Lecture 2

Pushdown Automata and Stack-Based Parsing

Last modified 2016/04/27

In an interview with Naoki Fukui and Mihoko Zushi (Chomsky 2004:174f), Chomsky claims that the fact that there is an algorithm for converting an arbitrary context-free grammar into an equivalent nondeterministic pushdown automaton is the only result in formal language theory that has any linguistic significance:[1]

> ... in all of this work of the late 50s and early 60s there was only one result that I know of that had any linguistic significance. ... That's the fact that there's a constructive procedure to map context-free grammars into a strongly equivalent — crucially — non-deterministic push-down storage automaton. ... It's not interesting mathematics. It's just a constructive procedure. In fact, it's what underlies every parser. That's why every parser is a non-deterministic push-down storage automaton. It doesn't contribute much, but it explains why that's what every parser is. In so far as language is more or less context-free, you can parse it that way. But apart from that, I don't know of any results that are interesting. I mean, some results are amusing, but not linguistically significant.

Setting aside the merit of this claim, the existence of such an algorithm is a very important fact of great significance to linguistic and psycholinguistic research. In fact, there are many ways of converting context-free grammars to pushdown automata, and different methods lead to different "stack-based" methods of recognizing/parsing context-free languages.[2]

[1] The fact was proved in Chomsky 1962. Chomsky (2004) speaks of "strong equivalence", but the term, along with "strong generative capacity", has no precise meaning and will be avoided in these lectures.

[2] A *recognizer* for a grammar is an algorithm which takes a string as input and returns either "accept" or "reject", depending on whether the string belongs to the language of the grammar. A

Top-down and bottom-up recognition

We describe *pushdown automata* (PDA) informally. A *configuration* of a PDA is described by a pair consisting of the stack contents and the remaining portion of the input, both of which are formally strings. The top of the stack is either on the left or on the right, depending on the type of the recognizer/parser. S is the start symbol of a given CFG. Possible moves of a PDA are written in the form $C_1 \vdash C_2$, where C_1, C_2 are configurations. In the following descriptions, x is going to range over terminal strings, $X, Y_1, \ldots$ over terminal or nonterminal symbols, and α over strings of terminal or nonterminal symbols.

Top-down recognizer The top of the stack is on the left.

- Initial configuration:
$$(S, x)$$
where x is the input.
- Possible moves:
  - *Predict:* For each production $X \to Y_1 \ldots Y_n$ of the grammar,
$$(X\alpha, x) \vdash (Y_1 \ldots Y_n\alpha, x)$$
  - *Match:*
$$(a\alpha, ax) \vdash (\alpha, x)$$
where a is a terminal.
- Accepting configuration:
$$(\varepsilon, \varepsilon).$$

Bottom-up recognizer The top of the stack is on the right.

- Initial configuration:
$$(\varepsilon, x)$$
where x is the input.
- Possible moves:

parser does the same, but in addition also returns a representation of the parse tree(s) for the input string when it belongs to the language. Every well-known recognizer is implicitly a parser or can easily be turned into one, and for context-free grammars, it is known that every recognizer can be turned into a parser (which returns a single parse) with only logarithmic overhead (Ruzzo 1979), so we often do not make a clear distinction between the two types of algorithms.

– *Shift:*

$$(\alpha, ax) \vdash (\alpha a, x)$$

where a is a terminal.

– *Reduce:* For each production $X \to Y_1 \ldots Y_n$ of the grammar,

$$(\alpha Y_1 \ldots Y_n, x) \vdash (\alpha X, x)$$

• Accepting configuration:

$$(S, \varepsilon)$$

Let us see how the two recognition methods work for the following simple grammar:

1. S → NP VP
2. NP → Det N
3. VP → V NP
4. VP → V
5. Det → the
6. N → dog
7. N → cat
8. V → chased

The top-down recognizer accepts the input the dog chased the cat as follows:

(S, the dog chased the cat)	⊢ (NP VP, the dog chased the cat)	predict, 1
	⊢ (Det N VP, the dog chased the cat)	predict, 2
	⊢ (the N VP, the dog chased the cat)	predict, 5
	⊢ (N VP, dog chased the cat)	match
	⊢ (dog VP, dog chased the cat)	predict, 6
	⊢ (VP, chased the cat)	match
	⊢ (V NP, chased the cat)	predict, 3
	⊢ (chased NP, chased the cat)	predict, 8
	⊢ (NP, the cat)	match
	⊢ (Det N, the cat)	predict, 2
	⊢ (the N, the cat)	predict, 5
	⊢ (N, cat)	match
	⊢ (cat, cat)	predict, 7
	⊢ $(\varepsilon, \varepsilon)$	match

Here is how the bottom-up recognizer accepts the same input:

(ε, the dog chased the cat)	$\vdash$ (the, dog chased the cat)	shift
	$\vdash$ (Det, dog chased the cat)	reduce, 5
	$\vdash$ (Det dog, chased the cat)	shift
	$\vdash$ (Det N, chased the cat)	reduce, 6
	$\vdash$ (NP, chased the cat)	reduce, 2
	$\vdash$ (NP chased, the cat)	shift
	$\vdash$ (NP V, the cat)	reduce, 8
	$\vdash$ (NP V the, cat)	shift
	$\vdash$ (NP V Det, cat)	reduce
	$\vdash$ (NP V Det cat, ε)	shift
	$\vdash$ (NP V Det N, ε)	reduce, 7
	$\vdash$ (NP V NP, ε)	reduce, 2
	$\vdash$ (NP VP, ε)	reduce, 3
	$\vdash$ (S, ε)	reduce, 1

In an accepting computation of the top-down recognizer, the concatenation of the stack contents with the already matched portion of the input results in a *leftmost* derivation of the input (i.e., a derivation in which every step rewrites the leftmost nonterminal). On the other hand, the sequence of configurations of an accepting computation of the bottom-up recognizer corresponds to the reverse of a *rightmost* derivation of the input (i.e., a derivation in which every step rewrites the rightmost nonterminal).

Both recognizers are *nondeterministic*. The possible configurations of a nondeterministic PDA on a given input can be arranged in the form of a *computation tree* (Figures 2.1 and 2.2). An accepting computation is represented by a branch of the computation tree leading from the root to a leaf labeled by an accepting configuration.

There are two common ways to simulate these nondeterministic algorithms deterministically: *depth-first* and *breadth-first* methods. We only describe the former. The depth-first method performs the *preorder* traversal of the computation tree,[3] using backtracking. When the recognizer cannot make any further move from the current configuration, the simulator backtracks to the most recent configuration from which the nondeterministic recognizer can make an alternative move that has

[3] A *traversal* of a tree is a way of visiting each node of the tree exactly once. In preorder traversal, the root node is visited first, after which the immediate subtrees of the tree are traversed *in preorder* in turn, proceeding from left to right.

