Computational Linguistics (CS579)

Ch4. Automatic Syntactic Analysis

Prof. Feng Zhiwei

4.1 Recursive transition Network (RTN)

4.1.1 recursive property of natural language:

FSTN are not mathematically adequate for the description of certain kinds of embedded structures. We shall extend FSTN to RTN.

The number of sentences in the language is infinite. We can not enumerate all the sentences in a language,. However, we may extend the length of the sentence by certain means.

E.g.

1) The man chants.
2) The man who the woman sees chants.
3) The man who the women the girl sees sees chants.

The sentence 2) is the extension of sentence 1) adding a WHO-clause (who the woman sees). The sentence 3) is the extension of sentence 1) adding a WHO-clause (who the girl sees).

The result of psychology shows, that the limit of short memory volume of human is seven. But if we do not consider the psychological factors, the number of WHO-clauses that can be added to the sentence is infinite.

The object-clause also can be added after the predicate. E.g.

1) Mayumi says that Maria is a genius.
2) Mayuimi says that Jack says that Maris is a genius.
3) Mayumi says that Jack says that John says that Maria is a genius.

The examples demonstrate that English syntax is fundamentally recursive. It is easy to construct an English sentence that contains an English sentence, an English sentence contains an English sentence that contains an English sentence, .., and so on for as long as we like. But, of course, such sentences will become increasingly hard to understand as they get longer.

4.1.2 Improvement of FSTN to RTN

If we have FSTH = (Q, V, T),

where

Q = {1,2,3,4}

V:= {everyone, Verb, me, ANYWORD}

VERB = {hates, dismisses, likes}

ANYCLAUSE = {when I ask a question, If I hit the boy, …}

T:

T{anyone, 1} ={2}

T{VERB, 2} = {3}

T {me, 3} = {4}

T {ANYCLAUSE, 4} = {4}

Net: EVERYONE_ME

.

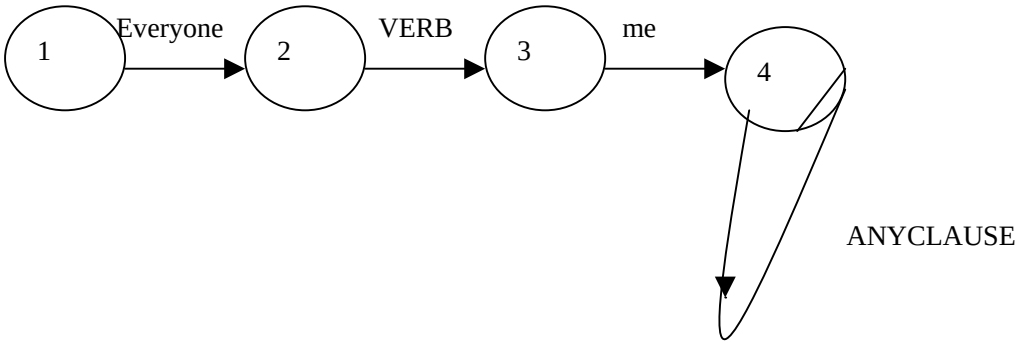


Fig. 1

It can accept (generate or recognize) following sentences:

E.g. 1) Everyone **hates** me when I hit the boy.

2) Everyone **dismisses** me when I ask a question.

Now let us try to generalize this FSTN. For, actually, in the pattern ”everyone … me…”,, we ought to allow for the first slot to be filled with more than one word, as the following sentences show.

3) Everyone **will punish** me when I hit the boy.

4) Everyone **has forgotten** me when I ask a question

5) .Everyone **will love** me when I ask a question.

The obvious solution is to include a whole bunch of connected arcs between 2 and 3, instead of the single verb arc.

In this case, we shall add some states (21, 22), and some function of state transition:

T{has, 2} = {21}

T {V_PERF, 21} = 3

T {will, 2} = {22}

T {V_BASE, 22} = {3}

T {V_PRES, 2} = {3}

Where:

V_PERF = {forgotten, punished, loved} :

V_BASE = {forget, punish, love}

V_PRES = {forgets, punishes, loves}

Our FSTN become as follows:

:

.

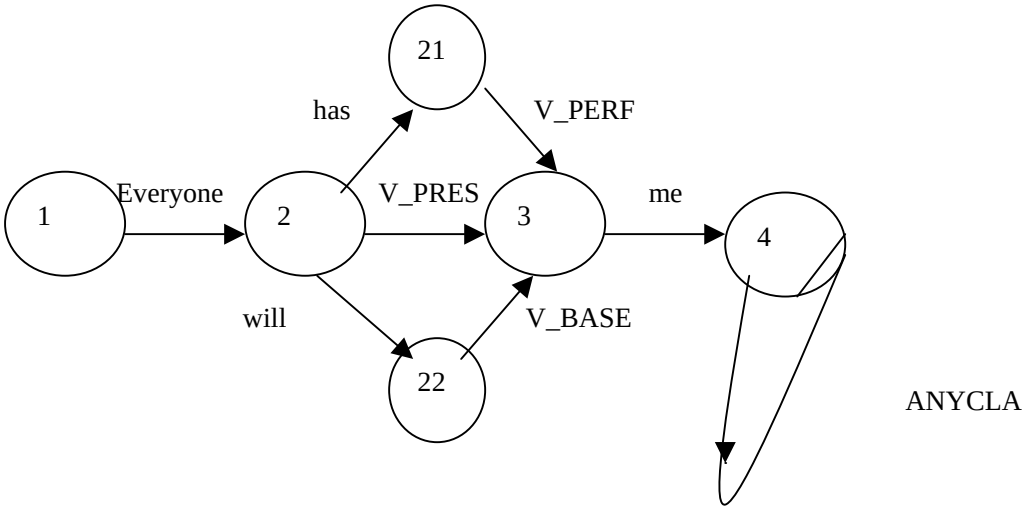
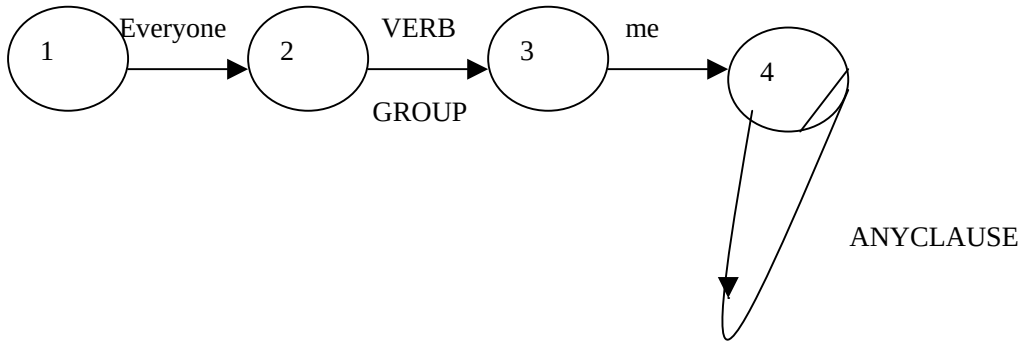


Fig. 2

The nodes 2, 21, 22, 3 form a bunch, it represents a verb group, and it can be separated from our FSTN and becomes a sub-network.

Net: EVERYONE_ME

.



Sub-network: VERB_GROUP

:

.

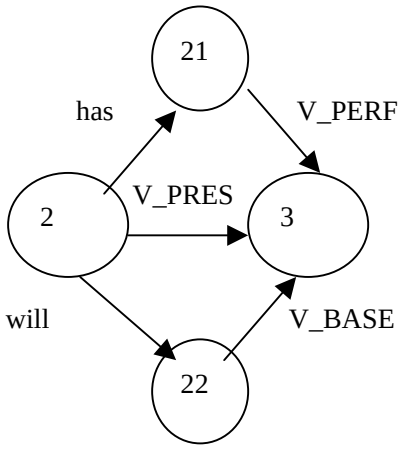


Fig. 3

The FSTN is improved as an RTN (Recursive Transition Network) consisting a sub-network.

.

A fundamental advance is made by using recursive transition network (RTN) instead of FSTN.. Basically, RTN is just like FSTN except that it introduces the extra concept of a named “sub-network”. That is, it is possible for an arc to name a sub-network to be traversed, instead of a specific word (or class of words) that is to appear. The idea is that if we have a commonly used bunch of arcs, we can express this abstraction by making it into a self-contained, namely network, this network can be referenced by its name in a network.

Just as an FSTN can be regarded as a specification of an FSA, sp an RTN can be regarded as a specification of a pushdown automaton (PA). Informally, in an RTN, to traverse an arc that is labeled with a sub-network, it is necessary to traverse the sub-network named, but remembering where to resume when that has been done.

A pushdown automaton is an FSA that is equipped with an extra memory, a stack, that can be used for this purpose.. Of course, the pushdown can be used recursively, this is the reason for the

‘recursive’ in recursive transition network..

4.1.3 Recursive Transition Network (RTN)

The RTN in above example is a very piecemeal and ad hoc RTN. Now we shall introduce a more general RTN for small fragment of English as follows:

1) John sees the house.
2) Maria sings.
3) The table hits Jack.
4) John sees that Maria sings.
5) The table that lacks a leg hits Jack.

We can design a FSTN to recognize them.

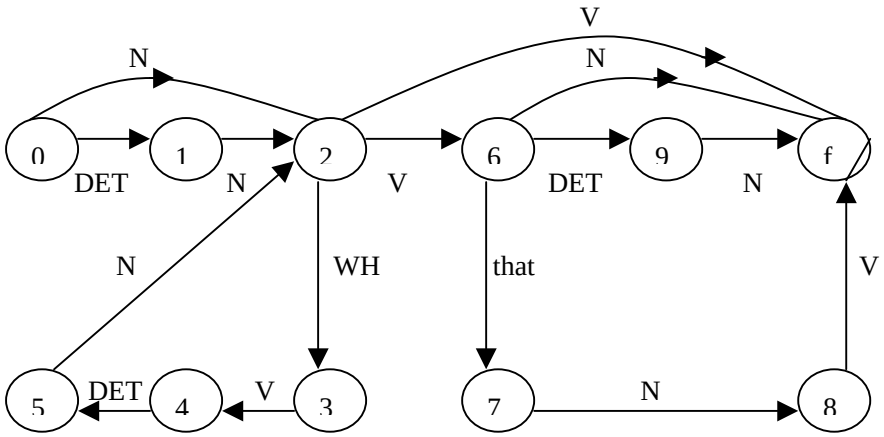


Fig. 4

The recognition process of 1) is: 0→ 2 → 6 → 9 → f.
The recognition process of 2) is: 0 → 2 → f.
The recognition process of 3) is: 0 → 1 →.2 → 6 → f.
The recognition of 4) is: 0 → 2 → 6 → 7 → 8 → f.
The recognition process of 5) is: 0 →. 1 → 2 → 3 → 4 → 5 → 2 → 6→ f.

This FSTN is very complex. If we would like to recognize more complex English sentences, the FSTN will be more complex. If we would like to recognize a book, the FSTN will be very complex.

However, the parts ‘0-1-2’ and ‘6-9-f’ are similar, the parts ‘2-6-9-f’ and ‘3-4-5-2’ are similar, the parts ‘7-8-f’ and ‘0-2-f’ are similar. It reflects the recursive property of natural language. We can use this recursive property to simplify our complex FSTN, and improve our FSTN to RTN.

