Languages & Grammars

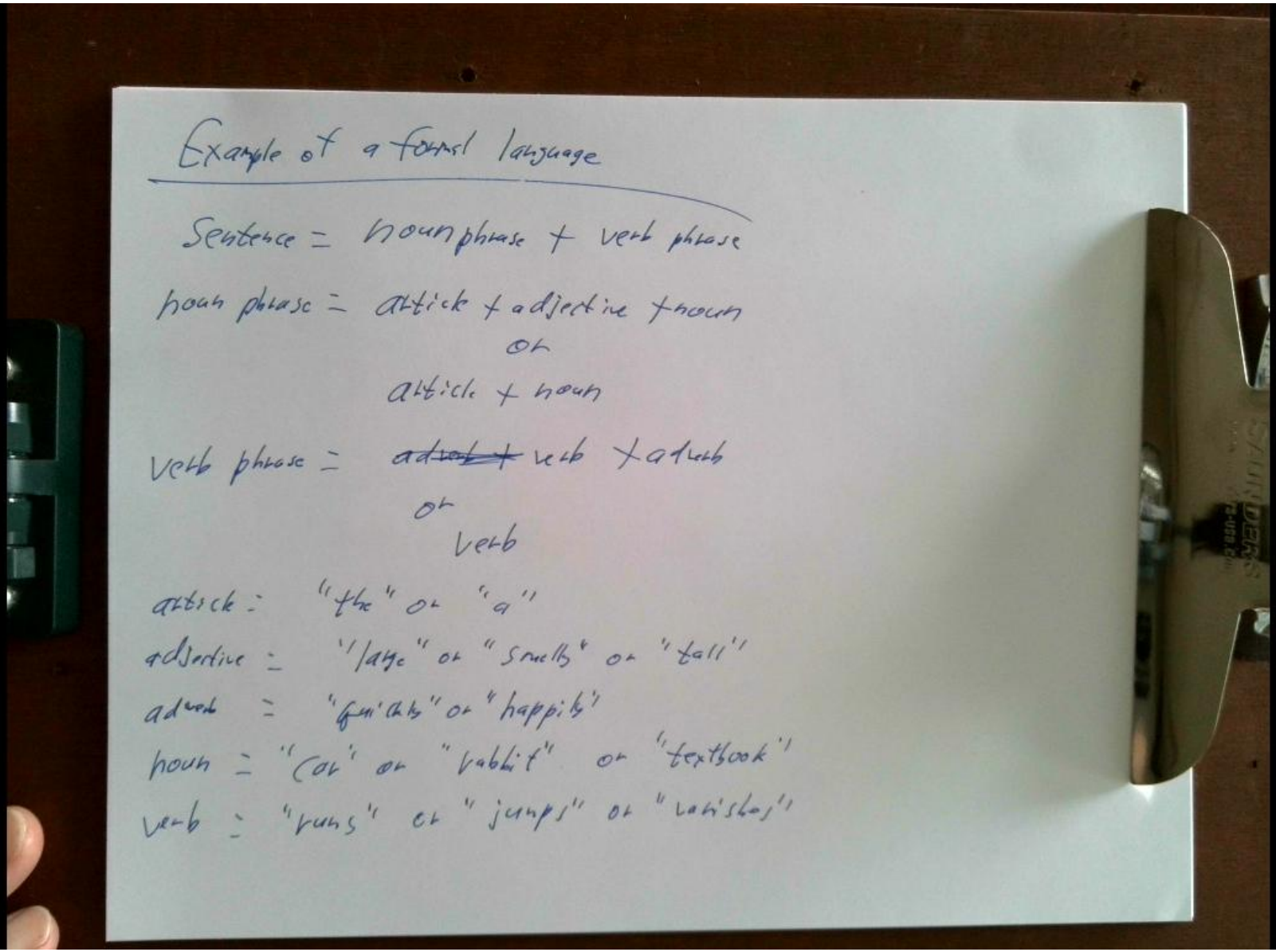
Syntax vs. Semantics

↑ Structure

↑ Meaning.

```
if (5 > 10) {
    x = x;
    x = x + 1 - 1;
    x = x + 0;
} else {
    x = x;
}
```

```
if (x = 0) {
    ~
    —
}
int *p;
int i;
p = i;
```