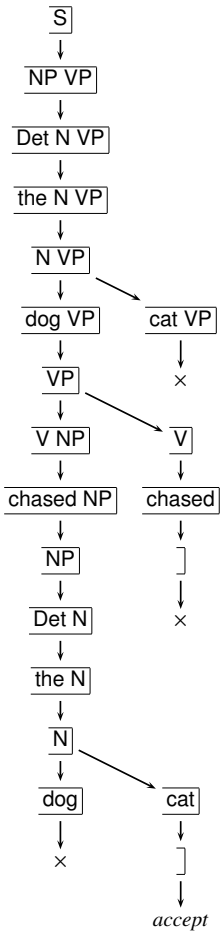


Figure 2.1: The computation tree of the top-down recognizer for the input the dog chased the cat. Only the stack contents are shown.

not yet been explored. Consider the following grammar:[4]

[4]This grammar is adapted from an example in Jurafsky and Martin 2009. As was already mentioned in footnote 5 of Lecture 1, the vertical bar notation is used to bundle together some productions with the same left-hand side. Thus, the notation Det → that | this | a represents three rules, Det → that, Det → this, and Det → a.

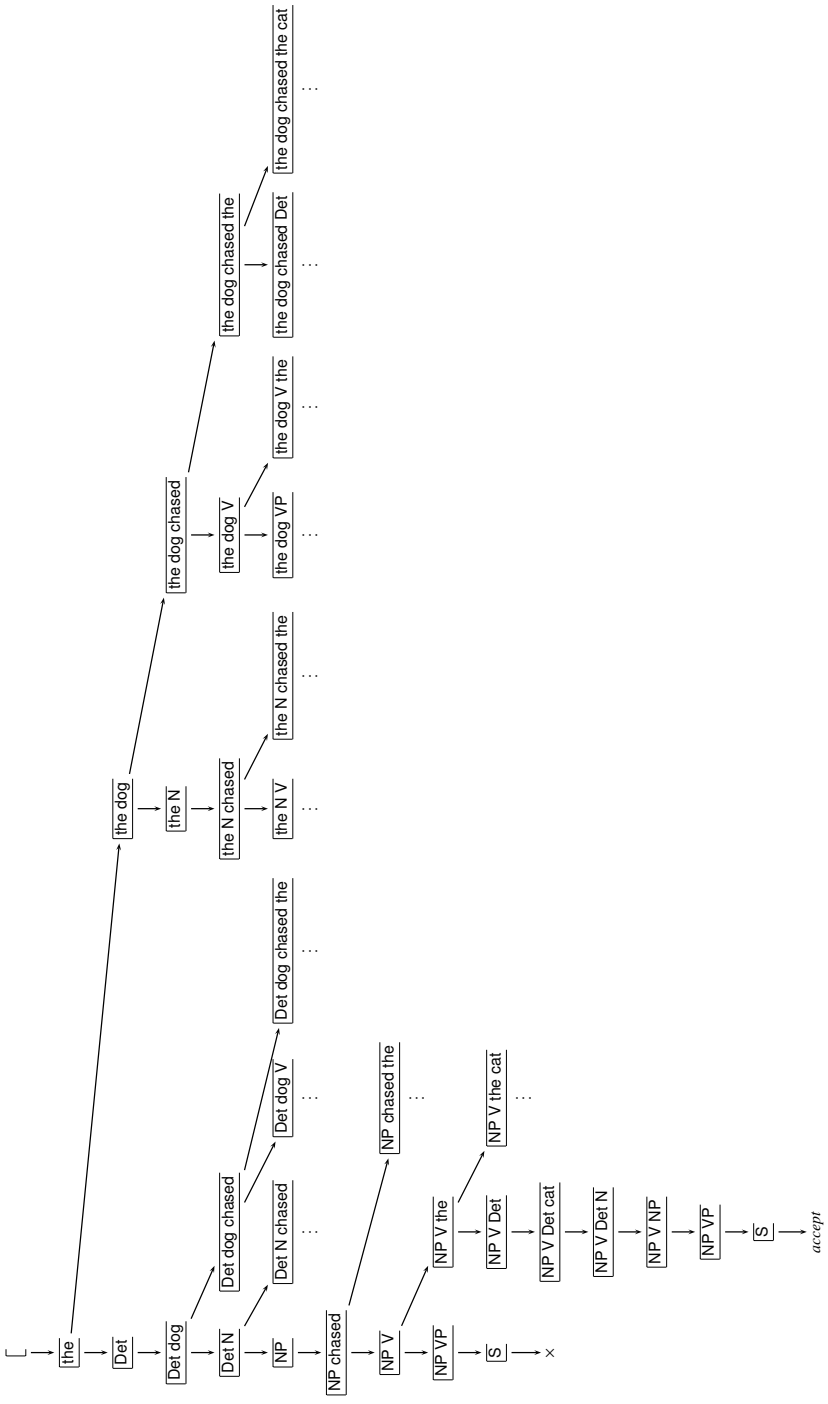
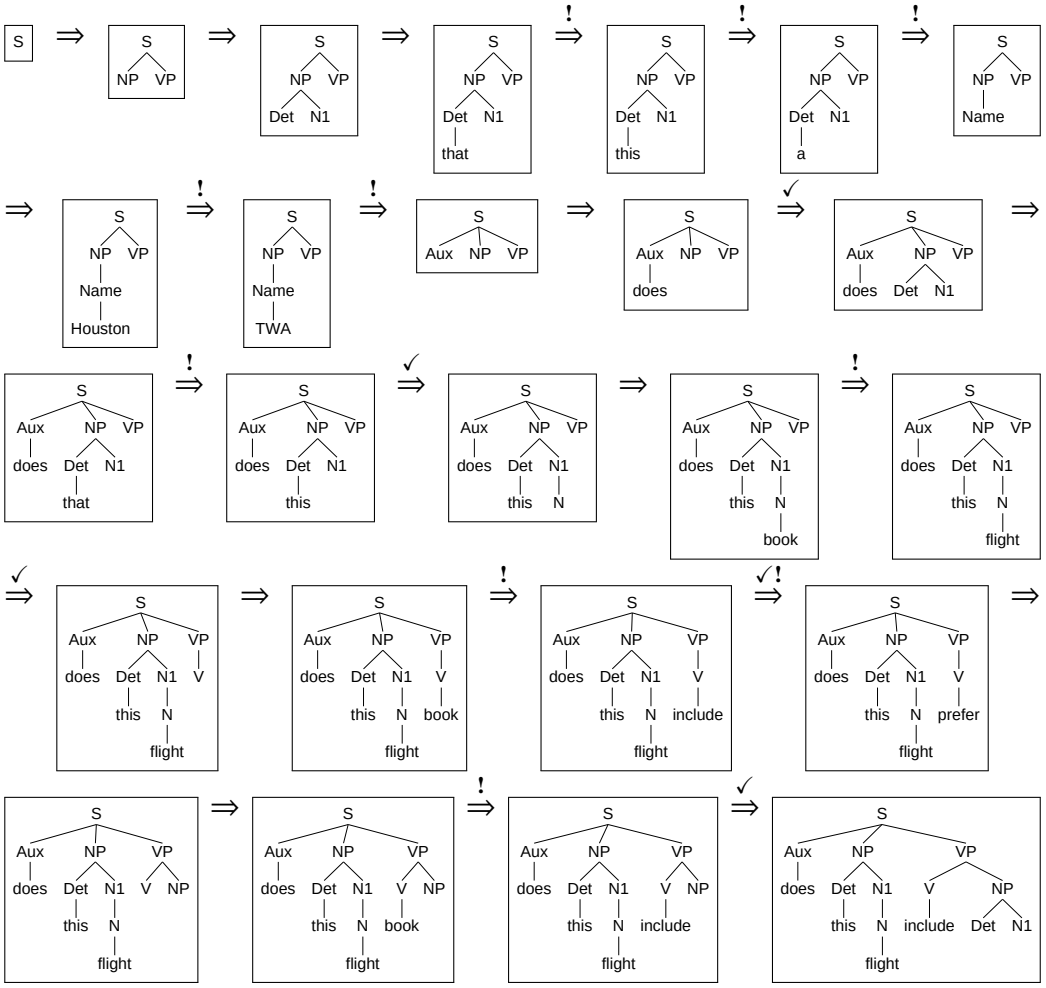
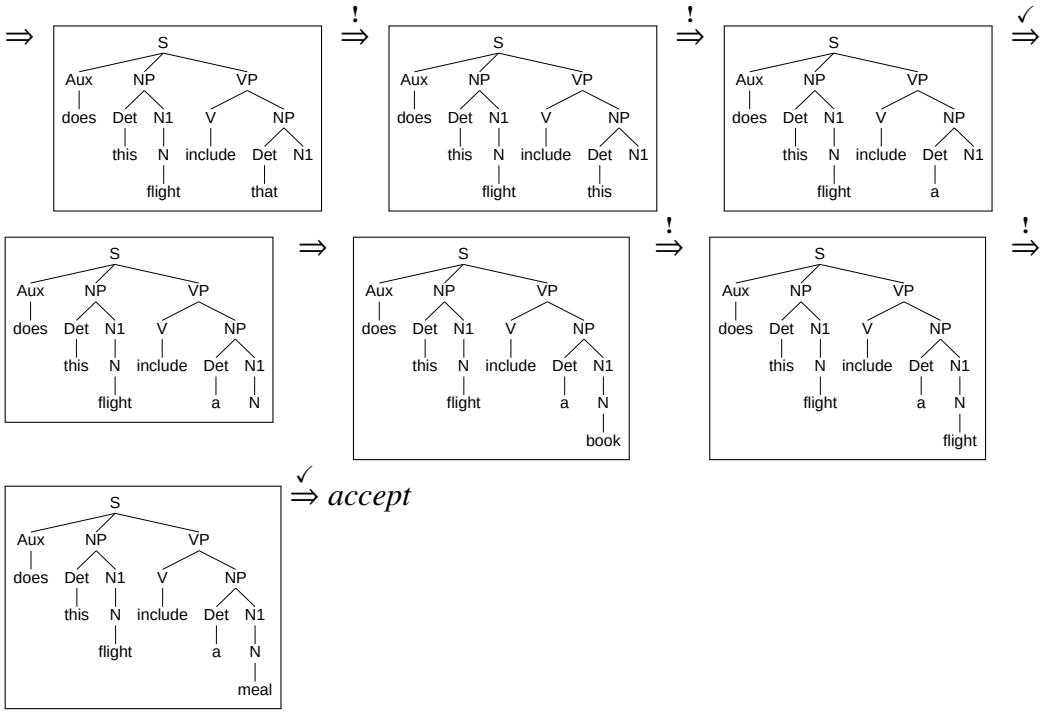