The RTN includes three parts:

S-network:

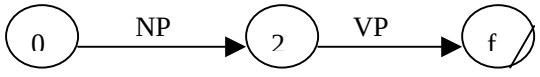


Fig. 5

NP-sub-network:

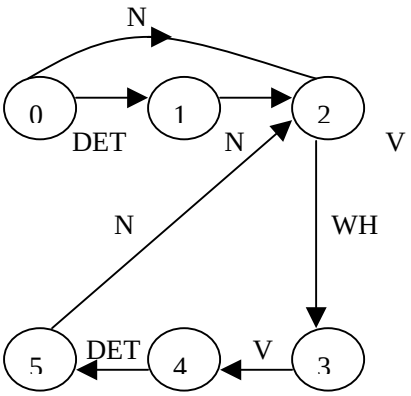


Fig. 6

VP-sub-network:

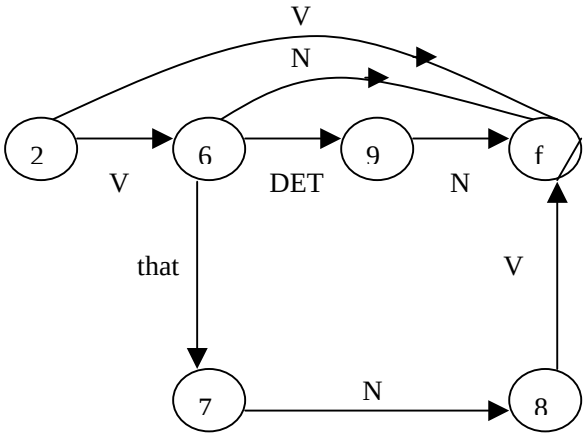


Fig. 7

We can further improve this RTN.

In NP sub-network, part ‘3-4-5-2” and part ‘2-6-9-f” is similar, the labels in arc is same: “V-DET-N’, it is a VP. We can simply NP sub-network as:

.

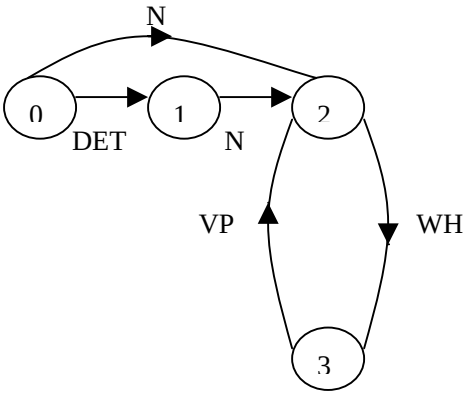


Fig. 8

In VP sub net-work, the part ‘6-9-f’ and the part ‘0-1-2’ of NP sub-network is similar, the label of

arc is ‘DET-N’ or ‘N’, it is a NP. The part ‘7-8-f’ and the part ‘0-2-f’ of S network is similar, it is a S. We can simplify VP sub-network as:

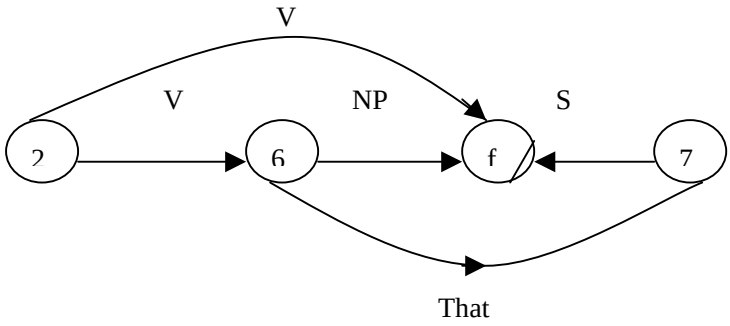


Fig. 9

This RTN is simplified on the base of recursive property of natural language. It includes three portions. The NP sub-network includes VP sub-network, the VP sub-network includes NP sub-network. It is really a recursive transition network..
Now we adjust the number in this RTN as follows:

S network:

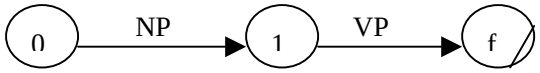


Fig. 10

This Network corresponds to following CFG (context-free grammar, or type 2 grammar) rule:

S → NP VP

NP sub-network:

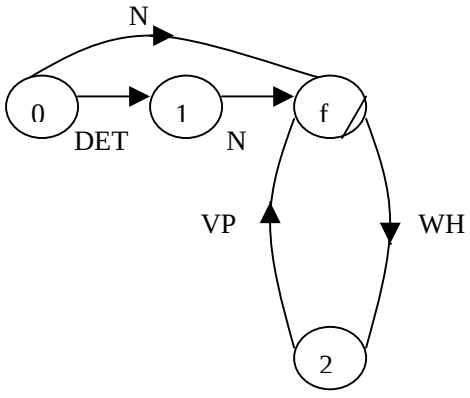


Fig. 12

This sub-network corresponds to following CFG rules:

NP → N
NP → DET N
NP → N WH VP
NP → DET N WH VP

VP sub-network:

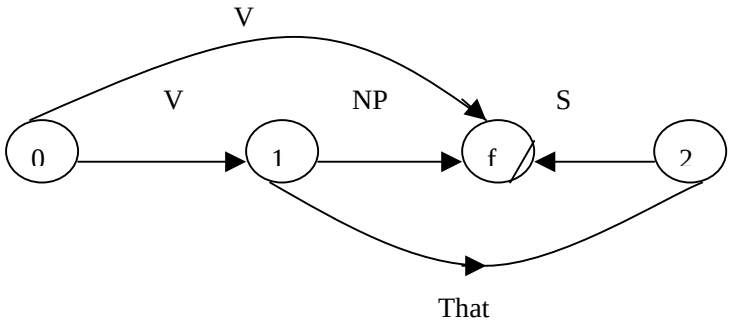


Fig. 13

This sub-network correspond to following CFG rules:

VP → V

VP → V NP

VP → V that S

If we have adequate CFG rules of English, it will be easy to design a good RNT to process English.

The principle of recognition in RTN:

- Start from S network,

--If research to NP, PUSH to NP-sub-network to recognize NP, then POP to S network when arrive to state f of NP sub-network.

--If search to VP, PUSH to VP sub-network to recognize VP, then POP to S network when arrive to state f of VP sub-network.

--If arrive to the final state f of S network, the recognition gets success.

- In NP sub-network, If search to VP, further PUSH to VP sub-network, then POP to NP sub-network when arrive to the state f of VP sun-network.
- In VP sub-network, if search to NP, further PUSH to NP sub-network, then POP to VP sub-network when arrive to the state f of NP sub-network.

The recognition of English sentences:

1) ‘John sees the house’.

--Start from S/0 (it denotes the state named 0 in the network S);

--PUSH to NP, at NP/0 to recognize ‘John’ (N), and enter NP/f;

--POP to S network, at the state S/1 to recognize VP;

--PUSH to VP sub-network, at VP/0 to recognize ‘sees’ (V). Then enter to state VP/1;

--PUSH to NP sub-network at NP/0 to recognize ‘the’ (DET) and enter to state NP/1, then recognize ‘house’ (N) then enter NP/f;

--POP to VP/f.

--POP to S/f.

2) ‘Maria sings’

--Start from S/0;
--PUSH to NP, at NP/0 to recognize ‘Maria’ (N), then enter NP/f;
--POP to S network, at the state S/1 to recognize VP;
--PUSH to VP sub-network, at VP/0 to recognize ‘sings’ (V). Then enter the VP/f;
--POP to S/f.

3) ‘The table hits Jack’

--Start from S/0;
--PUSH to NP/0 to recognize ‘the’ (DET), then enter NP/1 to recognize ‘table’ (N); and arrive to NP/f;
--POP to S/1 to recognize VP;
--PUSH to VP/0 to recognize ‘hits’ (V) then enter VP/1;
--PUSH to NP/0 to recognize ‘Jack’, then arrive to NP/f;
--POP to VP/f;
--POP to S/f..

4) ‘John sees that Maria sings’

--Start form S/0;
--PUSH to NP/0 to recognize ‘John’ (N); then enter to NP/f;
--POP to S/1 to recognize VP;
--PUSH to VP/0 to recognize ‘sees’ (V), then enter to VP/1; at VP/1 recognize ‘that’, then enter VP/2;
--PUSH to S/0 to recognize NP;
--PUSH to NP/0 to recognize ‘Maria’ (N), then enter to NP/f;
--POP to S/1 to recognize VP;
--PUSH to VP/0 to recognize ‘sings’ (V), then enter to VP/f;
--POP to S/f, it must be further POPped,
--POP to VP/f,
--POP to S/f.

5) ‘The table that lacks a leg hits Jack’

--Start from S/0;
--PUSH to NP/0 to recognize ‘the’ (DET), then enter to NP/1 to recognize ‘table’ (N), afterwards arrive NP/f;
--In state NP/f recognize ‘that’ (WH), then enter to state NP/2;
--PUSH to VP/0 to recognize ‘lacks’ (V), then enter to VP/1;
--PUSH to NP/0 to recognize ‘a leg’ (NP); at NP/0 to recognize ‘a’ (DET), at and NP/1 to recognize ‘leg’ (N), then enter to NP/f;
--POP to VP/f;
--POP to NP/f;

--POP to S/1;

--PUSH to VP/0 to recognize ‘hits’ (V), then enter the state VP/1;

--PUSH to NP/0 to recognize ‘Jack’ (N), then enter to NP/f;

--POP to VP/f;

--POP to S/f.

The recognition process of 50 can be illustrated as follows:

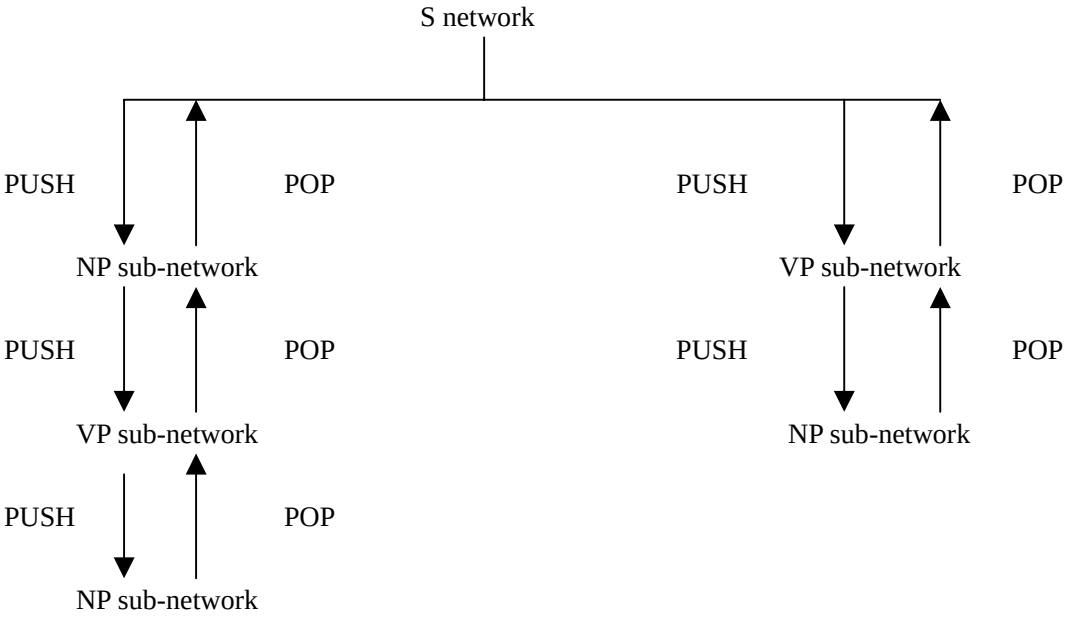


Fig. 14

This illustration represents the embedding structure in natural language. The original reason of embedding structure is the recursive property of natural language.

In RTN, the PUSH and POP operation is controlled by the ‘pushdown stack’.

A pushdown stack is a device for storing information that can manipulated by two basic operation:

- A piece of information can be PUSHed (added) on the stack;
- The most recently added piece of information can be POPped (removed) from the stack.