Example of a formal language

Sentence = noun phrase + verb phrase

noun phrase = article + adjective + noun
or
article + noun

verb phrase = ~~adverb +~~ verb + adverb
or
verb

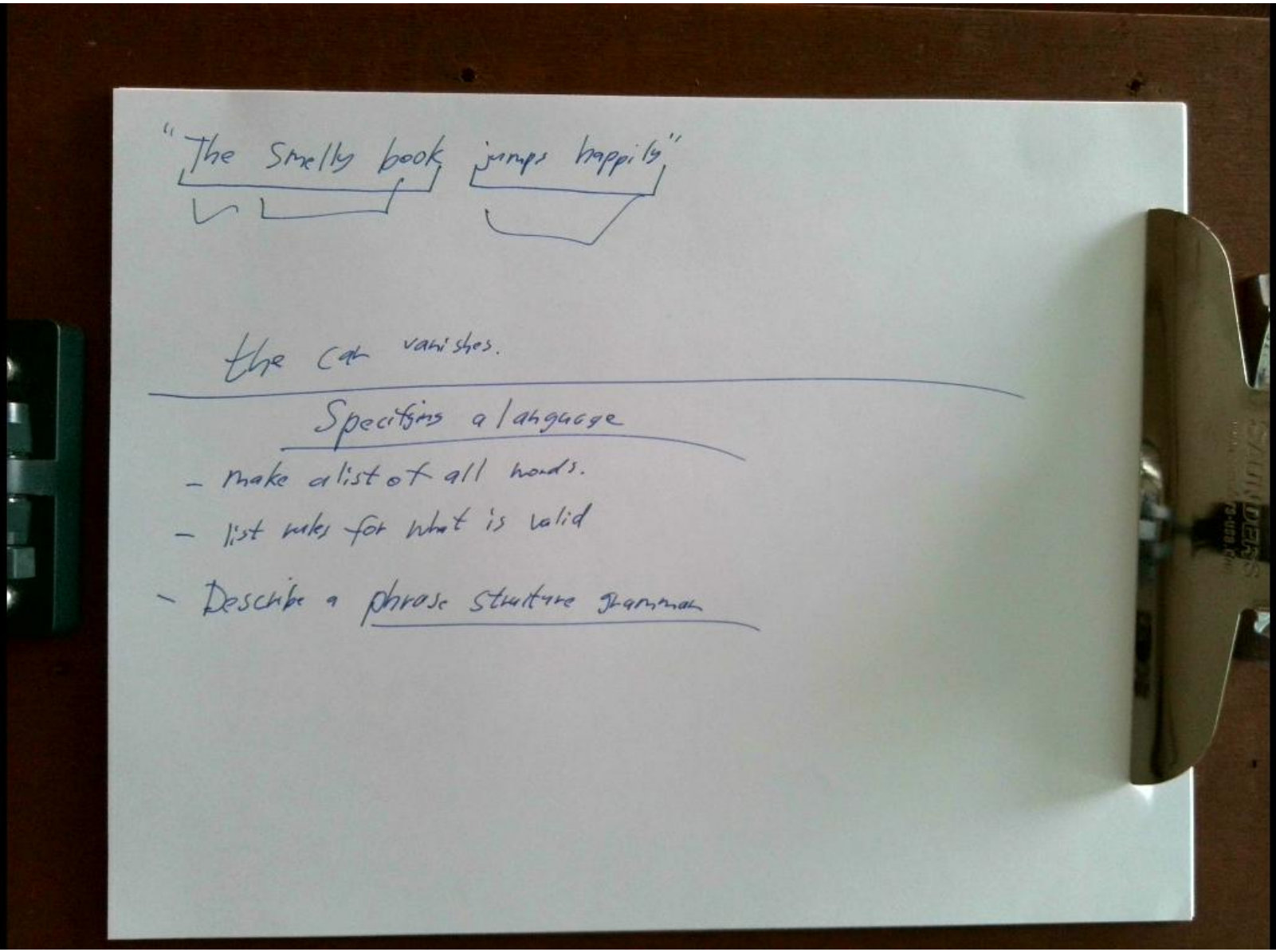
article : "the" or "a"

adjective : "large" or "small" or "tall"

adverb : "quickly" or "happily"

noun : "car" or "rabbit" or "textbook"

verb : "runs" or "jumps" or "vanishes"