Figure 2.2: Part of the computation tree of the bottom-up recognizer for the input the dog chased the cat. Only the stack contents are shown.

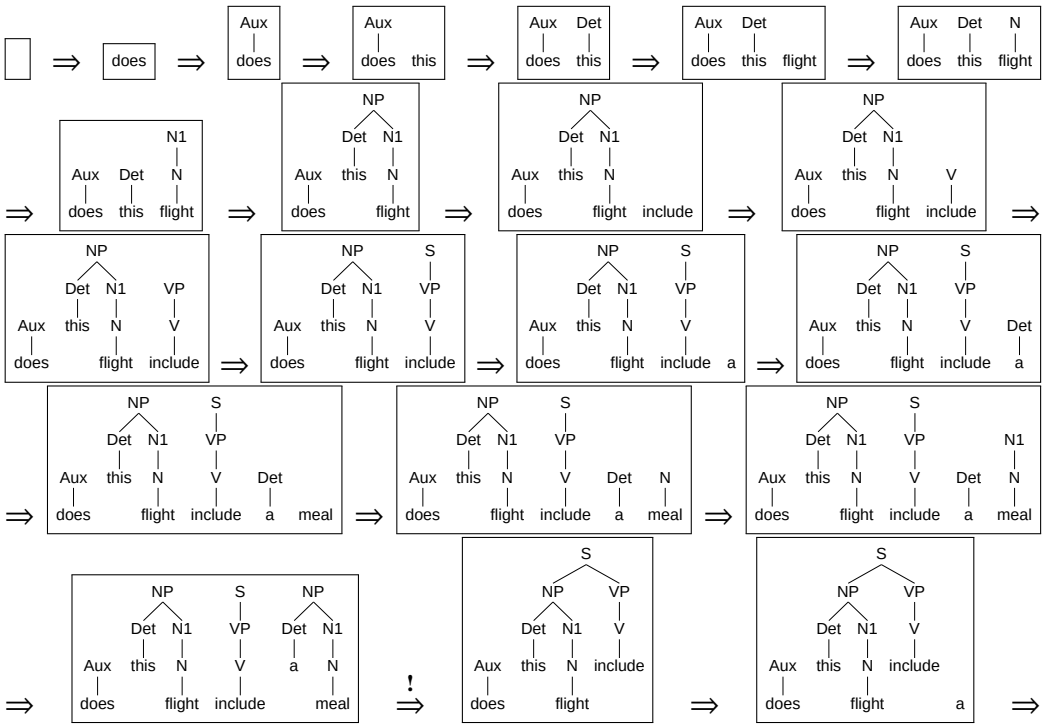
S → NP VP
S → Aux NP VP
S → VP
NP → Det N1
NP → Name
VP → V
VP → V NP
N1 → N
N1 → Name N1
N1 → N1 PP
PP → P NP
Aux → does
Det → that | this | a
Name → Houston | TWA
V → book | include | prefer
N → book | flight | meal | money
P → from | to | on