When we need to indicate the contents of a pushdown stack, we will simply separate the items with colons “:”, the first (most recently added) item being on the left. We will denote the empty stack by empty space. So here is how we would denote the stack that results from ‘pushing’ the symbols a, b and c on the empty stack in that order:

c: b: a:

The principle of pushdown is “**first in last out**”.

In the traversal of RTN, the configuration consists following three components:

- R1 (STATE): the name of current state;
- R2 (INPUT): remaining symbols that have not been recognized;
- R3 (STACK): a pushdown stack of position (state names).

The format of a configuration is:

<STATE, INPUT, STACK> .

For example, in the recognition process of the sentence “John sees the house”, in the moment when the ‘sees’ was recognized and the RTN returns to NP/0, the configuration can be expressed as:

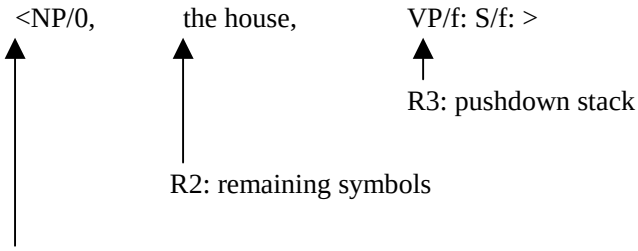


Fig. 16

The pushdown can be illustrated as follows:

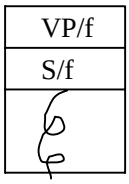


Fig. 17

It means that in the recognition of ‘the house’, firstly POP to the state VP/f, then POP to the state S/f according to principle “first in last out”.

In the beginning of the traversal, the configuration is:

<S/0, …, >

“…” is input symbols, the pushdown stack is empty..

If the traversal is successful, the final configuration is:

<S/f, , >

In the case, input symbols were recognized, the pushdown stack became empty.

Now let us to write the traversal process of above sentences.

1) ‘John sees the house’.

<S/0, John sees the house, >
<NP/0, John sees the house, S/1:>
<NP/f, sees the house, S/1:>
<S/1, sees the house, >
<VP/0, sees the house, S/f:>

<VP/f, the house, S/f: >
<S/f, the house, >
< , the house, >
FAIL

<VP/1, the house, S/f:>
<NP/0, the house, VP/f: S/f:>
<NP/1, house, VP/f: S/f:>
<NP/f, , VP/f: S/f:>
<VP/f, , S/f:>
<S/f, , >
< , , >
SUCCESS

2) ‘Maria sings’

<S/0, Maria sings, >
<NP/0, Maria, S/1:>
<NP/f, sings, S/1:>
<S/1, sings, >
<VP/0, sings, S/f:>
<VP/f, , S/f:>
<S/f, , >
< , , >
SUCCESS

3) ‘The table hits Jack’

<S/0, the table hits Jack, >
<NP/0, the table hits Jack, S/1:>
<NP/1, table hits Jack, S/1:>
<NP/f, hits jack, S/1:>
<S/1, hits jack, >
<VP/0, hits Jack, S/f:>

<VP/f, Jack, S/f:>
<S/f, Jack, >
FAIL

<VP/1, Jack, S/f:>
<NP/0, Jack, VP/f: S/f:>
<NP/f, , VP/f: S/f:>
<VP/f, , S/f:>
<S/f, , >
< , , >
SUCCESS

4) ‘John sees that Maria sings’

<S/0, John sees that Maria sings, >
<NP/0, John sees that Maria sings, S/1:>
<NP/f, sees that Maria sings, S/1:>
<S/1, sees that Maria sings, >
<VP/0, sees that Maria sings, S/f:>

<VP/f, that Maria sings, S/f:>
<S/f, that Maria sings, >
< , that Maria sings , >
FAIL

<VP/1, that Maria sings, S/f:>
<VP/2, Maria sings, S/f:>
<S/0, Maria sings, VP/f: S/f:>
<NP/0, Maria sings, S/1, VP/f: S/f:>
<NP/f, sings, S/1: VP/f: S/f:>
<S/1, sings, VP/f: S/f;>
<VP/0, sings, S/f: VP/f: S/f:>

,.

<VP/1, , S/f: VP/f; S/f:>
FAIL

<VP/f, , S/f: VP/f: S/f:>
<S/f, ,VP/f: S/f:>
<VP/f, , S/f:>
<S/f:, , >
< , , >
SUCCESS

5) ‘The table that lacks a leg hits Jack’

<S/0, the table that lacks a leg hits Jack, >
<NP/0, the table that lacks a leg hits Jack, S/1:>
<NP/1, table that lacks a leg hits Jack, S/1:>
<NP/f. that lacks a leg hits Jack, S/1:>
<NP/2, lacks a leg hit Jack, S/1:>
<VP/0, lacks a leg hits Jack, NP/f: S/1:>

<VP/f, a leg hits jack, NP/f: S/1:>
<NP/f, a leg hits Jack, S/1:>
<S/1, a leg hits jack, >
<VP/0, a leg hits jack, S/f:>
FAIL

<VP/1, a leg hits Jack, NP/f: S/1:>
<NP/0, a leg hits Jack, VP/f: NP/f: S/1:>
<NP/1, leg hits jack, VP/f: NP/f: S/f1>
<NP/f, hits Jack, VP/f: NP/f: S/1:>
<VP/f, hits Jack, NP/f: S/1:>
<NP/f, hits Jack, S/1:>
<S/1, hits Jack, >
<VP/0, hits Jack, S/f:>

<VP/f, Jack, S/f:>
<S/f, Jack, >
< , Jack, >
FAIL

<VP/1, Jack, S/f:>
<NP/0, Jack, VP/f: S/f:>
<NP/f, , VP/f: S/f:>
<VP/f, , S/f:>
<S/f, , >
< , , >
SUCCESS

4.1.4 Random generation in RTN

RTN can be used to generate. But the generate is random. Since the generation is random, we only need to consider one alternative at each point and can discard the others..

Following is the generation process of “Maria saw the dog”.

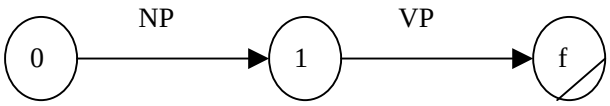
<S/o, , >
<NP/0, ,S/1:>
<NP/1, Maria, S/1:>
<S/1, Maria, >
<VP/0, Maria, S/f:>
<VP/1, Maria saw, S/f:>
<NP/0, Maria saw, VP/f: S/f:>
<NP/1, Maria saw the, VP/f: S/f:>
<NP/f, Maria saw the dog, VP/f: S/f:>
<VP/f, Maria saw the dog, S/f:>
<S/f, Maria saw the dog, >

In this case, the pushdown stack became empty, the RTN arrive the final state S/f, the sentence “Maria saw the dog” was generated.

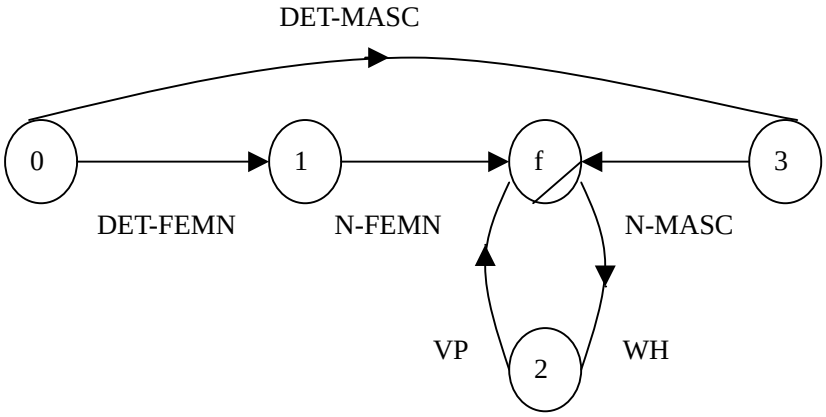
.

4.1.5 Pushdown Transducers (PT)

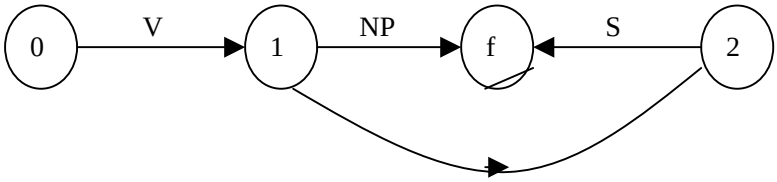
We saw that an FST is simple an FSA that deals with two tapes. Similarly, we can construct a pushdown transducer (PT) as a pushdown automaton that deals with two tapes.
It is similar easy to produce a simple PT for limited English-French translation.
S network:



NP sub-network:



VP sub-network:



that – que

Fig. 18

Here,

N-MASC: masculine noun, English-French pairs are ‘man - homme’, ‘horse – cheval’, leg-jambe..

N-FEMN: feminine noun, English-French pairs are: ‘house-maison’, ‘table-table’.

DET-MASC: masculine article, English-French pairs are: ‘a-un’ , ‘the-le’, ‘this-ce’.

DET-FEMN: feminine article, English-French pairs are: ‘a-une’, ‘the-la’, this-cette’.

NP: noun phrase (proper noun), English-French pairs are: ‘John-Jean’, ‘Maria-Marie’, ‘Jack-Jack’.

VP: verb, English-French pairs are: ‘see-voit’, ‘hits-frappe’ ,’sings-chante’, ‘lacks-manque’.

WH: WH word, English-French pairs are: ‘who-qui’, ‘which-qui’, ‘that-qui’.

Apart from the alternations for deal with gender in noun phrase, this PT is a direct translation of our previous RTN for English sentences.

E.g. John sees the house → Jean voit la maison

Maria sings → Marie chante

The table hits Jack → la table frappe Jack

John sees that Maria sings → Jean voit que Marie chante

The table that lacks a leg hits Jack → la table qui manque un jambe frappe Jack.

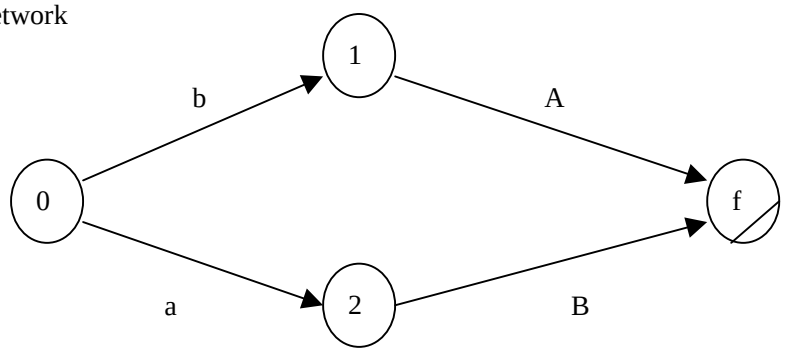
In our MT, the word order of both languages is always same.

4.2 Augmented Transition Network (ATN)

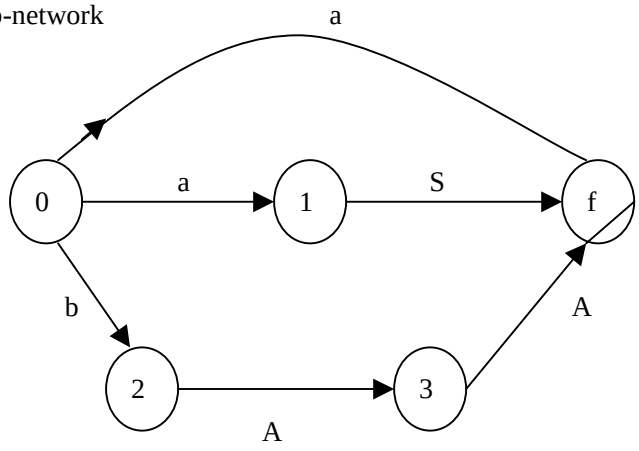
4.2.1 Limitation of RTN:

RTN has more powerful explanation force for the linguistic phenomena. The strings $\{a^n b^n\}$ that can't be generated (or recognized) by FSA now can be generated (or recognized) by following RTN.

