19/4/87

1. Graph Unification and PATR-II continued

We saw above how features on grammar rules could be interpreted by means of the operation of unification, but that representing feature structures as terms (i.e. suitable for direct interpretation by Prolog) gave rise to certain problems or inelegancies in grammar writing. Then we discussed an alternative representation as DAGs (or sets of name:value pairs) which obviated this problem, and started to look at a formalism called PATR-II which supports grammar rules over structures of this form.

The skeleton of a PATR-II rule is a context-free rule which introduces variable names for each of the feature structures (DAGs) in the rule, and the remainder of the rule consists of a set of equations which define the values of those variables in terms of paths through the DAGs. Thus the rule:

```
X0 -> X1 X2
<X0 cat> = s
<X1 cat> = np
<X2 cat> = vp
<X1 agr> = <X2 agr>
```

category

unifies ~~with~~ with agreement structure

has the following interpretation:

Suppose the feature structure X_1 is associated (by the lexicon, or by application of a previous rule) with some string of words S_1, and similarly X_2 with S_2. Then the feature structure X_0 is associated with the string of words obtained by concatenating S_1 and S_2 (or vice versa). The values of the X_i are given by the path equations associated with the rule. Notice that the path equations indistinguishably constrain or define the feature structures. That is, this interpretation of a rule is totally declarative - no particular parsing strategy or order of solution of equations is assumed or will make any difference to the final result. Asking for the value of a feature and giving a value to a feature are the same operation.

Here is another example of a PATR-II rule for further illustration. This handles the same phenomenon as the above rule, i.e. subject-verb (phrase) agreement, but in a different way. It is assumed that each DAG has two principal features - cat, whose values range over the standard syntactic categories, and head - whose values are complex, i.e. DAGs themselves. Head features are those which are shared between the lhs category in a rule and that constituent on the rhs which is deemed the 'head' of the rule. Nouns are the heads of noun phrases, verbs of verb phrases, verb phrases of sentences, etc.

```
X0 -> X1 X2
<X0 cat> = s
<X1 cat> = np
<X2 cat> = vp
<X0 head> = <X2 head>
<X2 head subj> = <X1 head>
```

Notice that agreement features are not mentioned at all. They will have been given values in the lexical entries as below, and the final two equations in the rule ensure that the appropriate agreement behaviour follows.

```
X0 -> fred
<X0 cat> = np
<X0 agr num> = sing        <X0 head agr num> = sing
<X0 agr pers> = 3          <X0 head agr person> = 3
```

19/11/87

X_0 -> sleeps
<X_0 cat> = vp
<X_0 form> = finite
<X_0 head subj agr num> = sing
<X_0 head subj agr pers> = 3

Applying the rule to the two lexical entries will give the following pairings of strings of words and associated feature structures:

fred sleeps:

$$\begin{bmatrix} \text{cat:} & \text{s} \\ \text{head:} & [1] \begin{bmatrix} \text{form:} & \text{finite} \\ \text{subj:} & [2] \begin{bmatrix} \text{agr:} & \begin{bmatrix} \text{num:} & \text{sing} \\ \text{pers:} & 3 \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

See dags of X_0, X_1 & X_2 in written notes.

fred:

$$\begin{bmatrix} \text{cat:} & \text{np} \\ \text{head:} & [2] \end{bmatrix}$$

sleeps:

$$\begin{bmatrix} \text{cat:} & \text{vp} \\ \text{head:} & [1] \end{bmatrix}$$

2. Graph Unification Algorithm

In order to give some insight into how graph unification might be implemented, here is an algorithm in Prolog. Only the interpretation of the equal signs in the path equations is given. The computation of the paths themselves is left as an exercise for the reader.

```
% equal defines the equality (i.e.unification) of pairs of
% DAGs each represented as a list of attribute:value pairs
% with variable tail

equal(X,X) :- !.
equal([A:V1|R1],F2) :-
    del(A:V2,F2,R2),
    equal(V1,V2),
    equal(R1,R2).
```

same atom

```
% del(Element,L0,L1) if L1 is L0 with no occurences of Element. The
% tricky thing is that Element is always in L0 after execution, since
% the latter is a list with variable tail

del(F,[F|X],X) :- !.
del(F,[E|X],[E|Y]) :-
   del(F,X,Y).
```