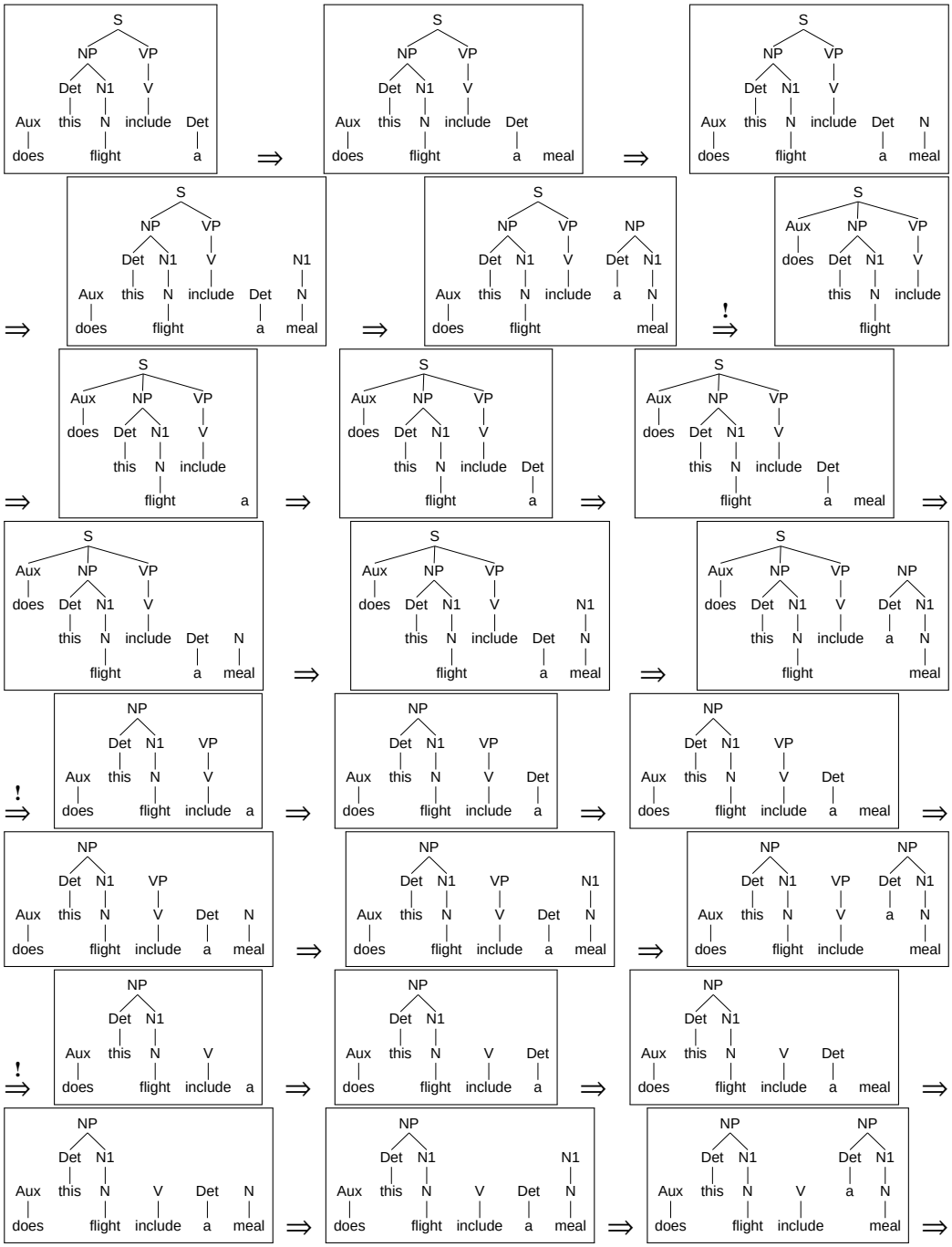
Here is how the top-down recognizer accepts the input does this flight inlcude a meal using backtracking. For readability, we show incomplete parse trees rather than stack contents. The *match* steps are combined with the next step and indicated with the check mark, and the stack contents correspond to the frontier of the tree minus the terminal nodes already matched against the input. The steps that involve backtracking are marked with "**!**".

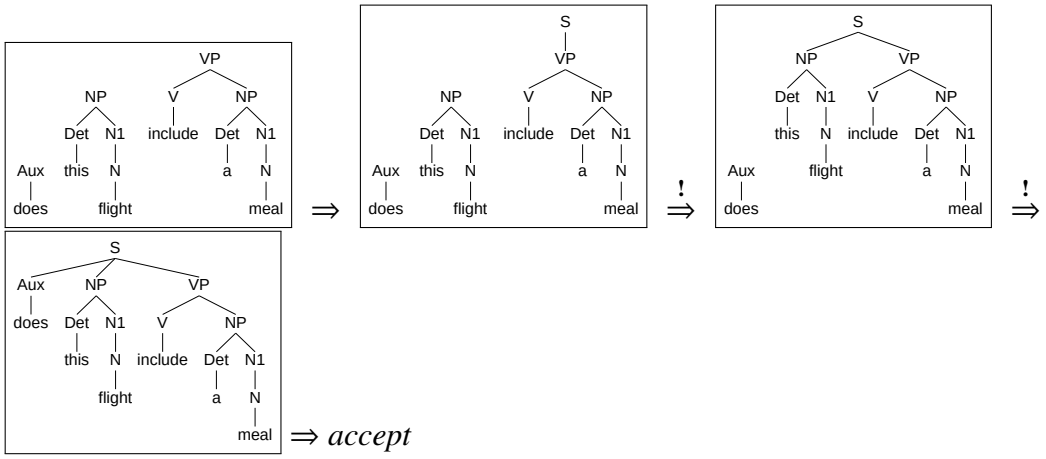




Here is how the backtracking bottom-up recognizer accepts the same input. The stack contents correspond to the sequence of root nodes of the trees. Again, the steps involving backtracking are indicated with **!**.




S NP VP Det N1 V Aux this N include Det does flight a
S NP VP Det N1 V Aux this N include Det does flight a meal
S NP VP Det N1 V Aux this N include Det N does flight a meal
S NP VP Det N1 V N1 Aux this N include Det N does flight a meal
S NP VP Det N1 V NP Aux this N include Det N does flight a meal
S Aux NP VP does Det N1 V this N include flight
S Aux NP VP does Det N1 V this N include Det flight a
S Aux NP VP does Det N1 V this N include Det N flight a meal
S Aux NP VP does Det N1 V this N include Det N flight a meal
S Aux NP VP does Det N1 V this N include Det N N1 flight a meal
S Aux NP VP does Det N1 V this N include Det N NP Det N flight a meal
NP Det N1 VP Aux this N V does flight include a
NP Det N1 VP Aux this N V Det does flight include a
NP Det N1 VP Aux this N V Det does flight include a meal
NP Det N1 VP Aux this N V Det N does flight include a meal
NP Det N1 VP N1 Aux this N V Det N does flight include a meal
NP Det N1 VP NP Det N1 Aux this N V a N does flight include meal
NP Det N1 Aux this N V Det does flight include a
NP Det N1 Aux this N V Det does flight include a meal
NP Det N1 Aux this N V Det N does flight include a meal
NP Det N1 N1 Aux this N V Det N does flight include a meal
NP Det N1 NP Det N1 Aux this N V a N does flight include meal



The need for backtracking can be greatly reduced for the top-down recognizer by pre-computing the set FIRST(X) for every nonterminal X:

$$\text{FIRST}(X) = \{\, a \in \Sigma \mid X \Rightarrow^* a\alpha \text{ for some } \alpha \,\}$$

and looking ahead at the next input symbol at the predict step, as follows:

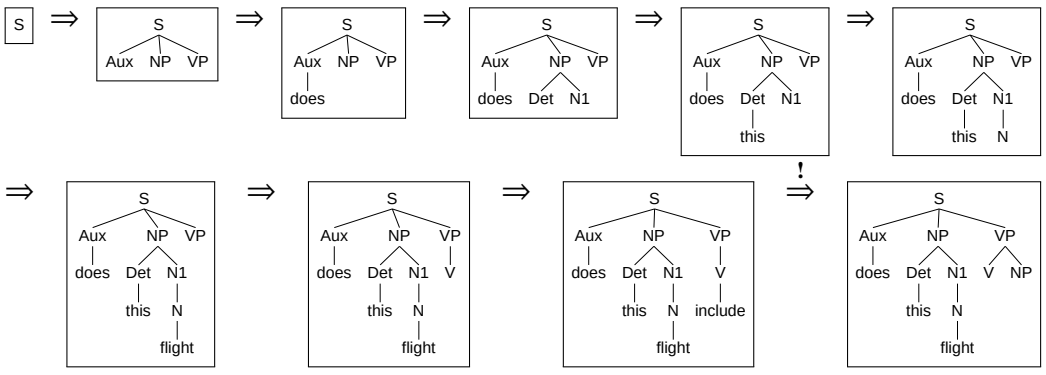
Predict: For each production $X \to Y_1 \ldots Y_n$ of the grammar,

$$(X\alpha, ax) \vdash (Y_1 \ldots Y_n\alpha, ax)$$

provided that $a = Y_1$ or $a \in \text{FIRST}(Y_1)$.[5]

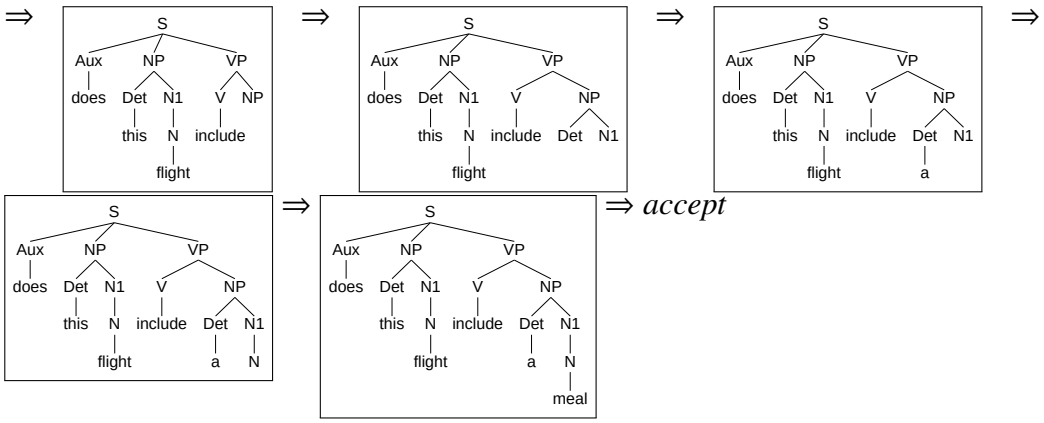
(The *match* steps will then be superfluous.) This idea is behind the $LL(1)$ method of deterministic parsing (see Aho and Ullman 1972).

Here is how the backtracking top-down recognizer with lookahead accepts the earlier input does this flight include a meal:



[5]This formulation assumes that the grammar does not contain ε-productions. In the general case, it is necessary to compute $\text{FIRST}(Y_1 \ldots Y_n) = \{\, a \in \Sigma \mid Y_1 \ldots Y_k \Rightarrow^* a\alpha \text{ for some } \alpha \,\} \cup \{\, \varepsilon \mid Y_1 \ldots Y_k \Rightarrow^* \varepsilon \,\}$ and $\text{FOLLOW}(X) = \{\, a \in \Sigma \mid S \Rightarrow^* \alpha X a \beta \text{ for some } \alpha, \beta \,\}$. The proviso becomes

either $a \in \text{FIRST}(Y_1 \ldots Y_n)$, or else $\varepsilon \in \text{FIRST}(Y_1 \ldots Y_n)$ and $a \in \text{FOLLOW}(X)$.



Whether or not lookahead is used, the backtracking top-down recognizer may loop (i.e., the computation tree may be infinite) when the grammar contains *left recursion*:

$$A \Rightarrow^+ A\alpha$$

Left recursion does not pose a problem for the backtracking bottom-up recognizer, but ε-productions and the presence of *cycles*

$$A \Rightarrow^+ A$$

cause it to loop.

Left recursion may be removed from a CFG, but the parse trees of the resulting grammar will look very different from those of the original grammar. Cycles and ε-productions may be removed without drastically changing the structures of parse trees.

No matter which method is used, simulating stack-based recognizers by systematically exploring the entire computation tree is inherently inefficient because the computation tree contains a great deal of redundant information. Since different branches of the computation tree encode incomplete derivations for the whole input, information about possible derivations of the same substring are often duplicated in different branches. For example, in Figure 2.2, the subtree under the node Det N chased is almost identical to that under the node NP chased.

Exercise 2.1. Expanding Figure 2.2, draw the entire computation tree of the bottom-up recognizer for the input the dog chased the cat.

Left-corner recognition and psychological plausibility

It is tempting to view the stack used in stack-based recognizers as a crude model of (part of) the working memory used by the human sentence processor. We might be

able to explain difficulties that humans encounter with certain complex sentences by pointing to the large amount of space required by the stack of the recognizer for such sentences, which presumably translates into heavy load on working memory. It is immediately obvious, however, that neither the top-down nor the bottom-up recognizer behaves in a way that gives this account any credence.

It is well-known at least since Miller and Chomsky 1963 that the humans have great trouble comprehending *center-embedded* structures like (2.1–2.3), while experiencing no comparable difficulty with *left-branching* structures like (2.4–2.6) or *right-branching* structures like (2.7):

(2.1) [The rat [that the cat [that the dog chased] bit] ate the cheese]

(2.2) [Katsuo ga [Sazae ga [Masuo ga katte kita] zairyou de
Katsuo-NOM Sazae-NOM Masuo-NOM buy came ingredients with
tsukutta] gochisou o zenbu tabete shimatta]
made feast-ACC all eat finished
'Katsuo ate up the feast that Sazae made with the ingredients that Masuo bought'

(2.3) [Sazae ga [Wakame ga [Tara-chan ga nakidashita] toki
Sazae-NOM Wakame-NOM Tara-chan-NOM started to cry time
me o samashita] no ni kizuita]
woke up that noticed
'Sazae noticed that Wakame woke up when Tara-chan started to cry'

(2.4) [[[[[John]'s uncle]'s friend]'s sister]'s son] appeared on TV

(2.5) [[[Masuo ga katte kita] zairyou de Sazae ga tsukutta] gochisou o Katsuo ga zenbu tabete shimatta]

(2.6) [[[Tara-chan ga nakidashita] toki Wakame ga me o samashita] no ni Sazae ga kizuita]

(2.7) [The dog chased the cat [that bit the rat [that ate the cheese]]]

The stack depth of the top-down recognizer increases in proportion to the length of sequences of *predict* moves that have not yet had the last nonterminal it introduced reach the top of the stack:

$$\begin{aligned}
(X\alpha, xy) &\vdash (Y_1 \ldots Y_n\alpha, xy) && \text{predict with } X \to Y_1 \ldots Y_n \ (n \geq 2)\\
&\vdash^* (ZY_{i+1} \ldots Y_n\alpha, y) && (i < n)\\
&\vdash (W_1 \ldots W_m Y_{i+1} \ldots Y_n\alpha, y) && \text{predict with } Z \to W_1 \ldots W_m \ (m \geq 2)
\end{aligned}$$

Here, the first *predict* move introduced nonterminals $Y_1, \ldots, Y_n$ on the stack, and the nonterminal Y_n was stil buried inside the stack at the time of the next *predict*

move in the sequence (introducing $W_1 \ldots W_m$), which means that the stack depth strictly increased between the two *predict* moves. Consequently, the stack depth of the top-down recognizer can get arbitrarily large on center-embedded and left-branching structures like (2.1)–(2.3) and (2.4)–(2.6), as the level of embedding increases, but it is bounded by a constant on purely right-branching structures like (2.7).