S network



A sub-network



B sub-network

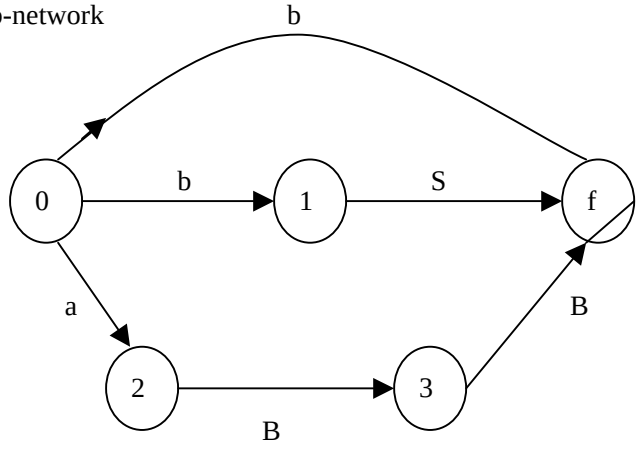
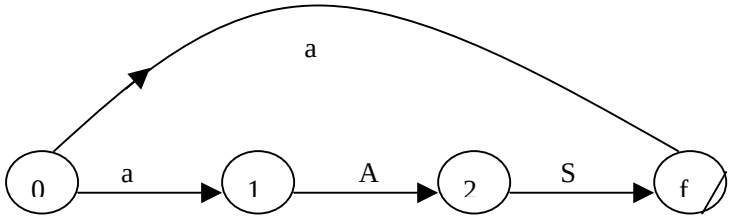


Fig. 1

The string $\{\alpha\ \alpha^*\}$ can be generated (or recognized) by following RTN:

S network:



A sub network:

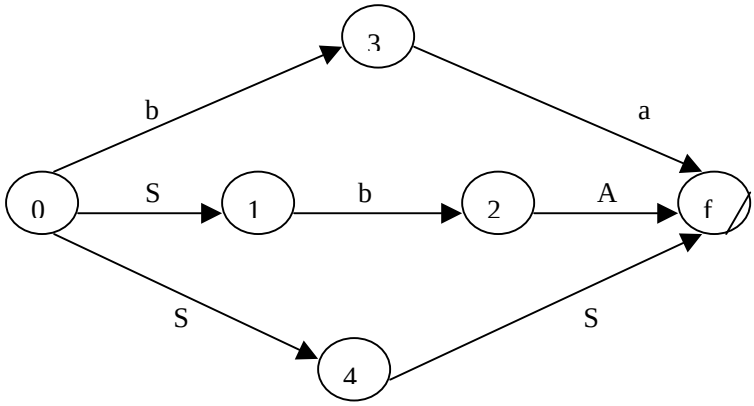


Fig. 2

These RTN are more complex. For more information, please read: John E. Hopcroft, Jeffrey D. Ullman, Introduction Automata theory, language, computation, Addison-Wesley Publishing Company, 1979, p81-84.

However, the RTN can‘t deal with the problem of word order in machine translation. In French, adjective standardly follows the noun they modify, whereas in English they precede it. E.g. we would want ‘a short name’ to be translated to ‘un nom court’. If we try to extend the PT to include adjective in the English side, we have to write network like the following:

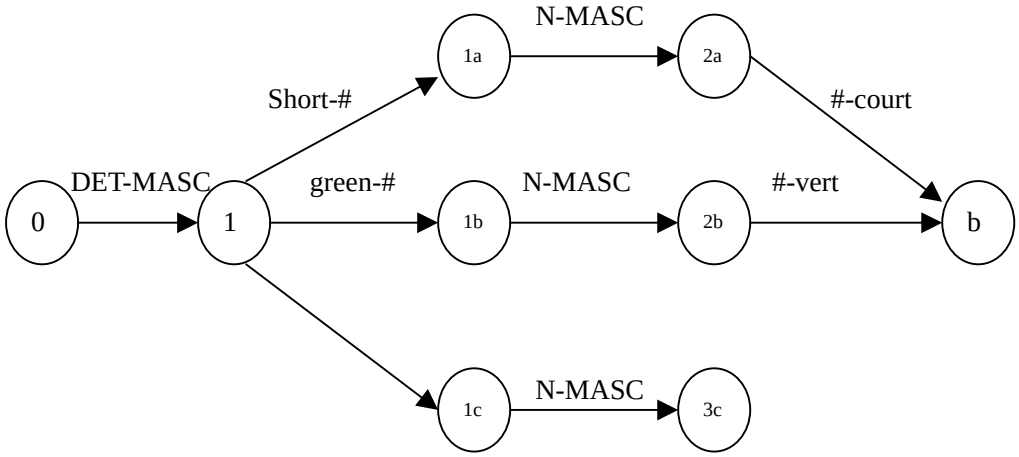


Fig. 3

Where there are three arcs for each possible English adjective (and yet this network only deal with single adjective !). The problem is that the French adjectives, which hare known before the English noun is encountered, have somehow to be remembered and then produced on the second

tape after the noun is translated.

4.2.2 ATN

ATN allow value to be remembered in this way during a network traversal by providing **register** (variables) for storing information.

Registers are rather like (local) variables in a procedural programming language. Thus, in our notation for ATNs, we will include declaring the registers used at the beginning of each network. Each arc of the network may then be annotated with instructions for how to shuffle information between these registers when it is traversed.

4.2.2.1 Adjustment of word order in translation

In the translation of noun phrase, we will use the register FNP to keep track of the value (French phrase) to be returned by the NP network.

- in the French translation of a proper noun, this is simple whatever single French name corresponds to that English name, if there is one.
- In the French translation of a simple noun phrase, we need to have a French determiner, a collection of French adjective and a French noun. We can introduce registers called FDET, FADJS and FNOUN to keep track of this information used in the translation of a noun phrase.

We can annotate the ATN network (for NOUN translation) informally as follows:

NP network:

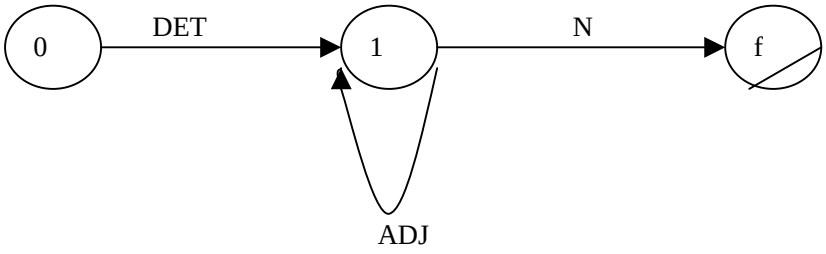


Fig. 4

NP network:

Registers	FADJS, FNOUN, FDET, FNP
Initial 0	set FADJS to the empty string
Final f	return FNP
From 0 to 1 by DET	set FDET to French(*)
From 1 to 1 by ADJ	set FADJS to FADJS + French(*)
From 1 to f by N	set FNP to FDET + FNOUN + FADJS

Where '*' is the current word, '+' is a form of string concatenation that insert spaces between the words, 'French' is a function that English words to their French translations (which we naively assume to be unique).

The register FADJS is used here to hold a string that will translate a whole sequence of adjectives. As more English adjectives are discovered, their French translations are added to the end of the current value of this register.

4.2.2.2 Gender agreement in translation

The precise form of a determiner or an adjective in French depends on the gender of the noun it qualified, this being either masculine or feminine. E.g.

English	French	
a green tree →	an arbre vert	(masculine)

a green table → une table verte (feminine)

In ATN, we can use the register FGENDER to record the gender that we have chosen, then translate the determiner and adjective on the basis of whatever values FGENDER has. To force the analysis to succeed only when the choice of FGENDER is the same as the gender of the French noun

Network NP:

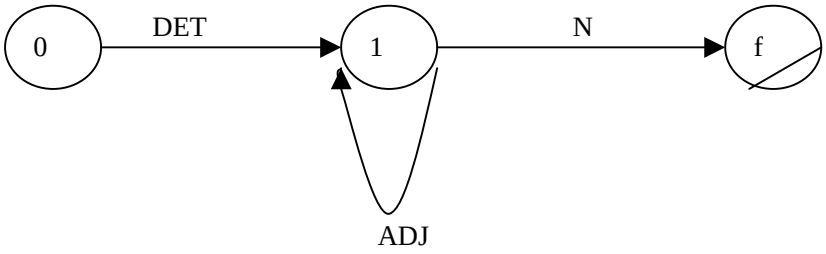


Fig. 5

Network NP

Registers FADJS, FDET, FNP, FNOUN, FGENDER	
Initial 0	set FADJS to empty string
Final f	return FNP
From 0 to 1 by DET	set FGENDER to 'masculine'
	set FDET to French (*, 'masculine')
From 0 to 1 by DET	set FGENDER to 'feminine'
	set FDET to French (*, 'feminine')
From 1 to 1 by ADJ	set FADJS to FADJS + French (*, FGENDER)
From 1 to f by N	set FNOUN to French (*)
	the gender of FNOUN must be the same as FGENDER
	set FNP to FDET + FNOUN + FADJS

where we now assume that the function 'French' is provided with a gender as well as an English word, where this is required.

The use of the register FGENDER enabled us to use each arc for both of the two alternatives (masculine and feminine).

In our simple English-French MT example, we have seen how ATN register can be used locally to reorder material to the output.

ATN us an extension of RTN because of the use of registers. A state of an ATN recognition can be represented by the following:

- R1, the current state (and network name);
- R2, the remaining symbols in the input string;
- R3, the pushdown stack of return points;
- R4, the values of registers.

The first three are more or less same as for the RTN case.

4.3 Bottom-up paring and top-down parsing

4.3.1 context-free grammar (CFG)

A grammar G can be defined as follows:

G = (Vn, Vt, S, P)

Vn is the set of non-terminal symbols, e.g. NP, VP, N, S;.., etc

Vt is the set of terminal symbols, e.g. John, run, the, …, etc;

S is starting symbol, and S ∈ Vn;

P is rewriting rule. The format of rule is as

$$\phi \rightarrow \psi$$

where ϕ is called Left Hand Side (LHS), and ψ is called Right Hand Side (RHS). '$\phi \rightarrow \psi$' means that we may replace ϕ (LHS) by ψ(RHS)..

If we have a grammar G, then we can use it to derive the language L(G). concretely speaking, we can have $S \rightarrow \phi 1 \rightarrow \phi 2 \rightarrow \phi 3 \cdots \rightarrow \phi n$. The terminal string ϕn is called the well-formed sentence of L(G).

In the rewriting rule $\phi \rightarrow \psi$, if ϕ is a single non-terminal symbol, and ψ is a string (consisting non-terminal symbols and terminal symbols), then the grammar g is called con-text-free grammar (CFG) or phrase structure grammar (PSG).

The Bacus-Naur Normal Form in programming language is just a kind of rewriting rule of CFG grammar.

We have following PSG grammar:

G = (Vn, Vt, S, P)

Vn = {S, NP, VP, S}

Vt = {John, workers, employed, died}

S = {S}

P:

S → NP VP 1

VP → V NP 2

VP → V 3

NP → {John, workers} 4

V → {employed, coughed} 5

The derivation history (derivation process) of sentence "John employed workers" is as follows:

Derivation process	**used rule**
S	start
NP VP	1
NP V NP	2
John	4
John employed NP	5
John employed workers	4

Above derivation process is also the generation process of the sentence

The sentence structure can be represented by the tree graph:

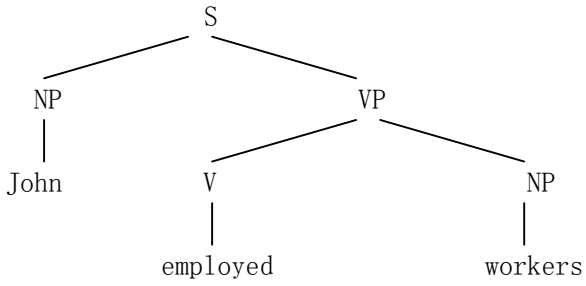


Fig. 6

This tree is called as ‘Phrase Structure Tree’.