"The Smelly book jumps happily"
the cat vanishes.
Specifying a language
- make a list of all words.
- list rules for what is valid
- Describe a phrase structure grammar

Definitions

A vocabulary V is a finite set of symbols, $\neq \phi$

A word (or, sometimes, a ~~set~~ sentence) over V is a finite-length string of symbols from V.

λ = empty string (word of length 0)

V^* = {set of all words over V}

A language over V is a subset of V^*.

A phrase structure grammar $G = (V, T, S, P)$

V = a vocabulary $T = \{\text{terminal symbols}\} \subseteq V$

S = start symbol $\in V$

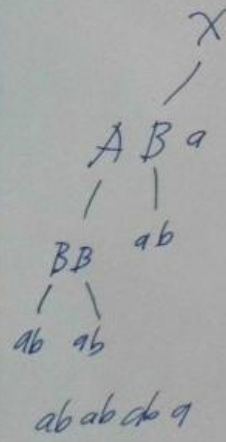
$P = \{\text{production rules}\}$ ← rules for replacing one word with another

EXAMPLES

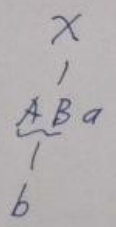
$G = (V, T, S, P)$

$V = \{a, b, A, B, X\}$ $T = \{a, b\}$ $S = X$

$P = \{ X \to ABa, A \to BB, B \to ab, AB \to b \}$



ab ab ab a

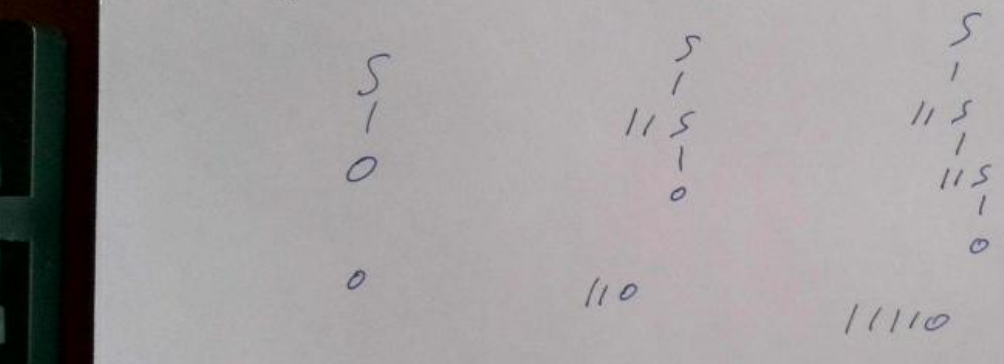


ba

$G = (V, T, S, P)$

$V = \{S, 0, 1\}$ $T = \{0, 1\}$ $S = S$

$P = \{S \to 11S,\ S \to 0\}$



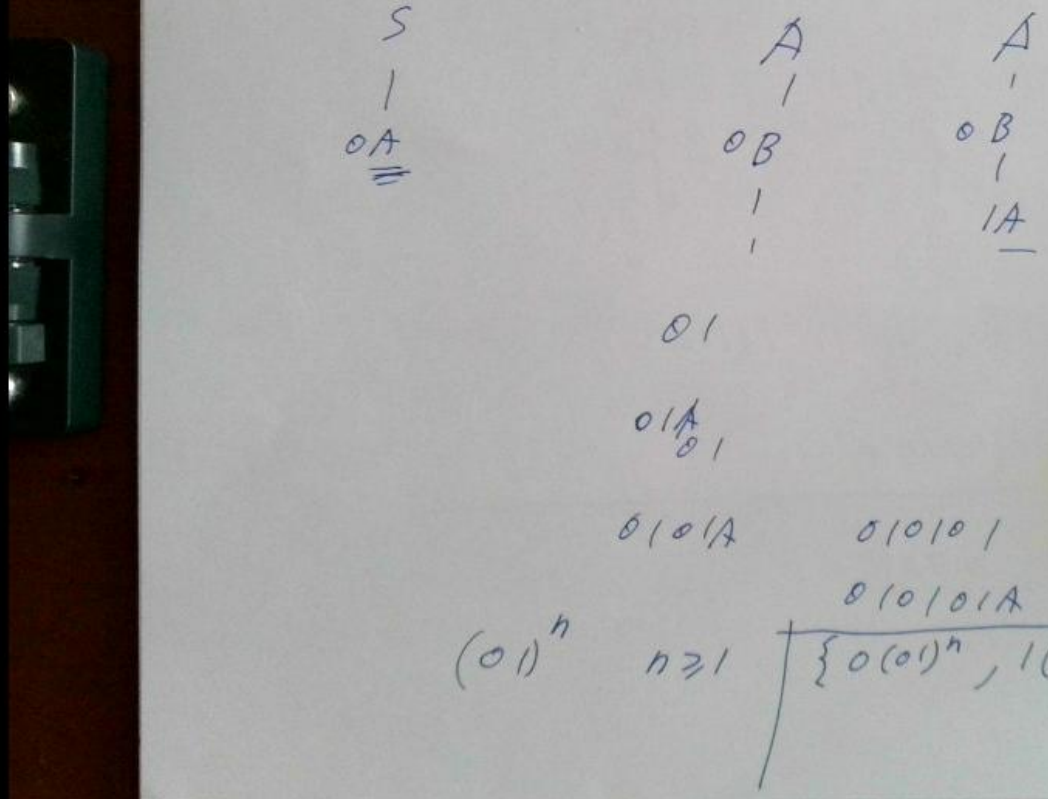
0 110 11110

1111110 111111110 . . .

$\{(11)^n 0 \ , n \geq 0\}$

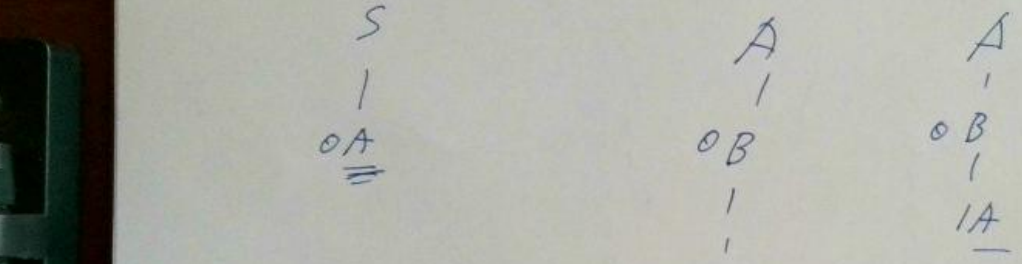
$V = \{0, 1, A, B, S\}$ $T = \{0, 1\}$ $S = S$