In the case of the bottom-up recognizer, in contrast, the stack depth increases with the number of shift moves that have yet to be matched by a corresponding reduce move that incorporates the shifted terminal into a larger constituent involving some preceding terminal(s):

$$\begin{aligned} (\alpha, axby) &\vdash (\alpha a, xby) && \text{shift} \\ &\vdash^* (\alpha X, by) \\ &\vdash^* (\alpha Xb, y) && \text{shift} \end{aligned}$$

Consequently, the maximal stack depth of the bottom-up recognizer is unbounded on center-embedded and right-branching structures like (2.1)–(2.3) and (2.7), but bounded on left-branching structures like (2.4)–(2.6).

Assuming the grammar in (2.8), Figure 2.3 shows the incomplete parse trees corresponding to the stages where the nondeterministic top-down recognizer reaches maximal stack depth in the course of accepting (2.1) and (2.4). Figure 2.4 shows the incomplete parse trees corresponding to the stages where the nondeterministic bottom-up recognizer reaches maximal stack depth in the course of accepting (2.1) and (2.7).

(2.8)

S → NP VP	S/NP → NP VP/NP
NP → Det N1	VP/NP → Vt
NP → Name	Det → the
VP → Vt NP	Name → John | TV
VP → Vi PP	Vt → chased | bit | ate
Det → NP Poss	Vi → appeared
N1 → N	N → dog | cat | rat | cheese | uncle
N1 → N1 RC	| friend | sister | son
PP → P NP	Poss → 's
RC → C VP	P → on
RC → C S/NP	C → that

The *left-corner* recognizer (Rosenkrantz and Lewis 1970, Aho and Ullman 1972, Johnson-Laird 1983, Resnik 1992) combines aspects of both the top-down and the bottom-up recognizers and is attractive on grounds of psychological plausibility because its stack depth is bounded on both purely left-branching and purely right-branching structures.

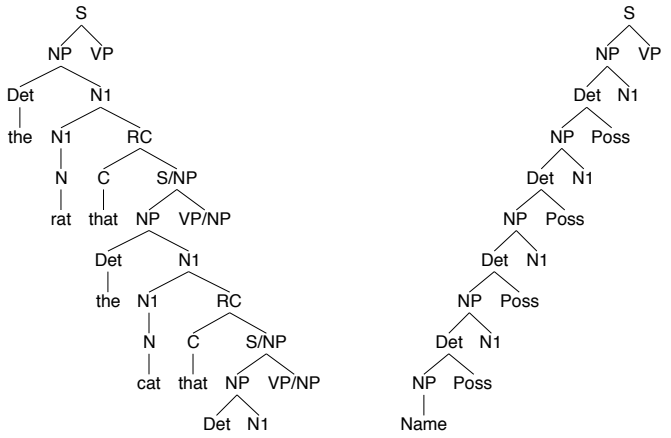


Figure 2.3: Stages of the top-down recognizer.

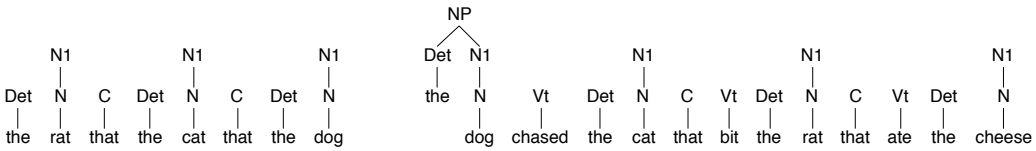


Figure 2.4: Stages of the bottom-up recognizer.

Left-corner recognizer The top of the stack is on the left. In addition to the grammar symbols, the stack symbols include symbols of the form $\sim X$, where X is any grammar symbol (terminal or nonterminal). The grammar is assumed to contain no ε-productions.

- Initial configuration:

  $$(\sim S, x)$$

  where x is the input.
- Possible moves:
  - *Shift:*

    $$(\alpha, ax) \vdash (a\alpha, x)$$

    where a is a terminal.
  - *Match:*

    $$(\sim a\alpha, ax) \vdash (\alpha, x)$$

    where a is a terminal.
  - *Reduce/predict:* For each production $X \to Y_1 \ldots Y_n$ of the grammar,

    $$(Y_1\alpha, x) \vdash (\sim Y_2 \ldots \sim Y_n X\alpha, x)$$

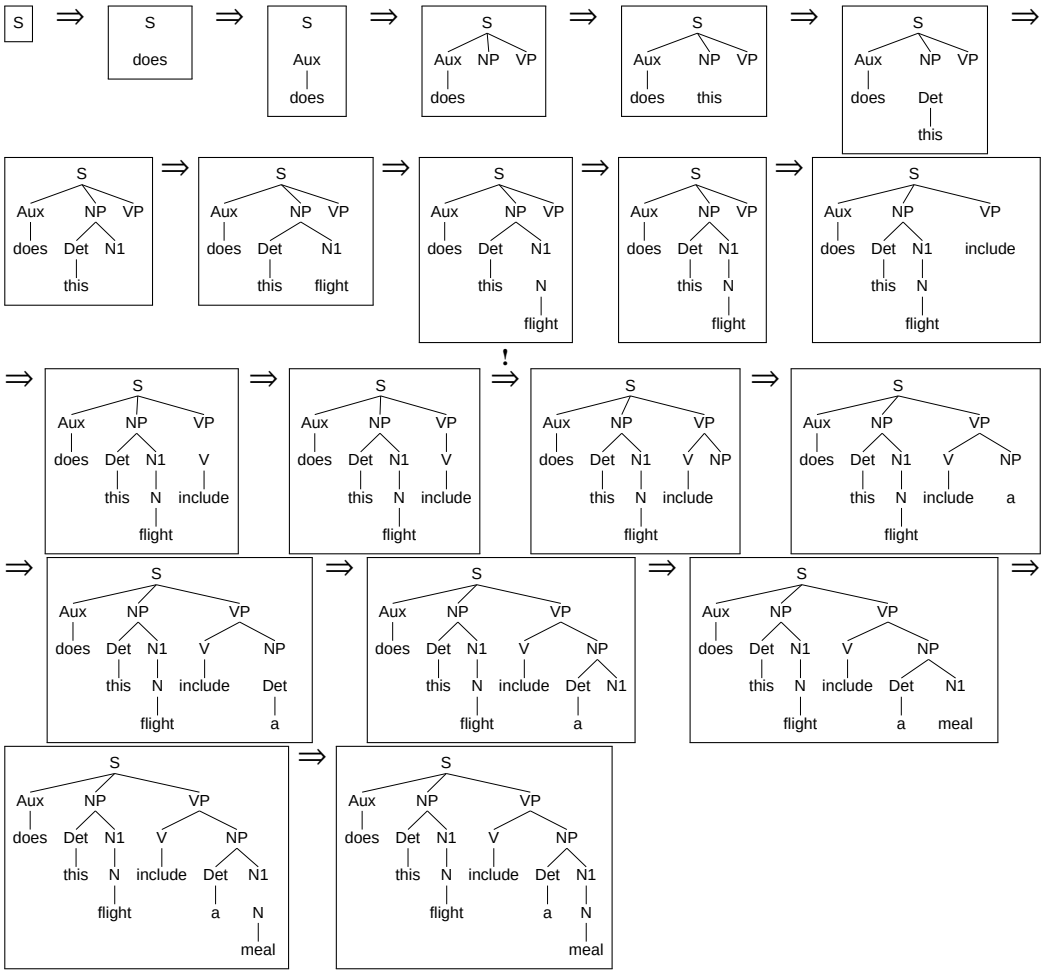
– *Reduce/predict/complete:* For each production $X \to Y_1 \ldots Y_n$ of the grammar,