We may also use the list to represent the phrase structure tree. The first element of list is the label in the root of tree, following elements are the labels of corresponding descendants of tree. In LISP, above phrase structure tree can be expressed as

```
(S(NP John)(VP(V employed)(NP workers)))
Or
(S
    (NP John)
    (VP
        (V employed)
        (NP workers)))
```

This phrase structure grammar may be used to generate sentence ‘John coughed’, the derivation history is:

Derivation process	**used rule**
S	start
NP VP	1
NP V	3
John V	4
John coughed	5

Its phrase structure is:

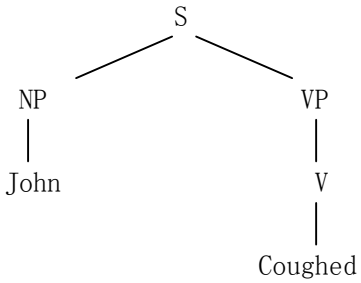


Fig. 7

The list representation is:

```
(S
    (NP  John)
    (VP
        (V  coughed)))
```

The bottom-up parsing and top-down parsing are based on PSG (or CFG).

4.3.2 bottom-up parsing

4.3.2.1 The process of bottom-up parsing

Giving our three-word sentence

John employed workers

We find the first place (‘John’) in the string that matches the RHS of the rule or lexical entry. We can then label the sequence of words or categories that matched the RHS of the rule with the LHS of the rule:

NP_

John employed workers

Now we continue, trying to further rewrite the string “NP employed workers’.

Given only that, we can ask if there is any rule in the grammar whose RHS is simply NP. For example, K → NP. If there is, then we could explore the possibility of allowing this NP to be dominated by K in the structure we are trying to build up. But there is no such rule, so we are forced to consider the next word in the string, which we find to be category V:

.NP_ V_____

John employed workers

We ask:

If any rule simply has NP and nothing on the RHS? -- No;

There is any RHS in the grammar rule that allows us to group the sequence NP V together? -- NO.

There is any RHS in the grammar rule is identical to V? -- Yes, there is a rule ‘VP → V’.

So we can add this information to the parsing tree that we are trying to build up:

VP_____

NP_ V_____

John employed workers

We ask:

There is any rule simply has NP and nothing on the RHS? – No;

There is any RHS in the grammar rule that is identical to NP VP? – Yes. It is rule ‘S → NP VP’.

Then we have a parse tree with an S spanning the initial NP and the immediately following VP that we found.. Our parse tree became:

S _____

VP_____

NP_ V _____

John employed workers

We ask:

There is any RHS in the grammar rule is identical to ‘S’? – No.

There is any RHS in the grammar rule is identical to ‘S workers’? –No

There is any RHS in grammar rule is identical ‘workers’? –Yes. It is rule ‘NP → {John, workers}.

Then our parse tree became:

S _____

VP_____

NP_ V _____ NP ____

John employed workers

At this point, however, we cannot proceed. S does not occur as an RHS, nor does S NP, and nor does NP. So, we have nothing we can do with the sequence.

Our goal is to find an S spanning the entire string, but this route has led an S spanning the first two words in the string and a dangling unattached NP at the end. Clearly, we have gone wrong somewhere.

In this case, we will backtrack to the last point at which we were faced with a choice and explore

one or more other possibilities. Let we go back to the following situation:

VP_____

NP_ V _____

John employed workers

We have just seen that putting the NP and VP together here as an S led nowhere.

We ask:

There is any RHS in the grammar rule is identical to NP? – No.

There is any RHS in the grammar rule is identical to VP? – No.

There is any RHS in the grammar rule is identical to 'workers'? – Yes. It is rule 'NP → John, workers'. Then our parse tree became:

VP_____

NP_ V _____ NP ____

John employed workers

We shall check exhaustively all the possibilities. We ask:

: There is any RHS in the grammar rule is identical to NP? – NO.

There is any RHS in the grammar rule is identical to VP? – NO.

There is any RHS in the grammar rule is identical to NP VP? – Yes. However, it shall lead nowhere.

There is any RHS in the grammar rule is identical to NP VP NP? – No.

There is any RHS in the grammar rule is identical to VP NP? – No.

Now we again come to nowhere. We shall backtrack again. We arrive at the following configuration:

NP_ V _____ NP ____

John employed workers

We ask,

There is any RHS in the grammar rule is identical to NP? – No.

There is any RHS in the grammar rule is identical to V? – Yes. However, It will lead nowhere. .

There is any RHS in the grammar rule is identical to NP V? – No.

There is any RHS in the grammar rule is identical to V NP? – Yes. It is the rule 'VP → V NP'.

Now our parse tree became

VP_____

NP_ V _____ NP ____

John employed workers

Returning to the start of the sentence, We check whether NP can exhaust an RHS (for the nth time), and then check if NP VP can. The answer is 'Yes, so we can proceed to add S to our parse tree:

S _____

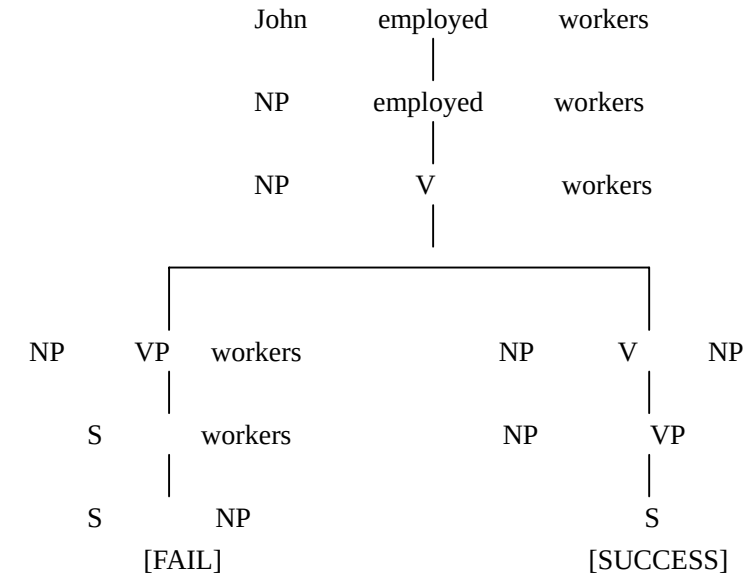
VP _____

NP_ V _____ NP ____

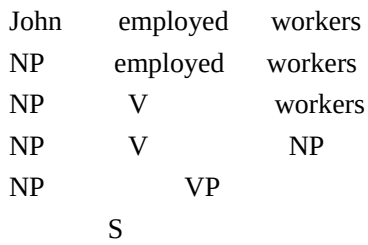
John employed workers

This S spans the entire string, and so we have a success.

The process of bottom-up parsing can be illustrated by following search tree:



The success process is as follows:



Its list representation is:

(John employed workers)
((NP John) employed workers)
((NP John) (V employed) workers)
((NP John) (V employed) (NP workers))
((NP John) (VP (V employed) (NP workers)))
(S ((N John) (VP (V employed) (NP workers))))

In the bottom-up parsing, there is redundancy in different parts of the search space (see Fig. 8):

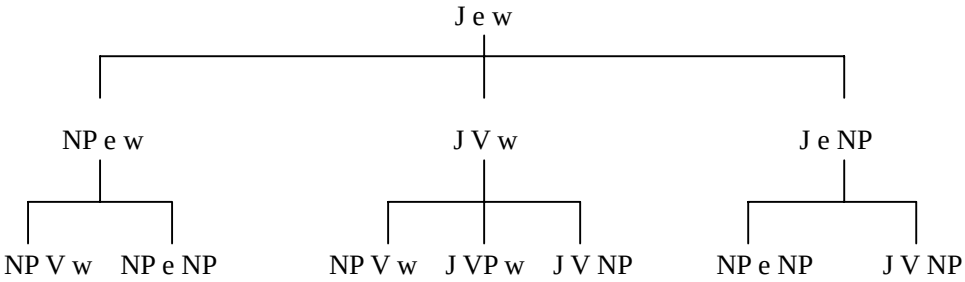


Fig. 8

The redundancy is caused through rewriting of lexical items. By this reason, the direction of our bottom-up parsing must be from left to right and start from left. It is a kind of LR-algorithm.

4.3.2.2 Shift-Reduce algorithm:

In the shift-reduce algorithm, the stack is used for information access. The operation methods are shift, reduce, refuse and accept. In the shift, the symbol waiting to process is move to the top of stack. In the reduce, the symbol on stack top is replaced by RHS of grammar rule, if the RHS of the rule is matched with the symbol on stack top. If the input string is processed, and the symbol

on stack top becomes S (initial symbol in the string), then the input string is accepted. Otherwise, it is refused.

Following is the shift-reduce process of sentence “John employed workers”.

	Stack	**operation**	**the rest part of input string**
			John employed workers
	John	shift	employed workers
	NP	reduce with rule 4	employed workers
	NP employed	shift	workers
+++	NP V	reduce with rule 5	workers
	NP VP	reduce with rule 3	workers
	S	reduce with rule 1	workers
	NP V	backtracking to +++	workers
	NP V workers	shift	
	NP V NP	reduce with rule 4	
	NP VP	reduce with rule 2	
	S	reduce with rule 1	

[SUCCESS]

More complex example:

Sentence :“The boy hits the dog with a rod”.

Our PS grammar is as follows:

G = {Vn, Vt, S, P}

Vn = {S, NP, VP, Det, N. V. Prep}

Vt = {the, boy, rod, dog, hits, with, a}

S = S

P:

S → NP VP	1
NP → Det N	2
VP → V NP	3
PP → Prep NP	4
Det→ {the}	5
Det→ {a}	6
N → {boy}	7
N → {dog}	8
N → {rod}	9
V → {hits}	10
Prep → {with}	11
VP → VP PP	12

Stack	**operation**	**the rest part of input string**
		the boy hits the dog with a rod
the	shift	boy hits the dog with a rod
Det	reduce by rule 5	boy hits the dog with a rod
Det boy	shift	hits the dog with a rod
Det N	reduce with rule 7	hits the dog with a rod
NP	reduce with rule 2	hits the dog with a rod

NP hits	shift	the dog with a rod
NP V	reduce with rule 10	the dog with a rod
NP V the	shift	dog with a rod
NP V Det	reduce with rule 5	dog with a rod
NP V Det dog	shift	with a rod
NP V Det N	reduce with rule 8	with a rod
NP V NP	reduce with rule 2	with a rod
+++ NP VP	reduce with rule 3	with a rod
S	reduce with rule 1	with a rod
S with	shift	a rod
S Prep	reduce with rule 11	a rod
S Prep a	shift	rod
S Prep Det	reduce with rule 6	rod
S Prep Det rod	shift	
S Prep Det N	reduce with rule 9	
S Prep NP	reduce with rule 2	
S PP	reduce with rule 4	

In this moment, there is no suitable rule can be used in the grammar, the parsing can not go on. We have to backtracking to '+++'. In stead of reducing with rule 1, we shift next word 'with', then reduce with rule 11.

