Last month, I have studied several ways of grammar checker. Mainly, there are two basic ways. Traditionally, we have rule-based methods like LanguageTool and Link Grammar. Also, we can use some AI models like SyntaxNet.

I have dived into the algorithm of Link Grammar.

HOW LINK GRAMMAR WORKS：

Terminology for link grammar:

Basically, there are several terminologies for a word. For example,

dog: {@A+} & Ds- & {@M-} & (Ss+ or SIs- or O- or J-);

The capital English character represent a syntax component, + means it allows for such at its right position and - means it allows for such at its left position.

Then we can transfer this dict into a different version easier for analyzing:

$$((L_1, L_2, \ldots, L_m)\ (R_n, R_{n-1}, \ldots, R_1))$$

L represent all possible components at its left and R represent the right.
Then we call L1 a connector, the whole is called a disjunct for a word.

For example, this is a disjunct $d = ((D,O)\ ())$ for a word. It represents that this word needs an 'O' at its left and a 'D' at the left of O. It needs nothing at its right.

There are also some pointers in this list.

For example, Left[d] point to 'O' and right[d] points to nothing, we name right[d] = NIL
We also have the next pointer in the list. Next[left[d]] = 'D' and next[next[left[d]]] = NIL

Build Link Solution:

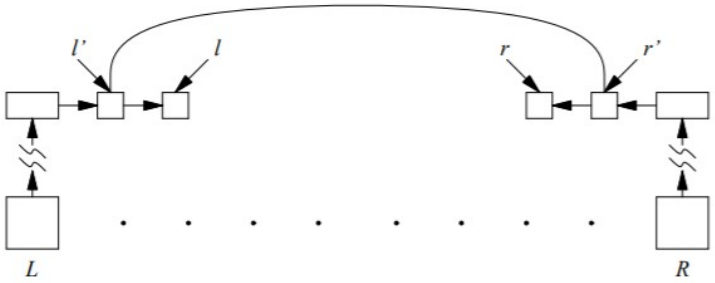
Now we consider building a link solution in a sentence.
The link has following rules:
1. Different links should not cross.
2. Links should link all words in a sentence.
3. No two link should link the same pair of words.
4. When the connector in the formula are from left to right, the word they link to are from near to far.

If all links in the sentence follow these rules, we can determine that this sentence is correct.

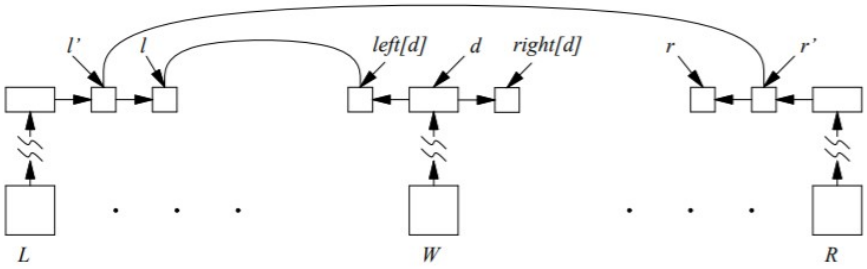
Parse the sentence:

we parse the sentence from the first word to the last word in the sentence. As in the picture, connector l' and r' are connected.

If no word is in L and R, this link is valid if l = NIL AND r = NIL.



If there are other words between L and R, we can build another link between them, using the connector l and r.



Suppose the connector left[d] of the word W are connected with l, we will continue to do solve two problem:

1. Whether link between L and W are valid, using connector next[l] and next[left[d]].
2. Whether link between W and R are valid, using connector right[d] and r.

Finally, all processes are done, we can count all possible link solutions in the sentence. If the result > 0, we can say that this sentence is valid.

This is the pseudocode of the process.

```
PARSE
 1  t ← 0
 2  for each disjunct d of word 0
 3       do  if left[d] = NIL
 4             then t ← t + COUNT(0, N, right[d], NIL)
 5  return t
```

```
COUNT(L, R, l, r)
 1  if L = R + 1
 2     then if l = NIL and r = NIL
 3             then return 1
 4             else  return 0
 5     else  total ← 0
 6           for W ← L + 1 to R − 1
 7              do  for each disjunct d of word W
 8                     do  if l ≠ NIL and left[d] ≠ NIL and MATCH(l, left[d])
 9                            then leftcount ← COUNT(L, W, next[l], next[left[d]])
10                            else  leftcount ← 0
11                         if right[d] ≠ NIL and r ≠ NIL and MATCH(right[d], r)
12                            then rightcount ← COUNT(W, R, next[right[d]], next[r])
13                            else  rightcount ← 0
14                         total ← total + leftcount ∗ rightcount
15                         if leftcount > 0
16                            then total ← total + leftcount ∗ COUNT(W, R, right[d], r)
17                         if rightcount > 0 and l = NIL
18                            then total ← total + rightcount ∗ COUNT(L, W, l, left[d])
19           return total
```

This is an O(N^3) algorithm.

Way to improve the speed:

I think pruning can improve the speed of searching process.

We can first search for the list and delete the connector that has no match in the sentence as many connectors are useless and never occurs in the sentence again.