$$(Y_1 \sim X\alpha, x) \vdash (\sim Y_2 \ldots \sim Y_n \alpha, x)$$

- Accepting configuration:

$$(\varepsilon, \varepsilon)$$

The symbol $\sim X$ means "X is predicted", while X on the top of the stack means "X has been recognized". Here is how the backtracking left-corner recognizer accepts the input does this flight include a meal. The stack contents correspond to the nodes that have not yet been connected to all of the neighboring nodes.



Exercise 2.2. Assuming the grammar in (2.8), draw the incomplete parse trees corresponding to stages where the nondeterministic left-corner recognizer reaches maximal stack depth in the course of accepting (2.1), (2.4), and (2.7).

Exercise 2.3. (a) Show that the *shift* rule can be modified as follows:

Shift:

$$(\sim X\alpha, ax) \vdash (a\sim X\alpha, x)$$

where a is a terminal and X is a nonterminal.

(b) What goes wrong if the *reduce/predict* rule of the left-corner recognizer is changed to the following form?

Reduce/predict: For each production $X \to Y_1 \ldots Y_n$ of the grammar,

$$(Y_1\sim Z\alpha, x) \vdash (\sim Y_2 \ldots \sim Y_n X\sim Z\alpha, x)$$

where $X \neq Z$.

Exercise 2.4. Let G be a CFG without ε-productions. Prove that the left-corner recognizer for G satisfies the following properties by simultaneous induction:

(a) $V \Rightarrow_G^* x$ implies $(\sim V, x) \vdash^* (\varepsilon, \varepsilon)$.

(b) $V \Rightarrow_G^+ Xy$ implies $(X\sim V, y) \vdash^* (\varepsilon, \varepsilon)$.

Exercise 2.5. Let G be a CFG without ε-productions, and consider the left-corner recognizer for G.

(a) Prove the following properties by simultaneous induction:

(i) $(\sim V\alpha, x) \vdash^* (\beta, \varepsilon)$, $|\alpha| \geq |\beta|$ implies that there exist $u, v \in \Sigma^*$ such that $x = uv$, $(\sim V, u) \vdash^* (\varepsilon, \varepsilon)$ and $(\alpha, v) \vdash^* (\beta, \varepsilon)$.

(ii) $(U\sim V\alpha, x) \vdash^* (\beta, \varepsilon)$, $|\alpha| \geq |\beta|$ implies that there exist $u, v \in \Sigma^*$ such that $x = uv$, $(U\sim V, u) \vdash^* (\varepsilon, \varepsilon)$ and $(\alpha, v) \vdash^* (\beta, \varepsilon)$.

(b) Prove the following properties by simultaneous induction:

(I) $(\sim Z, x) \vdash^* (\varepsilon, \varepsilon)$ implies $Z \Rightarrow_G^* x$.

(II) $(Y\sim X, x) \vdash^* (\varepsilon, \varepsilon)$ implies $X \Rightarrow_G^+ Yx$.

The left-corner recognizer we just presented corresponds to the *arc-eager* strategy of Abney and Johnson (1991) (see Resnik 1992). The version of the left-corner recognizer following the *arc-standard* strategy is slightly simpler, but does not have the same psychological plausibility of the arc-eager version.

Left-corner recognizer, arc-standard The top of the stack is on the left. In addition to the grammar symbols, the stack symbols include symbols of the form $\sim X$, where X is any grammar symbol (terminal or nontermial). The grammar is assumed to contain no ε-productions.

- Initial configuration:
$$(\sim S, x)$$
where x is the input.
- Possible moves:
  - *Shift:*
$$(\alpha, ax) \vdash (a\alpha, x)$$
where a is a terminal.
  - *Match:*
$$(\sim a\alpha, ax) \vdash (\alpha, x)$$
where a is a terminal.
  - *Reduce/predict:* For each production $X \to Y_1 \dots Y_n$ of the grammar,
$$(Y_1\alpha, x) \vdash (\sim Y_2 \dots \sim Y_n X\alpha, x)$$
  - *Complete:*
$$(X \sim X\alpha, x) \vdash (\alpha, x)$$
where X is a nonterminal.
- Accepting configuration:
$$(\varepsilon, \varepsilon)$$

Exercise 2.6. Repeat Exercise 2.2, this time using the arc-standard version of the left-corner recognizer.

Human sentence processing

The exact conditions that are responsible for the processing overload effect are by no means as simple as is suggested by the simple pushdown automata model that we have seen above. The following sentences illustrate the complexity of the data that have to be accounted for by any successful theory of human sentence processing (Gibson 1998, 2000):

(2.9) a. The fact that the employee who the manager hired stole office supplies worried the executive.

b. # The executive who the fact that the employee stole office supplies worried hired the manager.

(2.10) a. The student who the professor who I collaborated with had advised copied the article.

b. The student who the professor who Jen collaborated with had advised copied the article.

c. The student who the professor who the scientist collaborated with had advised copied the article.

(2.11) a. # The intern who the information that the doctor hated the patient had bothered supervised the nurse.

b. # The intern who the information that Joe hated Sue had bothered supervised the nurse.

c. The intern who the information that you hated me had bothered supervised the nurse.

d. The intern who the information that you hated him had bothered supervised the nurse.

(2.12) a. The school board which the teachers who were neglecting the students had angered troubled the superintendent.

b. The school board which the teachers neglecting the students had angered troubled the superintendent.

(2.13) a. # Which box did you put the very large and beautifully decorated wedding cake bought from the expensive bakery in?

b. In which box did you put the very large and beautifully decorated wedding cake bought from the expensive bakery?

(2.14) Which box did the very tall and beautifully dressed bride who was married in the church down the hill put the cake in?