NP VP	backtracking	with a rod
NP VP with	shift	a rod
NP VP Prep	reduce with rule 11	a rod
NP VP Prep a	shift	rod
NP VP Prep Det	reduce with rule 6	rod
NP VP Prep Det rod	shift	
NP VP Prep Det N	reduce with rule 9	
NP VP Prep NP	reduce with rule 2	
NP VP PP	reduce with rule 4	
NP VP	reduce with rule 12	
S	reduce with rule 1	

[SUCCESS]

The tree of this sentence is:

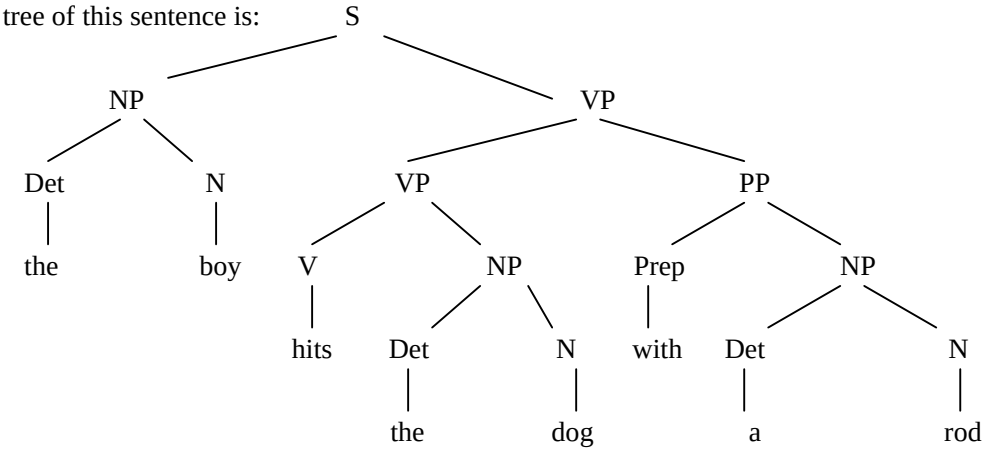


Fig 9

In English, the PP may also be used as modifier of NP. If we add a new rule

NP → NP PP

In our grammar, then we may get another result, its tree is as follows:

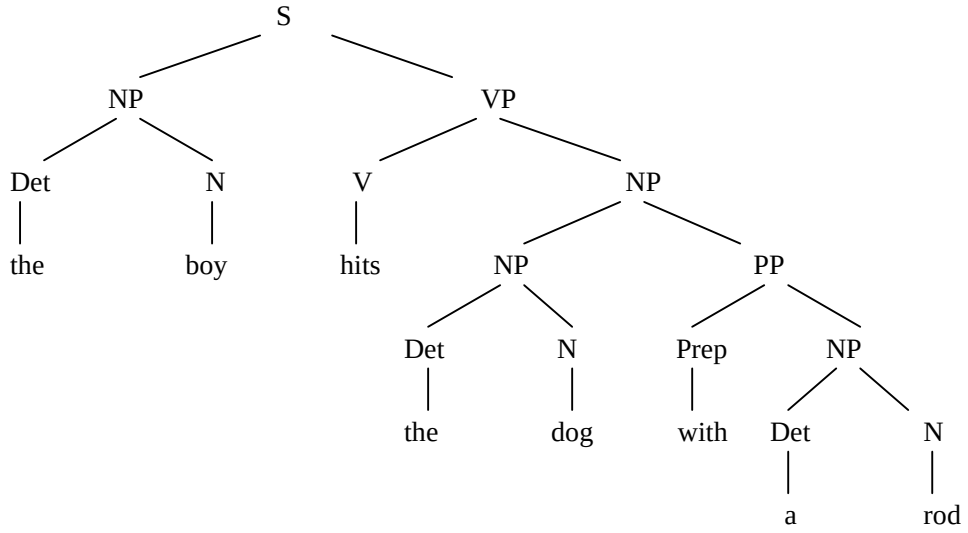


Fig. 10

In this case, the parsing result is ambiguous.

The structure of sentence ‘the man saw a boy with a telescope’ is same as our example.

PP-attachment is a difficult ambiguity in English parsing. We shall discuss the ambiguity in next chapter.

4.3.3 Top-down parsing

A pure (left-to-right) top-down (depth-first) parser would have proceeded roughly as follows:

I am looking for an S.

What can an S consist of?

An S can consist of an NP followed by a VP.

So I need to look for an NP first.

What can an NP consist of?

There are no grammar rules expanding NP.

There is a lexical entry listing ‘workers’ as a member of the category NP.

Is the first word in the string ‘workers’?

No.

There is a lexical entry listing ‘John’ as a member of the category NP.

Is the first word in the string ‘John’?

Yes.

I have found an NP consisting of the word ‘John’.

I now need to look for a VP.

+++ What can a VP consist of?

A VP can consist of a V.

What can a V consist of?

There are no grammar rules expanding V.

There is a lexical entry listing ‘coughed’ as a member of the category V.

Is the next word in the string ‘coughed’?
No.
There is a lexical entry listing ‘employed’ as a member of the category V.
Is the next word in the string ‘employed’?
Yes.
I have found a V consisting of the word ‘employed’.
I have found a VP consisting of a V consisting of the word ‘employed’.
I have found an S consisting of an NP consisting of a word ‘John’ and a VP consisting of a V consisting of a word ‘employed’.
Have I reached the end of the string?
No.
Oh dear, I must have done something wrong.
Try going back to +++ and doing something different.
I still need to look for a VP.
A VP can also consist of a V following by an NP.
I now need to look for a V.
What can a V consist of?
There a no grammar rules expanding V.
There is a lexical entry listing ‘coughed’ is a member of the category V.
Is the next word in the string ‘coughed’?
No.
There is a lexical entry listing ‘employed’ as a member of the category V.
Is the next word in the string ‘employed’?
Yes.
I have found V consisting of the word ‘employed’.
Now I need to look for an NP.
Where can an NP consist of?
There are no grammar rules expanding NP.
There is a lexical entry listing ‘workers’ as a member of the category NP.
Is the next word in the string ‘workers’?
Yes.
I have found an NP consisting of the word ‘workers’.
I have found a VP consisting of a V consisting of the word ‘employed’ followed by an NP consisting of the word ‘workers’.
I have found an S consisting of an NP consisting of the word ‘John’ and a VP consisting of a V consisting of the word ‘employed’ followed by an NP consisting of the word ‘workers’.
Have I reached the end of the string?
Yes.
I have succeeded.

Fig. 11 shows the search tree for top-down parsing of the example sentence ‘John employed workers’

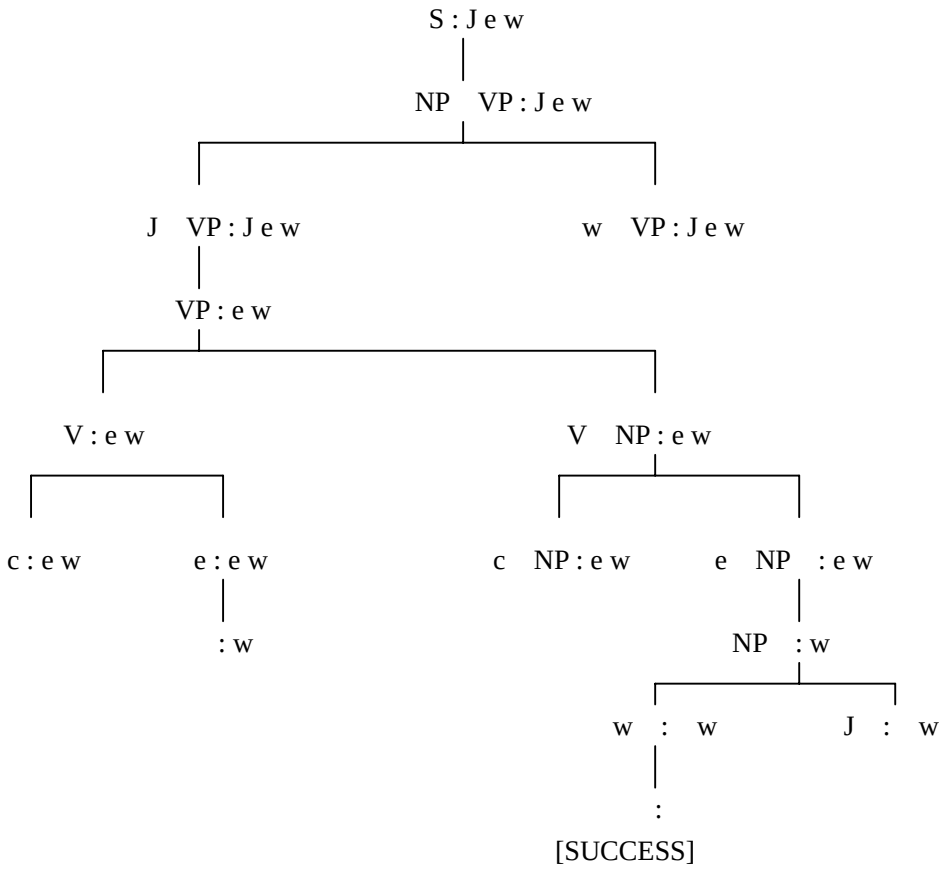


Fig. 11

In this search tree, if we follow the searching order as follows, we shall get success.

Searching goal		**The rest part of input string**
S	:	John employed workers
NP VP	:	John employed workers
John VP	:	John employed workers
VP	:	employed workers
V NP	:	employed workers
employed NP	:	employed workers
NP	:	workers
Workers	:	workers:

[success]

The list representation is:

Goal sequence	**word sequence**
((S))	(John employed workers)
((NP)(VP))	(John employed workers)
(John (VP))	(John employed workers)
((VP))	(employed workers)
((V)(NP))	(employed workers)
(employed (NP))	(employed workers)
((NP))	(workers)
(workers)	(workers)
( )	( )

The search process of the sentence “the boy hits the dog with a rod”:

	Searching goal	**The rest part of input string**
	S	the boy hits the dog with a rod
	NP VP	the boy hits the dog with a rod
	Det N VP	the boy hits the dog with a rod
	N VP	boy hits the dog with a rod
+++	VP	hits the dog with a rod
	V NP	hits the dog with a rod
	NP	the dog with a rod
	Det N	the dog with a rod
	N	dog with a rod
	VP (backtracking to +++)	hits the dog with a rod
	VP PP	hits he dog with a rode
	V NP PP	hits the dog with a rod
	NP PP	the dog with a rod
	Det N PP	the dog with a rod
	N PP	dog with a rod
	PP	with a rod
	Prep NP	with a rod
	NP	a rod
	Det N	a rod
	N	rod
	[SUCCESS]	

4.4 Left-corner parsing

4.4.1 definition of ‘left-corner’: The left-lower corner of every sub-tree in the PSG tree is left corner.

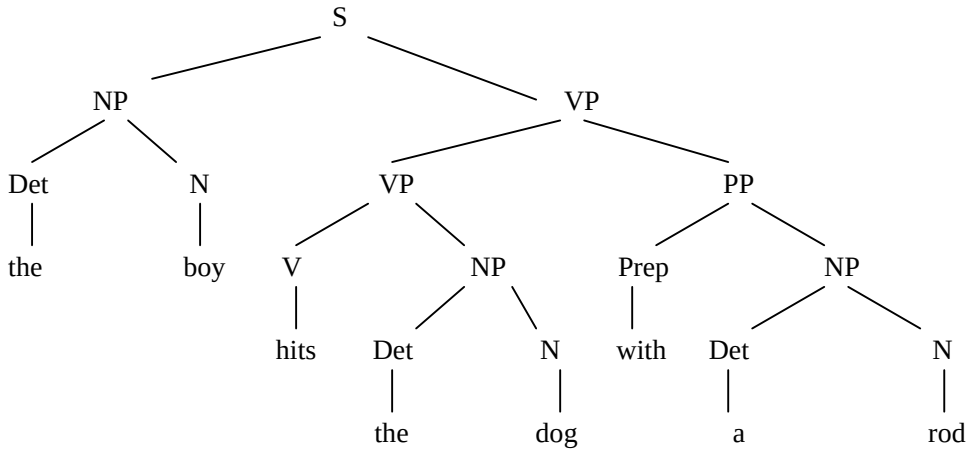


Fig. 12

In Fig. 12, ‘the’ is the left corner of Det. Det is the left corner of NP. NP is the left corner of S. ‘hits’ is the left corner of V. V is the left corner of VP. ‘with’ is the left corner of Prep. Prep is the left corner of PP. In the rewriting rule, the left corner is the first symbol of RHS. E,g, in A → BC, B is the left corner.