$P = \{ S \to 0A,\ S \to 1A,\ A \to 0B,\ B \to 1A,\ B \to 1 \}$



$V = \{0, 1, A, B, S\}$ $T = \{0, 1\}$ $S = S$

$P = \{ S \to 0A,\ S \to 1A,\ A \to 0B,\ B \to 1A,\ B \to 1 \}$



01

$01A01$

$0101A$ 01010

$0101010A$

$(01)^n$ $n \geq 1$ $\{0(01)^n, 1(01)^n \quad n \geq 1\}$

0010101

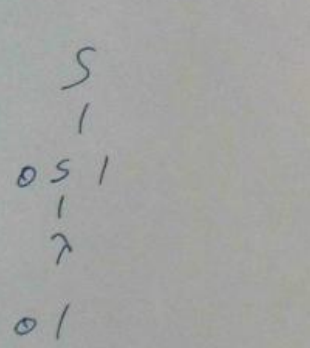
Constructing a grammar

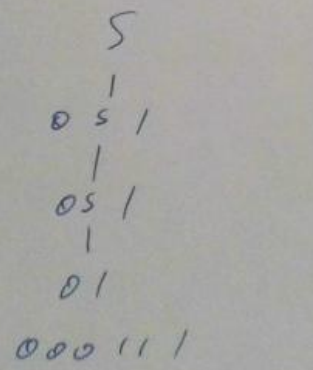
Suppose $L(G) = \{\lambda, 01, 0011, 000111, 00001111, \ldots\}$

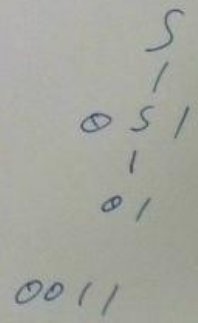
$V = \{0, 1, S\}$ $\quad T = \{0, 1\}$ $\quad S = S$

$P = \{S \to \lambda, (S \to 01), \cancel{(S \to 0011, S \to 000111)}$

$S \to 0S1\}$







$L(G) = \{0^m, m \geq 0\}$

$V = \{0, S\}$ $T = \{0\}$ $S = S$

$P = \{S \to 0, S \to \lambda, S \to 0S\}$

$L(G) = \{0^n, n \geq 1\}$

$S \to 0J$

$J \to \lambda, J \to 0J$

$L(G) = 1^n, n \geq 1$

$S \to 1K$

$K \to \lambda, K \to 1K$

$L(G) = \{0^m 1^n, \; mn \geq 1\}$

$V = \{S, 0, 1$ $\quad T = \{0, 1\}$ $\quad S = S$

$P = \{S \to AB, \; A \to 0J, \; J \to \lambda, \; J \to 0J,$
$B \to 1K, \; K \to \lambda, \; K \to 1K\}$

$P = \{S \to 0S \;\; S \to 01A, \; A \to 1A, A \to \lambda\}$?

Let $w_0 = L\, z_0\, R$

Let $w_1 = L\, z_1\, R$

(w_0, w_1, L, R, z_0, z_1 are strings)

$G(V, T, S, P)$

If $z_0 \to z_1 \in P$ then

we say "w_1 is directly derivable from w_0"