19/11/87

```
test(X,Y) :-
      X = [a1:v1,a2:v2,a3:[a11:v11,a21:v21|_]|_],
      Y = [a2:v2,a4:v4,a3:[a21:v21,a31:v31|_]|_],
      equal(X,Y).


| ?- test(X,Y).

Y = [a2:v2,a4:v4,a3:[a21:v21,a31:v31,a11:v11|_112],a1:v1|_57]
X = [a1:v1,a2:v2,a3:[a11:v11,a21:v21,a31:v31|_112],a4:a4|_57]

yes
```

The remainder of this handout is culled from Shieber's "Introduction to Unification-Based Approaches to Grammar". Note that a notational convention has been adopted, whereby any symbol in the context-free part of a rule other than a (subscripted) X is interpreted as the value of the feature cat, as well as the variable corresponding to the DAG concerned. For example, the last rule above would simplify to:

```
S -> NP VP
<S head> = <VP head>
<S head subj> = <NP head>
```

This notational convention would be somewhat tricky to incorporate in a Prolog implementation of PATR-II.

The astute reader will notice that the first sample grammar allowed no postverbal complements of verbs, a considerable limitation. Our second grammar deals with the problem of lexical selection of postverbal "subcategorization frames"—the manner in which, for example, the verb "storm" (as in "Uther storms Cornwall") lexically selects (*subcategorizes for*) a single postverbal NP, whereas "persuade" subcategorizes for an NP and an infinitival VP as complements.

A simple solution would be to add rules of the form

$$\begin{array}{ll} VP \rightarrow V\ NP & \\ \quad \langle VP\ head \rangle = \langle V\ head \rangle & \qquad (R_3) \end{array}$$

and

$$\begin{array}{ll} VP_1 \rightarrow V\ NP\ VP_2 & \\ \quad \langle VP_1\ head \rangle = \langle V\ head \rangle & \qquad (R_4) \\ \quad \langle VP_2\ head\ form \rangle = infinitival & \end{array}$$

and so on, one for each subcategorization frame. The problem of matching up verbs with a VP rule could be achieved (as usual) with unification. A feature *subcat* in the verb's feature structure would be forced to unify with an arbitrary value specified in the rule. The following rules and lexical entries achieve such a matching.

$$\begin{array}{ll} VP \rightarrow V\ NP & \\ \quad \langle VP\ head \rangle = \langle V\ head \rangle & \qquad (R'_3) \\ \quad \langle V\ subcat \rangle = np & \end{array}$$

$$\begin{array}{ll} VP_1 \rightarrow V\ NP\ VP_2 & \\ \quad \langle VP_1\ head \rangle = \langle V\ head \rangle & \\ \quad \langle VP_2\ head\ form \rangle = infinitival & \qquad (R'_4) \\ \quad \langle V\ subcat \rangle = npinf & \end{array}$$

$$storms \longmapsto \left[\begin{array}{ll} cat: & V \\ head: & \left[\begin{array}{ll} form: & finite \\ subject: & \left[\begin{array}{ll} agreement: & \left[\begin{array}{ll} number: & singular \\ person: & third \end{array}\right] \end{array}\right] \end{array}\right] \\ subcat: & np \end{array}\right]$$

$$persuades \longmapsto \left[\begin{array}{ll} cat: & V \\ head: & \left[\begin{array}{ll} form: & finite \\ subject: & \left[\begin{array}{ll} agreement: & \left[\begin{array}{ll} number: & singular \\ person: & third \end{array}\right] \end{array}\right] \end{array}\right] \\ subcat: & npinf \end{array}\right]$$

Early GPSG used this type of analysis with some forty basic verb phrase rules.