4.4.2 process of left-corner parsing

The tree graph representation of rule A → B C is:

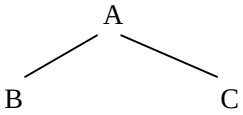


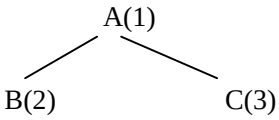
Fig. 13

If we use top-down approach, the parsing process is A → B → C, first parsing the top part, then parsing the bottom part.

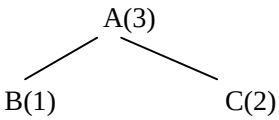
If we use bottom-up approach, the parsing process is B → C → A, first parsing the bottom part, then parsing the top part.

If we use left-corner approach, the parsing process is B → A → C, the direction of parsing is: bottom – top – bottom.

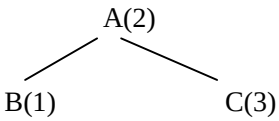
If we use number to express the parsing order, then the parsing process of three approaches is as follows:



Top-down approach



Bottom-up approach



Left-corner approach

Fig. 14

The process order of left-corner parsing starts from left corner B, then go to A (bottom-up), then go to C (top-down).

Obviously, the left-corner paring combines the bottom-up parsing and the top-down parsing together.

4.4.3 example of left-corner parsing

Our PSG grammar is as follows:

G = {Vn, Vt, S, P}

Vn = {S, NP, VP, Det, N. V. Prep}

Vt = {the, boy, rod, dog, hits, with, a}

S = S

P:

S → NP VP 1

NP → Det N 2

VP → V NP 3

PP → Prep NP 4

Det→ {the} 5
Det→ {a} 6
N → {boy} 7
N → {dog} 8
N → {rod} 9
V → {hits} 10
Prep → {with} 11
VP → VP PP 12

Now we parsing the sentence “the boy hits the dog with a rod” using left-corner approach.

Step 1: Start from ‘the’ in the head of the sentence, using rule 5, from the left-corner ‘the’ of the rule 5, we can get Det.

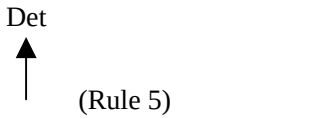


Fig. 15

Step 2: The left-corner of rule 2 is ‘Det’, starting from ‘Det”, using rule 2 to predict N after Det.

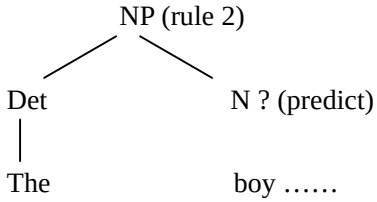


Fig. 16

Step 3: Using rule 7, from boy get N.

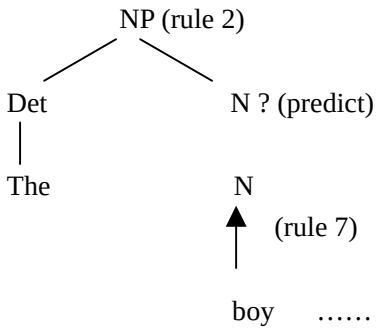


Fig. 17

Step 4: Due to the father node of node ‘boy’ is N, it can be matched with ‘N ? (predict)’, then we get sub-tree NP:

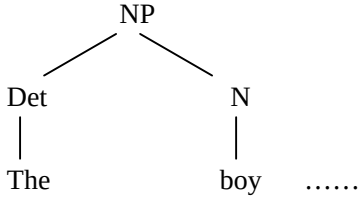


Fig. 18

Step 5: NP is the left-corner of rule 1, using rule 1 and predict VP.

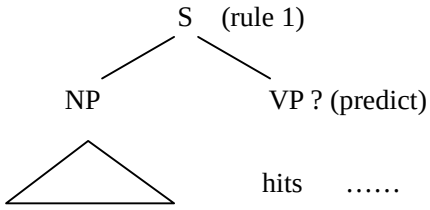


Fig. 19

Step 6: using rule 10, from ‘hits’ get V.

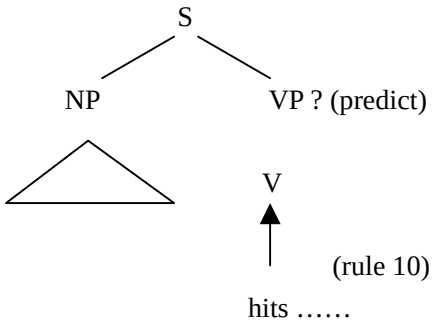


Fig. 20

Step 7; V is the left-corner of rule 3, using rule 3 and predict NP.

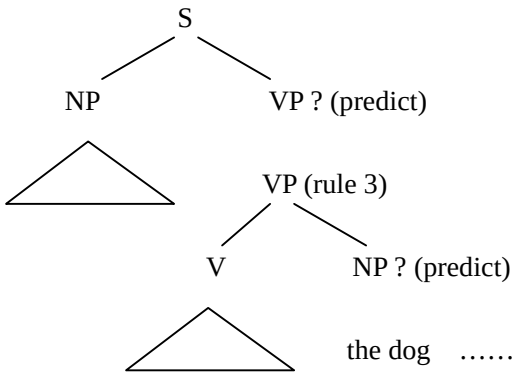


Fig. 20

Step 8: from ‘the dog’ get NP, then NP is confirmed, so we need now further to predict VP.

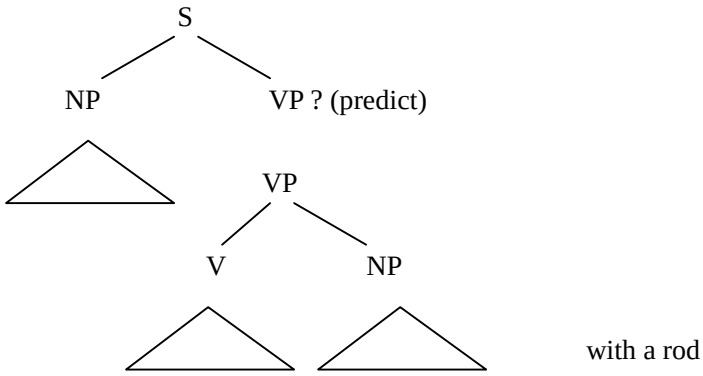
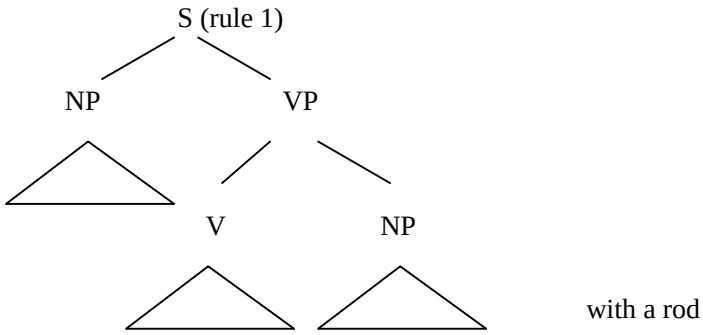


Fig. 21

Step 9: Reduce NP VP to S using rule 1. However, there is the sub-string ‘with a rod’ in the input, so the parsing fails. We have to backtrack to VP again



[Backtracking !]

Fig. 22

Step 10: VP is the left-corner of rule 12, using rule 12 and predict PP.

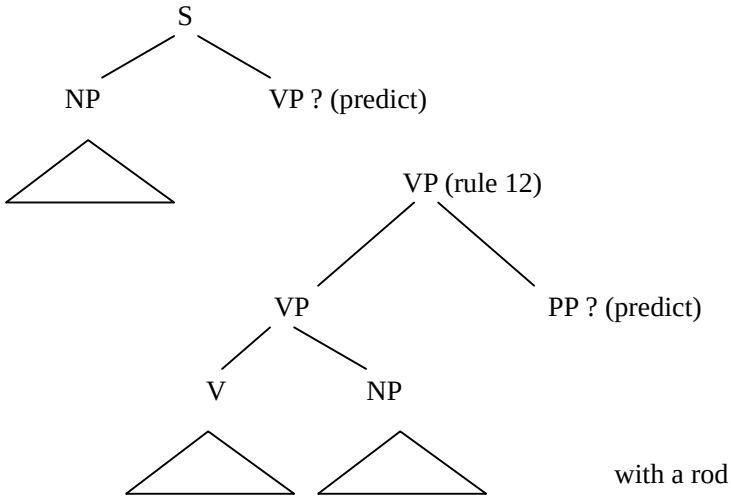
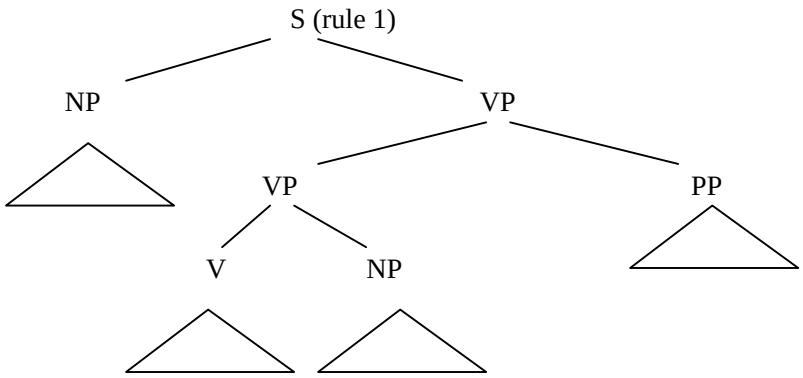


Fig. 23

Step 11: The prediction for VP is confirmed, now we arrive to the end of sentence, the parsing is completed successfully.



[Success]

Fig. 25

4.5 CYK approach

CYK approach is abbreviation of Cocke-Younger-Kasami approach. It is a parallel parsing algorithm.

4.5.1 Chomsky Normal Form:

CYK algorithm based on the Chomsky normal form.
Chomsky Normal form is the rewriting rule as follows:

A → BC

Here, LHS is non-terminal symbol (A), and RHS always includes two elements (B and C). It means that the RHS of Chomsky Normal Form is always binary.

Chomsky Normal Form is equivalent with all rule form of CFG. (Context Free Grammar). So CYK algorithm can be suitable for all form of CFG.

4.5.2 Table and box in CYK approach

Following table can expresses the result of CYK parsing for the sentence “the boy hits a dog”:

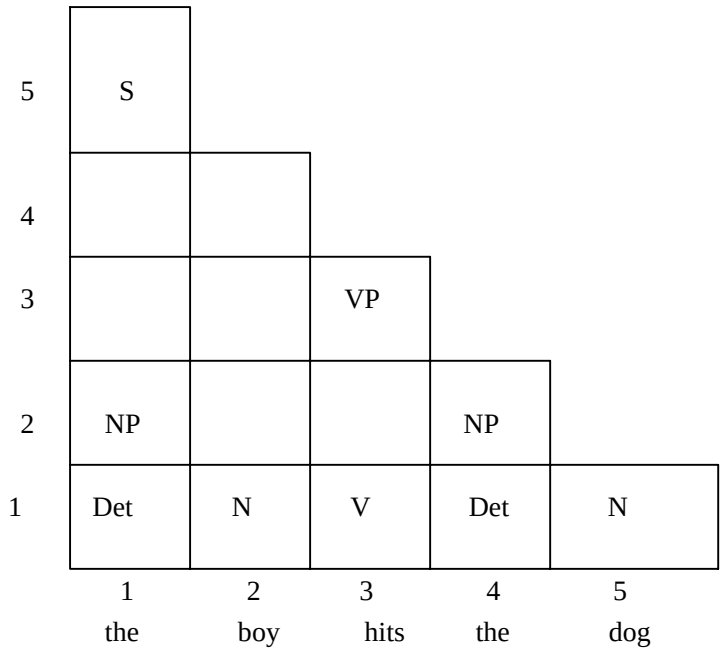


Fig. 26

In this table, the row number expresses the location of word in the sentence, the line number expresses the word number included in the grammatical category (e.g. N, V, NP, VP, S, etc). All the category is located in the box of the table. b_{ij} expresses the box that located in the row i and line j. Every grammatical category in the table can be expressed by b_{ij}.