$$w_0 \Rightarrow w_1$$

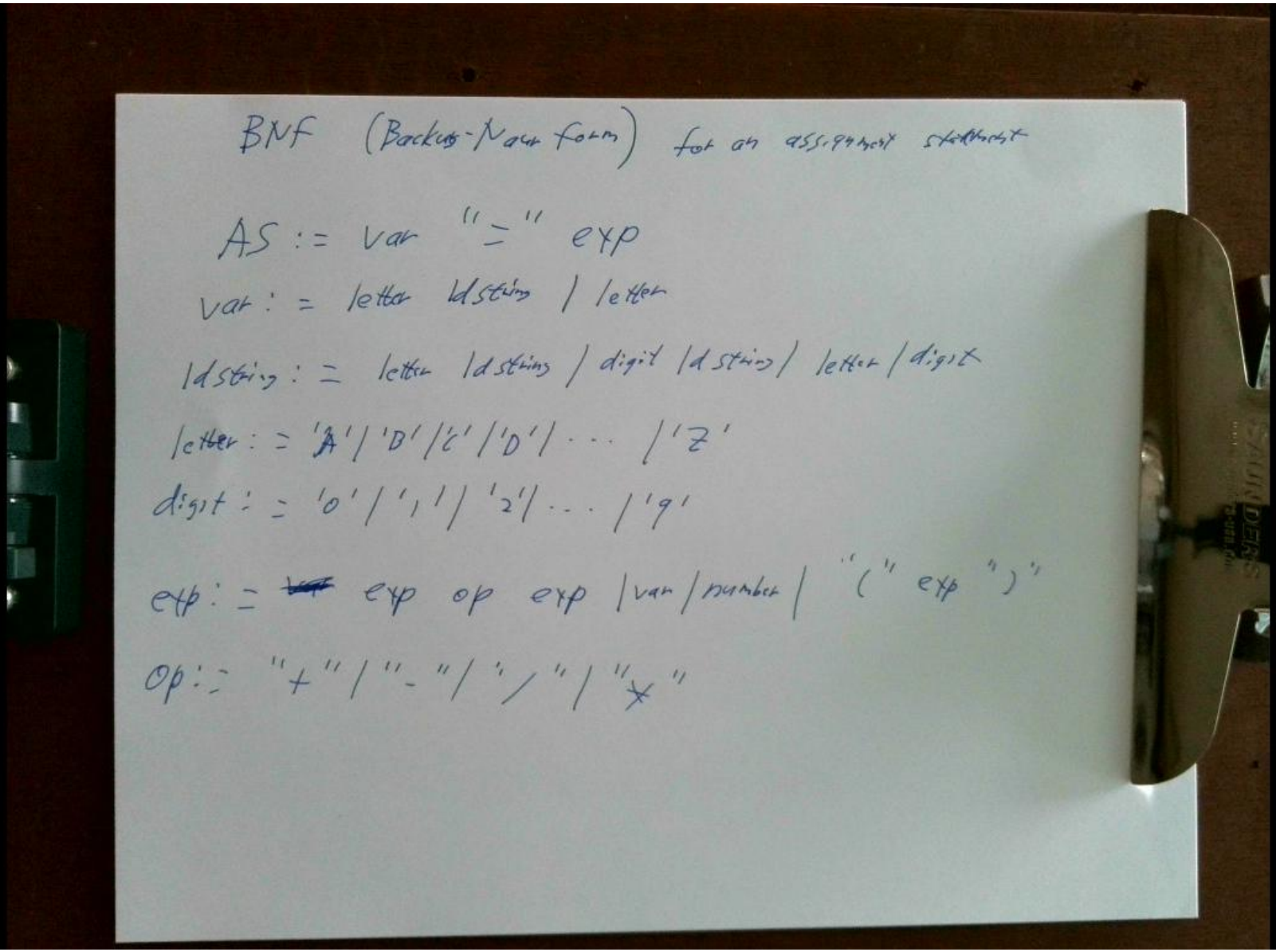
If $w_0 \Rightarrow w_1 \Rightarrow w_2 \Rightarrow w_3 \Rightarrow \dots \Rightarrow w_n$ then

we say "w_n is derivable from w_0"

$$w_0 \overset{*}{\Rightarrow} w_n$$

$L(G)$ is "the language generated by G" = {strings of terminal symbols derivable from S}

$$L(G) = \{w \in T^* : S \overset{*}{\Rightarrow} w\}$$


BNF (Backus-Naur form) for an assignment statement
AS := var "=" exp
var := letter idstring | letter
idstring := letter idstring | digit idstring | letter | digit
letter := 'A' | 'B' | 'C' | 'D' | ... | 'Z'
digit := '0' | '1' | '2' | ... | '9'
exp := exp op exp | var | number | "(" exp ")"
op := "+" | "-" | "/" | "*"

The syntax of C in Backus-Naur Form

```
<translation-unit> ::= {<external-declaration>}*

<external-declaration> ::= <function-definition>
                         | <declaration>

<function-definition> ::= {<declaration-specifier>}* <declarator> {<declaration>}* <compound-statement>

<declaration-specifier> ::= <storage-class-specifier>
                          | <type-specifier>
                          | <type-qualifier>

<storage-class-specifier> ::= auto
                            | register
                            | static
                            | extern
                            | typedef

<type-specifier> ::= void
                   | char
                   | short
                   | int
                   | long
                   | float
                   | double
                   | signed
                   | unsigned
                   | <struct-or-union-specifier>
                   | <enum-specifier>
                   | <typedef-name>

<struct-or-union-specifier> ::= <struct-or-union> <identifier> { {<struct-declaration>}+ }
```