, Typescript, and C.

Dynamically typed languages are more flexible and allow variables to be of multiple types. They are also easier to implement but are ~~more~~ prone to type errors during runtime. Examples are Python and Javascript.

Well-Typed Terms and Type Checking

1. Not well typed. First term is of type int -> int.

   Second term is also int -> int.

   We can't apply int -> int to int, which we try to do.

2. Well-typed. We get type:

   int -> int

3. Not well typed. We try to pass

   g: Bool -> Int as

   f: Int -> Bool