‘Det belongs to b_{11}’ means : Det is located in row 1 and line 1.

‘N belongs to b_{21}’ means : N is located in row 2 and line 1.

‘V belongs to b_{31}’ means: V is located in row 3 and line 1.

‘Det belongs to b_{41}’ means; Det is located in row 4 and line 1.

‘N belongs to b_{51}’ means; N is located in row 5 and line 1.

By this reason,

The location of NP (the boy) is b_{12} (including 2 words),

The location of NP (a dog) is b_{42} (including 2 words),

The location of VP (hits a dog) is b_{33} (including 3 words),

The location of S (the boy hits a dog) is b_{15} (including 5 words).

Obviously, the table and the b_{ij} in the table can describe the structure of the sentence. For every category b_{ij}, i describes its location in the sentence structure, j describes the word number included in this category. If we may create the table and the b_{ij} in the table, the parsing is completed.

4.5.3 CYK Description of Chomsky Normal Form

In the Chomsky normal form A → BC,

if B belongs to b_{ik}, C belongs to $b_{i+k\ j-k}$,

Then A must belong to b_{ij}.

If we start from i-th word of the sentence create a sub-tree B including k words,

and then from i+k-th word of the sentence create a sub-tree C including j-k words, then the tree graph A can be expressed as follows:

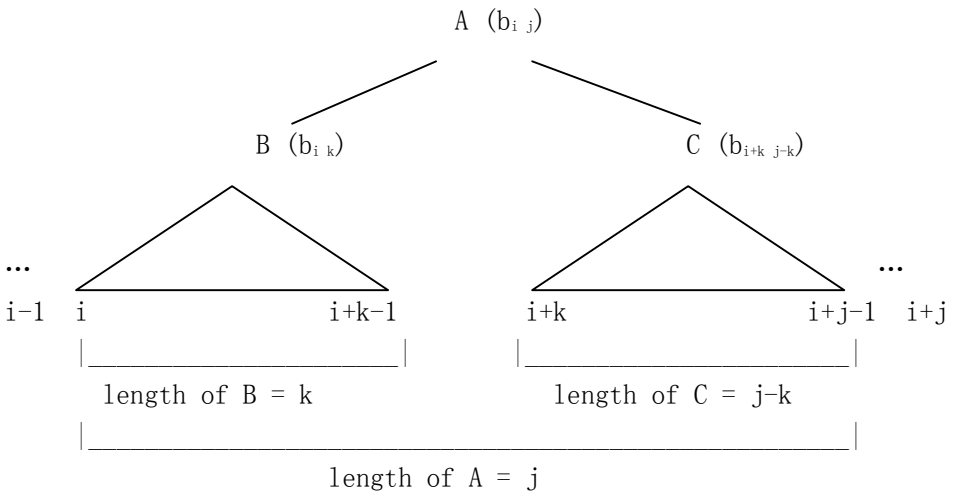


Fig. 27

For example, in Fig. 26, NP belongs to $b_{1\,2}$, Det belongs to $b_{1\,1}$, N belongs to $b_{2\,1}$, is represents the Chomscky normal form NP → Det N. In this case, i=1, k=1, j=2. Therefore, if we know the starting number i of B, the length k of B, the length j of A, then we can calculate the location of A, B and C in the CYK table: A belongs to $b_{i\,j}$, B belongs to $b_{i\,k}$, C belongs to $b_{i+k\,j-k}$.

In CYK approach, the important problem is how to calculate the location of A. The row number of A is always same as that of B, so if row number of B is i, then the row number of A must be i. The line number of A (=j) equals to the addition of the line number of B (=k) and the line number of C (=j-k): j = k + j - k.
Therefore, If we know the location of B and the location of C, it is easy to calculate the location of A.

If the length of input sentence is n, the CYK algorithm can be divided to two steps:

First step: start from i = 1, for every words Wi in input sentence (with length n), we have rewriting rule A → Wi, so we write the non-terminal symbol A of every Word Wi in the box of our table, and give the location number of box with bij. E.g, for sentence “The boy hits a dog”, we give the location number respectively for every words of sentence is as follows: b_{11} (for Det [non-terminal symbol of ‘the’]), b_{21} (for N [non-terminal symbol of ‘boy’]), b_{31} (for V [non-terminal symbol of ‘hits’]), b_{41} (for second Det [non-terminal symbol of ‘a’]), b_{51} (for second N [non-terminal symbol of ‘dog’]).

Second step; For 1≤ h ≤j and all i, create $b_{i\,h}$. non-terminal set including b_{ij} can be defined as follows:

$b_{i\,j}$ = {A |for 1≤ k ≤j, B is included in $b_{i\,k}$, C is included in $b_{i+k\,j-k}$, and exists

grammar rule A → BC that A is included in $b_{i\,j}$}.

If box $b_{1\,n}$ includes initial symbol S, then input sentence will be accepted. The analysis gets success.

E.g, for rule 'NP → Det N' and Det belongs to $b_{1\,1}$, N belongs to $b_{2\,1}$, we can confirm that NP belongs to $b_{1\,2}$;
for rule 'NP → Det N' and 'Det" belongs to $b_{4\,1}$, N belongs to $b_{5\,1}$, we can confirm that NP belongs to $b_{4\,2}$;
for rule VP → V NP and V belongs to $b_{3\,1}$, NP belongs to $b_{4\,2}$, we can confirm that VP belongs to $b_{3\,3}$;
for rule S → NP VP and NP belongs to $b_{1\,2}$, VP belongs to $B_{3\,3}$, we can confirm that S belongs to $b_{1\,5}$. In our input sentence, n=5, so our sentence is accepted.

4.5.4 A complex example for CYK algorithm

If the PSG grammar is as follows:
S → NP VP
NP → N
NP → DET N
NP → N WH VP
NP → DET N WH VP
VP → V
VP → V NP
VP → V that S

Use CYK approach to analyze sentence 'the table lacks a leg hits Jack".

■ Transformation of rewriting rules to Chomsky normal form:
S → NP VP
NP → N
NP → DET N
NP → N WH VP It must be transformed to:
  NP → N CL
  CL → WH VP
NP → DET N WH VP It must be transformed to:
  NP → NP CL
  NP → DET N
  CL → WH VP
Here CL is WH clause, it = (that + VP)
VP → V
VP → V NP
VP → V that S It must be transformed to:
  VP → V TH
  TH → WH S

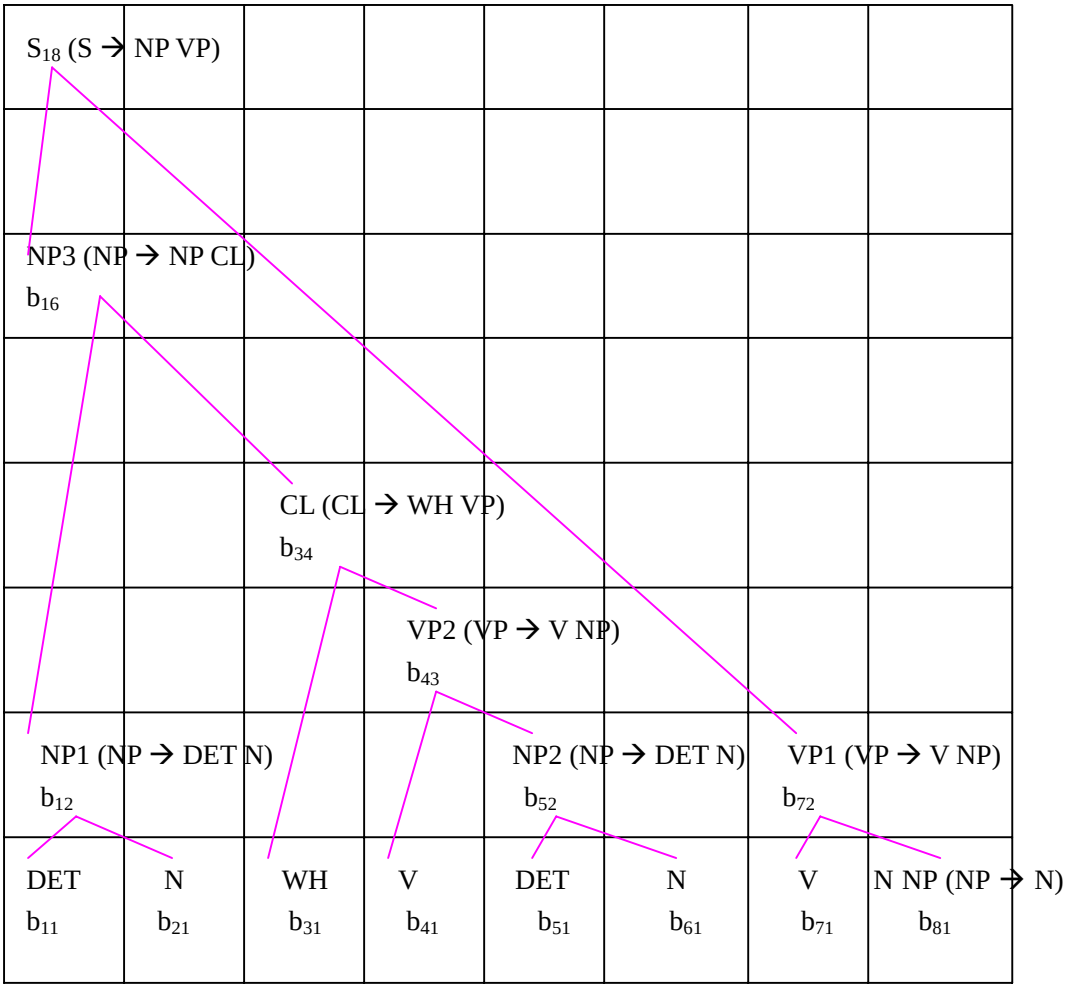
Here TH is that-clause, it = (that + S).

- Calculation of the bij of non-terminal symbols:

--To arrange POS non-terminal symbols and calculate their bij

"The	table	that	lacks	a	leg	hits	Jack"
DET	N	WH	V	DET	N	V	N
b_{11}	b_{12}	b_{13}	b_{14}	b_{15}	b_{16}	b_{17}	b_{18}

--To calculate the bij of phrase non-terminal symbols



The table that lacks a leg hits jack

Fig. 28

b_{ij} (NP1): i=1. j=1+1=2

b_{ij} (NP2): i=5, j=1+1=2

b_{ij} (VP1): i=7, j=1+1=2

b_{ij} (VP2): i=4, j=1+2=3

b_{ij} (CL): i=1, j=1+3=4

b_{ij} (NP3): i=1, j=2+4=6

b_{ij} (S): i=1, j=2+6=8

The length of this sentence is 8, and we get box line number of S is also 8, so the sentence was recognized.

By the CYK approach, we can create the pyramid in Fig. 28. This pyramid is also a tree graph.

The analysis result is the same as the result of RTN.