In the second grammar, we adopt a more radical approach that takes fuller advantage of the power of unification. Just as the first grammar had a "slot" for the subject NP complement, the second grammar uses slots for all the complements, both pre- and postverbal. This is achieved through the feature structure encoding of a list with features *first* and *rest* and end marker value *end*. The slots in the list correspond to the complements in the following order: postverbal complements from left to right, followed by the preverbal subject. For instance, the lexical entry for the verb "storms" would be given by the pairing

19/11/87

$$
\textit{sleeps} \longmapsto
\left[\begin{array}{ll}
\textit{cat:} & V \\
\textit{head:} & \left[\textit{form: finite}\right] \\
\textit{subcat:} & \left[\begin{array}{ll}
\textit{first:} & \left[\textit{cat: NP}\right] \\
\textit{rest:} & \left[\begin{array}{ll}
\textit{first:} & \left[\begin{array}{ll}
\textit{cat:} & NP \\
\textit{head:} & \left[\textit{agreement:} \left[\begin{array}{l}\textit{number: singular} \\ \textit{person: third}\end{array}\right]\right]
\end{array}\right] \\
\textit{rest:} & \textit{end}
\end{array}\right]
\end{array}\right]
\end{array}\right]
$$

while for the verb "persuades" we would have the even more complex pairing

$$
\textit{persuades} \longmapsto
\left[\begin{array}{ll}
\textit{cat:} & V \\
\textit{head:} & \left[\textit{form: finite}\right] \\
\textit{subcat:} & \left[\begin{array}{ll}
\textit{first:} & \left[\textit{cat: NP}\right] \\
\textit{rest:} & \left[\begin{array}{ll}
\textit{first:} & \left[\begin{array}{ll}
\textit{cat:} & VP \\
\textit{head:} & \left[\textit{form: infinitival}\right] \\
\textit{subcat:} & \left[\textit{rest: end}\right]
\end{array}\right] \\
\textit{rest:} & \left[\begin{array}{ll}
\textit{first:} & \left[\begin{array}{ll}
\textit{cat:} & NP \\
\textit{head:} & \left[\textit{agreement:} \left[\begin{array}{l}\textit{number: singular} \\ \textit{person: third}\end{array}\right]\right]
\end{array}\right] \\
\textit{rest:} & \textit{end}
\end{array}\right]
\end{array}\right]
\end{array}\right]
\end{array}\right]
$$

The convolu[illegible] *subcat* value here lists the complements of "persuades" as, in order, an NP (the object), a VP whose form is infinitival and whose subcategorization requirement is a single element list (i.e., only the subject is missing), and the subject NP itself (marked as third-person singular, to fold in the agreement conditions).

As each postverbal complement is concatenated onto the VP, its feature structure is unified with the next slot in the list. Thus, the verb can impose requirements on its complements—e.g., category requirements, or requiring a VP complement to be infinitival—by adding the appropriate features to the slot, as was done to ensure agreement with the subject in the previous grammar. Let us look at one such VP-forming rule that adds an NP complement to the VP.

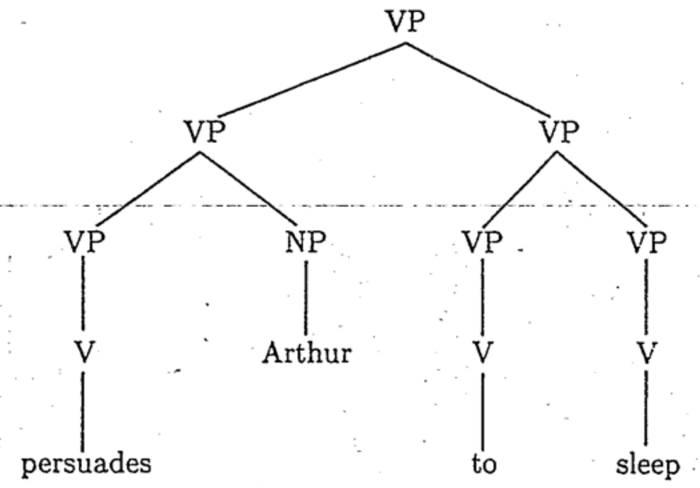
$$
\begin{array}{l}
VP_1 \rightarrow VP_2\ NP \\
\qquad \langle VP_1\ \textit{head} \rangle = \langle VP_2\ \textit{head} \rangle \\
\qquad \langle VP_2\ \textit{subcat first} \rangle = \langle NP \rangle \\
\qquad \langle VP_2\ \textit{subcat rest} \rangle = \langle VP_1\ \textit{subcat} \rangle
\end{array}
\tag{R_5}
$$

