

# Semantic Actions

# taking action

- Building a parse tree tells us the **syntax** of a program
  - Whether it is “grammatically correct”
  - What structures are used to build up the program
- But we are interested in the **semantics** of the program
  - When we recognize a structure, we want to build up some *meaning* for our program based on what that structure is

prog  $\rightarrow$  decls stmtlist

decls  $\rightarrow$  decl decls

decls  $\rightarrow$   $\lambda$

decl  $\rightarrow$  TYPE ID

stmtlist  $\rightarrow$  stmt stmtlist

stmtlist  $\rightarrow$   $\lambda$

stmt  $\rightarrow$  ID := NUM

stmt  $\rightarrow$  ID := ID + NUM

# taking action

```
int x
x = 0
x = x + 7
```

prog  $\rightarrow$  decls stmtlist

decls  $\rightarrow$  decl decls

decls  $\rightarrow$   $\lambda$

decl  $\rightarrow$  TYPE ID

stmtlist  $\rightarrow$  stmt stmtlist

stmtlist  $\rightarrow$   $\lambda$

stmt  $\rightarrow$  ID := NUM

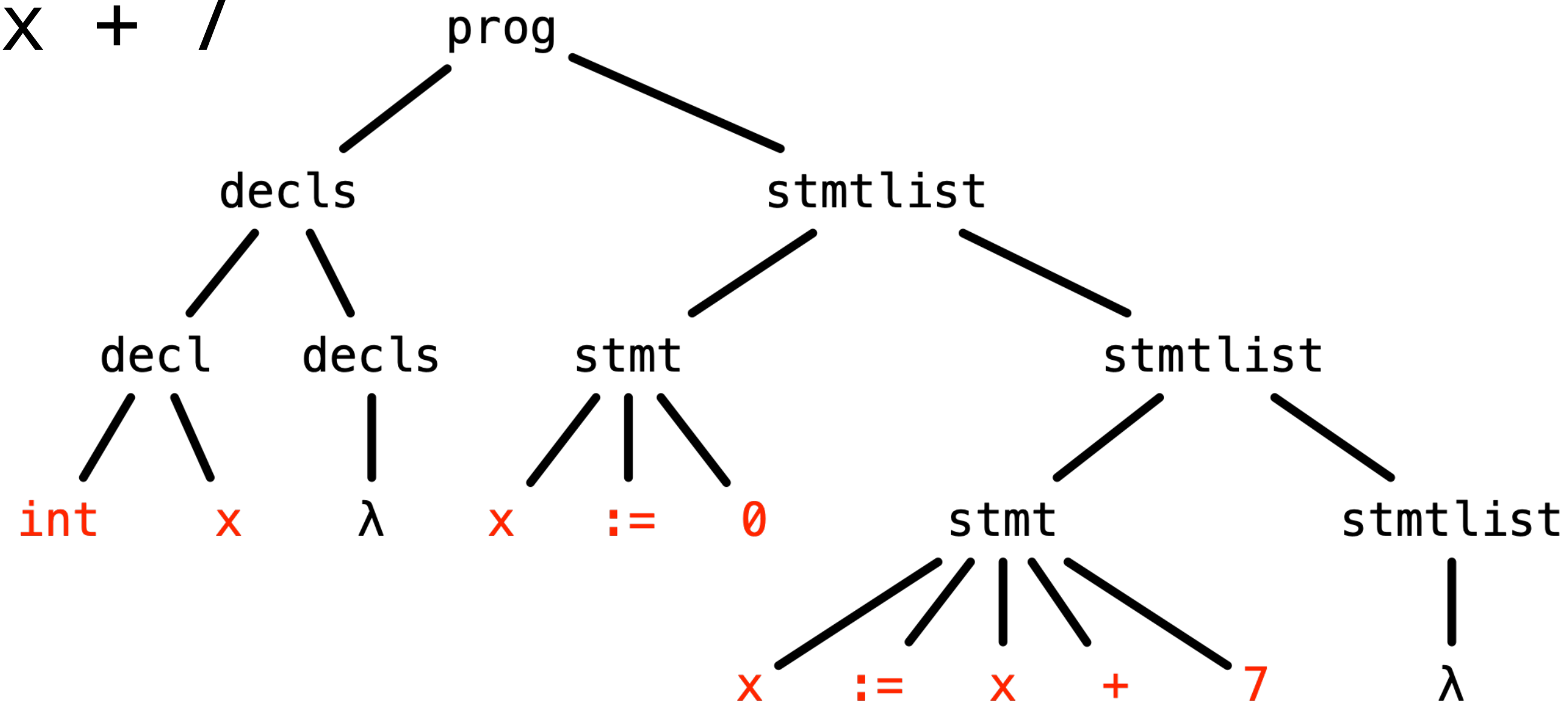
stmt  $\rightarrow$  ID := ID + NUM

# taking action

int x

x = 0

x = x + 7



prog → decls stmtlist

decls → decl decls

decls → λ

decl → TYPE ID

stmtlist → stmt stmtlist

stmtlist → λ

stmt → ID := NUM

stmt → ID := ID + NUM

# taking action

- What kinds of actions might we want to take?
  - Build up internal information in the compiler like a **symbol table**
  - Build up intermediate representation of program like an **abstract syntax tree**
- With a symbol table plus an abstract syntax tree, we can easily generate code for programs