In (2.9), a sentential complement embedded inside a relative clause causes more difficulty than a relative clause embedded inside a sentential complement. In (2.10) and (2.11), the presence of pronouns rather than full noun phrases in the most deeply embedded clause makes the sentence easier to process. Similarly, in (2.12), it is easier to process the sentence in which the most deeply embedded clause lacks tense. Under certain conditions, preposition stranding causes great difficulty (2.13), but not always (2.14).

Based on data such as these, Gibson (1998, 2000) has proposed (two slightly different) models that assume that sentence parsing involves two types of cost:

1. storage cost: the cost associated with keeping track of expected syntactic elements.

2. integration cost: the cost associated with connecting an incoming word to the structure that has been built thus far.

One of Gibson's key ideas is that the cost of integrating two elements (such as a head and a dependent) depends on the distance between the two: the longer the distance is, the more cost the integration incurs.[6]

Although recent research on human sentence processing has accumulated a large body of experimental data and offers interesting hypotheses that seem to account for a wide range of phenomena, it is regrettable that it is not usually accompanied by a precisely formulated computational model, and is consequently largely independent of the advances in the theory of parsing of formal languages.[7]

Problems

2.1. Let G be a CFG without ε-productions, and consider the (arc-eager) left-corner recognizer for G. Prove the following:

(III) $(\sim X, x) \vdash^* (\sim Z, \varepsilon)$ implies $X \Rightarrow_G^* xZ$.

(IV) $(Y, x) \vdash^* (X, \varepsilon)$ implies $X \Rightarrow_G^* Yx$.

(V) $(\varepsilon, x) \vdash^* (Y, \varepsilon)$ implies $Y \Rightarrow_G^* x$.

2.2. Let $G = (N, \Sigma, P, S)$ be a CFG in *Chomsky normal form*,[8] and consider the (arc-eager) left-corner recognizer M for G.

(a) Suppose that α is a possible stack content in an accepting computation of M. In other words, suppose that there exist x, y such that $(\sim S, xy) \vdash^* (\alpha, y) \vdash^* (\varepsilon, \varepsilon)$. Show

$$\alpha \in (N \cup \Sigma \cup \{\varepsilon\})(\sim(N \cup \Sigma)(N \cup \Sigma))^* \sim (N \cup \Sigma).$$

(b) Suppose $(\sim S, uw) \vdash^* (\sim V X_n \sim Z_n \ldots X_1 \sim Z_1, w) \vdash^* (\varepsilon, \varepsilon)$. Show that there exist $x_1, \ldots, x_n \in \Sigma^*$, $y_1, \ldots, y_n, v, z_1, \ldots, z_n \in \Sigma^+$ and $V_1, \ldots, V_{n+1}, Y_1, \ldots, Y_n \in N$ such that

- $u = x_1 y_1 \ldots x_n y_n$,

[6] Another line of research that contrasts with the *resource limitation* approach exemplified by Gibson's theory is the *resource allocation* approach, which holds that "the parser allocates different amounts of resources to different interpretations of the partial input, and difficulty arises when those resources turn out to be inefficiently allocated" (Levy 2008).

[7] But see Levy 2008, Hale 2001, 2003, 2006, 2011.

[8] A CFG is in Chomsky normal form if every production is of the form $A \to BC$ or $A \to a$, where B, C are nonterminals and a is a terminal.

- $w = vz_n \ldots z_1$,
- $V_1 = S$,
- $V_{n+1} = V$,

and for each $i = 1, \ldots, n$,

- $(\sim V_i, x_i) \vdash^* (\sim Z_i, \varepsilon)$,
- $(\varepsilon, y_i) \vdash^* (Y_i, \varepsilon)$,
- $X_i \rightarrow Y_i V_{i+1}$ is a production of G,
- $(\sim V, v) \vdash^* (\varepsilon, \varepsilon)$,
- $(X_i \sim Z_i, z_i) \vdash^* (\varepsilon, \varepsilon)$.

(c) Show that if G is not self-embedding, the maximal stack height in an accepting computation of M is bounded.[9]

(d) Conclude that if G is not self-embedding, $L(G)$ is regular.

2.3. Let G be a CFG without ε-productions or *unit productions* (i.e., productions of the form $X \rightarrow Y$, where Y is a nonterminal), and consider the (arc-eager) left-corner recognizer M for G. Show that if G is not self-embedding, the maximal stack height in an accepting computation of M is bounded. Show that the assumption that G does not contain ε- or unit productions is necessary.

2.4. Extend both types of left-corner recognizer so as to be able to handle CFGs with ε-productions.

References

Abney, Steven and Mark Johnson. 1991. Memory requirements and local ambibuities for parsing strategies. *Journal of Psycholinguistic Research* 20(3):233–250.

Aho, Alfred V. and Jeffrey D. Ullman. 1972. *The Theory of Parsing, Translation, and Compiling. Volume I: Parsing*. Englewood Cliffs, N.J.: Prentice Hall.

Chomsky, Noam. 1962. Context-free grammars and pushdown storage. In *Quarterly Progress Report no. 65*, pages 187–194. Cambridge, Mass.: MIT Research Laboratory of Electronics.

Chomsky, Noam. 2004. *Generative Enterprise Revisited*. Berlin: Mouton de Gruyter.

[9] We use "stack depth" and "stack height" interchangeably.

Gibson, Edward. 1998. Linguistic complexity: locality of syntactic dependencies. *Cognition* 68:1–76.

Gibson, Edward. 2000. The dependency locality theory: a distance-based theory of linguistic comprehension. In A. Marantz, Y. Miyashita, and W. O'Neil, eds., *Image, Language, Brain*, pages 95–126. Cambridge, Mass.: MIT Press.

Hale, John T. 2001. A probabilistic Earley parser as a psycholinguistic model. In *Proceedings of the Second Meeting of the North American Chapter of the Association for Computational Linguistics*.

Hale, John T. 2003. The information conveyed by words in sentences. *Journal of Psycholinguistic Research* 32(2):101–123.

Hale, John T. 2006. Uncertainty about the rest of the sentence. *Cognitive Science* 30(4):609–642.

Hale, John T. 2011. What a rational parser would do. *Cognitive Science* 35:399–443.

Johnson-Laird, Philip N. 1983. *Mental Models*. Cambridge, Mass.: Harvard University Press.

Jurafsky, Daniel and James H. Martin. 2009. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, Second Edition*. Upper Saddle River, New Jersey: Prentice Hall.

Levy, Roger. 2008. Expectation-based syntactic comprehension. *Cognition* 106:1126–1177.

Miller, George A. and Noam Chomsky. 1963. Finitary models of language users. In R. D. Luce, R. R. Bush, and E. Galanter, eds., *Handbook of Mathematical Psychology, Volume II*, pages 419–491. New York: John Wiley and Sons.

Resnik, Philip. 1992. Left-corner parsing and psychological plausibility. In *Proceedings of the Fourteenth International Conference on Computational Linguistics (COLING '92)*, pages 191–197.

Rosenkrantz, D.J. and P.M. Lewis, II. 1970. Deterministic left corner parser. In *IEEE Conference Record of the 11th Annual Symposium on Switching and Automata*, pages 139–152.

Ruzzo, Walter L. 1979. On the complexity of general context-free language parsing and recognition. In H. A. Maurer, ed., *Automata, Languages and Programming*, pages 489–497. Berlin: Springer.