The unifications in R_5 require, respectively, that

- Head features are shared by the VPs.
- The NP is unified with the first remaining slot in the subcategorization frame for the VP.
- The subcategorization frame for the newly formed VP is that of the shorter VP minus the first element just found.

The grammar therefore builds a left-recursive structure for verb phrases, so that, for instance, the phrases "persuades," "persuades Arthur," and "persuades Arthur to sleep" will all be VPs—the first subcategorizing an NP, a VP and a subject NP (as just seen in the foregoing lexical entry), the second a VP and a subject NP, and the third just the subject NP. The phrase structure for this final VP in accordance with this grammar would be

9/11/98



Of course, a similar rule would be required for VP complements.

$$\begin{aligned} &VP_1 \rightarrow VP_2 \ VP_3 \\ &\quad \langle VP_1 \ head \rangle = \langle VP_2 \ head \rangle \\ &\quad \langle VP_2 \ subcat \ first \rangle = \langle VP_3 \rangle \\ &\quad \langle VP_2 \ subcat \ rest \rangle = \langle VP_1 \ subcat \rangle \end{aligned} \qquad (R_6)$$

How does this left recursion bottom out? We add a rule for just this purpose.

$$\begin{aligned} &VP \rightarrow V \\ &\quad \langle VP \ head \rangle = \langle V \ head \rangle \\ &\quad \langle VP \ subcat \rangle = \langle V \ subcat \rangle \end{aligned} \qquad (R_7)$$

Finally, we need a rule to form sentences from NPs and *VPs whose subcategorization frame requires no more postverbal complements.* This latter condition is verified by unifying the *rest* of the frame with the end marker value *end.* We also unify the subject NP with the last remaining element in the frame.

$$\begin{aligned} &S \rightarrow NP \ VP \\ &\quad \langle S \ head \rangle = \langle VP \ head \rangle \\ &\quad \langle S \ head \ form \rangle = finite \\ &\quad \langle VP \ subcat \ first \rangle = \langle NP \rangle \\ &\quad \langle VP \ subcat \ rest \rangle = end \end{aligned} \qquad (R_8)$$

A final optimization can be made. Rather than having separate rules for each possible category of postverbal complement, we can substitute the following single general rule:[9]

$$\begin{aligned} &VP_1 \rightarrow VP_2 \ X \\ &\quad \langle VP_1 \ head \rangle = \langle VP_2 \ head \rangle \\ &\quad \langle VP_2 \ subcat \ first \rangle = \langle X \rangle \\ &\quad \langle VP_2 \ subcat \ rest \rangle = \langle VP_1 \ subcat \rangle \end{aligned} \qquad (R_9)$$

Thus instead of the forty basic rules (and the additional rules derived by metarule) that the early GPSG analysis postulated, we need just this one.

9/4/89

Artificial Intelligence/Computer Science 3: Natural Language Processing

Syllabus:

1. Introduction: Natural Language Processing as Knowledge Engineering (1)

- types of knowledge – phonetic, orthographic, morphological, syntactic, semantic, pragmatic, discourse, real-world.

Readings: Winograd, chap. 1

2. Natural Languages and Formal Languages (2)

- The Chomsky Heirarchy
- Weak and Strong Generative Capacity
- Finite State Phenomena
- Context-free Phenomena
- ✓ Non-context free phenomena
- Competence and Performance

Readings: Handout

3. Definite Clause Grammars (2)

- ✓ review of syntactic categories and structure of English
- Prolog interpretation, structure-building etc.