prog  $\rightarrow$  decls stmtlist

decls  $\rightarrow$  decl decls

decls  $\rightarrow$   $\lambda$

decl  $\rightarrow$  TYPE ID

stmtlist  $\rightarrow$  stmt stmtlist

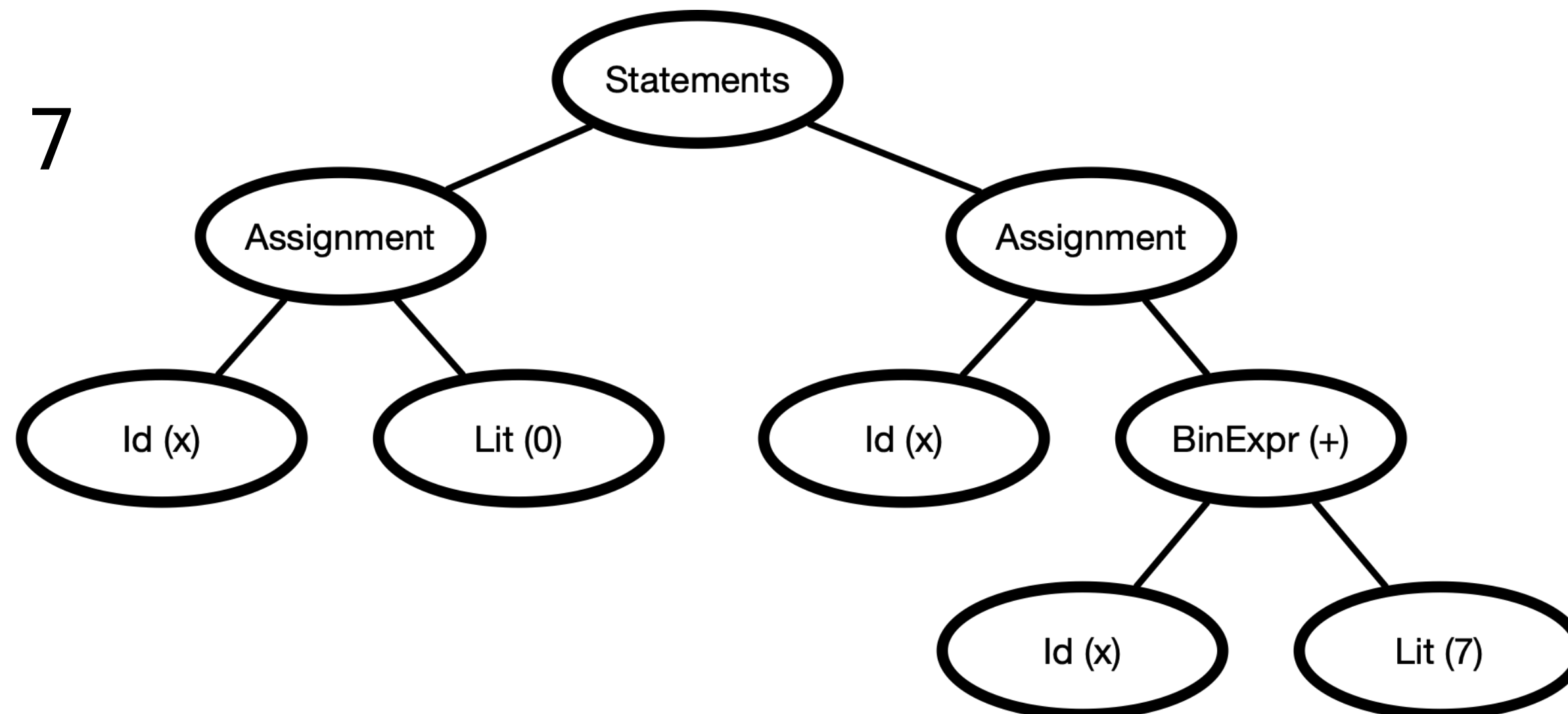
stmtlist  $\rightarrow$   $\lambda$

stmt  $\rightarrow$  ID := NUM

stmt  $\rightarrow$  ID := ID + NUM

# taking action

```
int x
x = 0
x = x + 7
```



GLOBAL  
<x, int, 0x4000>

prog  $\rightarrow$  decls stmtlist

decls  $\rightarrow$  decl decls

decls  $\rightarrow$   $\lambda$

decl  $\rightarrow$  TYPE ID

stmtlist  $\rightarrow$  stmt stmtlist

stmtlist  $\rightarrow$   $\lambda$

stmt  $\rightarrow$  ID := NUM

stmt  $\rightarrow$  ID := ID + NUM

# adding actions to parser

- Recursive descent parsers make it easy to take action
- As you match tokens and non-terminals, **return information along with the rest of the string**
- Use that information to **recursively build up the semantic information you want**

```
Context decl(string prog) {  
    TypeContext type = matchINT(prog); //match TYPE  
    IdentContext id = matchID(type.rest); //match ID  
    sym = new Symbol(type.text, id.text); //make symbol  
    return new DeclContext(sym, id.rest); //return info  
}
```

# adding actions to parser

- Recursive descent parsers make it easy to take action
- As you match tokens and non-terminals, **return information along with the rest of the string**
- Use that information to **recursively build up the semantic information you want**

```
Context prog(string prog) {  
    DeclContext ds = decls(prog); //match decls  
    StmtListContext ss =  
        stmtlist(ds.rest); //match stmtlist  
    symTable = buildSymbolTable(ds.declList);  
    ast = buildAST(ss.stmts);  
}
```



**next: adding actions in ANTLR**