ISO Standard for EBNF

```
"terminal"    'terminal'    (grouping)
[option]    {repetition}    *repitition
exception: A-B (A not in B)    | OR
concatenation: A,B
rule: LHS = RHS rule

syntax = syntax rule, {syntax rule}
syntax rule = meta id, '=', defns list, ';';
defns list = defn, {'|', defn};
defn = term, {',', term};
```

What can go wrong?

Tokens

Character confusion:

```
int microsoft = 27;
int microsoft = 28; (with cyrillic 0)
itn float = 29;
```

Greedy tokenization:

```
int -> [int]egral - 27;
if (a <= b)
    a = b + + + + + c;
    a = ((b++) ++) + c);
    a = ((++b) + + c);
```

Long tokens:

- give people ways of writing shorter tokens that mean the same thing
  - multiple strings or backslash newline
  - comments and whitespace

```
grammars (context free | BNF) = (token set (finite), nonterminals (finite), start symbol (a nonterminal), rule set (at least 1 rule must have start symbol at LHS))
```

```
rule: nonterminal(LHS) = finite sequencing of symbols(RHS)

uppercase - nonterminal
lowercase - terminal

ex: S -> a
    T -> b <- useless rule

S is the start symbol
T is a useless rule
```

Useless rules:

- LHS is unreachable from start symbol
- always results in blind alley

```
S -> a
S -> bT
T -> cT <- blind alley

S -> a
S -> bS
S -> ba <- redundant

S ->
S -> T
T -> aT <- can be removed
T -> Ta
T -> a
```

Extra constraints grammar doesn't capture

```
S -> N V.
N -> n
N -> adj n
```

```
V -> v
V -> V adj
```

Issues with this:

- dogs bark. (correct)
- Maxwell meows loudly. (correct)
- Maxwell meow loudly. (incorrect)

Fix:

```
S -> SN SV.
S -> PN PV.
SN -> sn
SN -> adj SN
PN -> pn
PN -> adj PN
...
```

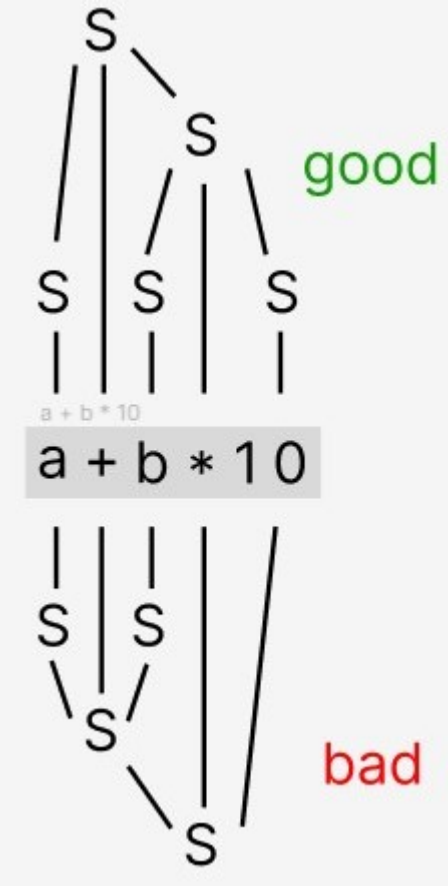
This fixed the buggy error but made grammar more complex

Ambiguity

```
S -> S + S
S -> S * S
S -> id
S -> (S)
S -> num
```

Issue:

- there are two ways to parse `a = b * 10`



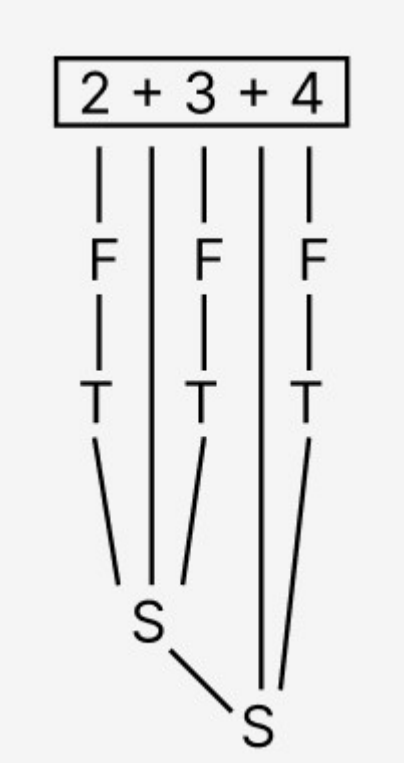
- the top follows PEMDAS while the bottom doesn't

```
S -> T + T
S -> T
T -> F * F
F -> id
F -> num
F -> (S)
```

Issue:

- we can't do 2 + 3 + 4

- this is because there is no `S -> S + T`




```
Fix:
S -> T + T ==> S = S + T
S -> T
New ==> T -> T * F
T -> F
F -> id
F -> num
F -> S
```

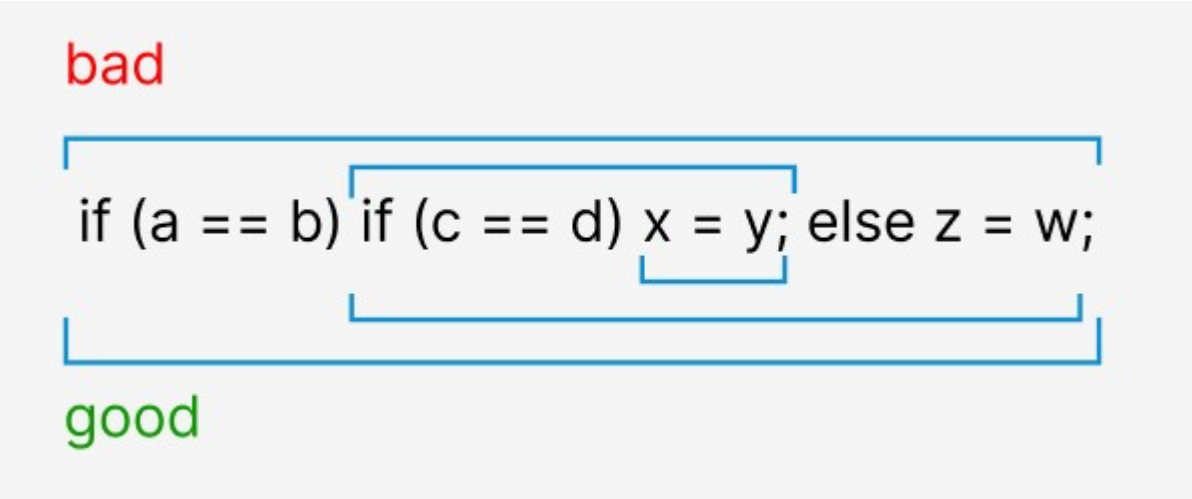
- fixed precedence and associativity

C standard grammar

```
stmt:
    ;
    expr;
    break;
    continue;
    return expr (optional ;)
    goto DI;
    while (expr) stmt
    do stmt while (expr); (parens not needed)
```

```
    for (expr (optional ;) expr (optional ;) expr (optional ;)) stmt
    switch (expr) stmt
    if (expr) stmt
    if (expr) stmt else stmt
```

Dangling else:




```
Good:
if (a == b)
    if (c == d)
        x = y;
    else
        z = w;

Bad:
 if (a == b)
 if (c == d)
     x = y;
where does the else part go???
```

```
grammar -> generate sample sentences
token sequence

grammar + token sequence ==> Please generate; a parse tree
```

Parsing - 3 main issues

1. Recursion in parser (relatively easy if tech is recursive)

2. Disjunction (OR)
3. Concatenation (,) S -> A --> matcher + acceptor