Readings:
Pereira and Warren: Definite Clause Grammars for NL Analysis, in Readings in NLP
Winograd, 3.1 – 3.5

4. Chart Parsers (3)

- procedural limitations of Prolog interpretation of DCGs
- Chart as Well-Formed Substring Table, the Representation of Ambiguity
- ✓ CKY Algorithm, Dotted Rules,
- Earley algorithm, Proposing, Scanning, Completing, Occurs Check
- Categorial Grammar Notation

Readings:
Earley: An Efficient Context-Free Parsing Algorithm, in Readings in NLP
Kay: Algorithm Schemata and Data Structures in Syntactic Processing, in Readings in NLP
Winograd: 3.6

5. Unification-based Approaches to Grammar (3)

- Introduction and Motivation for Unification 7
- Morphology and the Lexicon
- Current Grammatical Theories 6.

Readings: Shieber, Ritchie et al.,

6. Semantics (2)

- What is Semantics?
- Logic and other Semantic Representation Languages
- Compositionality
- Montague Semantics

Readings:
Handout
Schubert and Pelletier, From English to Logic, in Readings in NLP

7. Anaphoric Reference (2)

Readings:
Hobbs, Resolving Pronoun References, in Readings in NLP
Webber, So What Can We Talk About Now, in Readings in NLP

8. Generation (2)

Readings:
The Generation section in Readings in NLP

9. Conclusion (1)

Current Research in Edinburgh
Future Directions of Research

References:

Readings in Natural Language Processing
Barbara J. Grosz, Karen Sparck Jones, Bonnie Lynn Webber (eds.)
Morgan Kaufmann, 1986

A Computational Framework for Lexical Description M.Ritchie
G.D.Ritchie, S.G.Pulman, A.W.Black, G.J.Russell
DAI Research Paper no. 293, 1986

An Introduction to Unification-Based Grammatical Formalisms } – Not available
S.M.Shieber
Center for the Study of Language and Information, 1986

Language as a Cognitive Process
Terry Winograd
Addison-Wesley, 1983

An Informal Introduction to Formal Languages

Mathematically, we can define a language with respect to a vocabulary Σ as some subset of $\Sigma*$ i.e. the (infinite) set of all strings obtained by concatenating elements of the vocabulary.

Do not be confused by the terminology often used in texts on mathematical linguistics, which calls the vocabulary 'an alphabet' and each sentence in the language under consideration 'a word'. We will always refer to the members of the vocabulary as **words** or possibly **morphemes.**

We will look at two alternative ways of defining languages, one by means of **grammars** which generate the strings, i.e. sentences, of the languages concerned, and the other by means of **automata** which are machines that recognise sentences of the languages.

In particular, we will introduce four types of grammar, and four corresponding automata, that form a heirarchy, in which the languages that are describable by a type n device can be shown to properly include those definable by a type n + 1 device. That is, the lower the number, the more powerful the device. This is called the Chomsky heirarchy.

The import of this for natural language processing is as follows. We can tentatively identify certain constructions in in natural languages that we know require a certain type of device for their description - a grammar of a higher type will just not do. We can argue about the data - whether a particular construction is indeed of a certain form - but once we agree on the linguistics, we have mathematical certainty about the sort of formal device required.

1. Grammars

A grammar is defined as a 4-tuple $G = (N, \Sigma, S, P)$ where

(1) N is a finite set of **nonterminal symbols** (corresponding to syntactic categories).

(2) Σ is a finite set of **terminal symbols** disjoint from N (corresponding to words or perhaps lexical categories).

(3) S is a distinguished symbol, a member of N, called the start symbol

(4) P is a finite subset of

$$(N \cup \Sigma)* \, N \, (N \cup \Sigma)* \times (N \cup \Sigma)*$$

i.e. a set of pairs of strings of elements from the two sets of symbols, the first of which must include one non-terminal element. These are the familiar rewrite rules of linguistics, so we write an element $(\alpha,\beta) \in P$ as $\alpha \rightarrow \beta$.

The language generated by a grammar L(G) is defined as follows:

(1) S is a sentential form.

(2) if $\alpha\beta\gamma$ is a sentential form, and $\beta -> \delta \in P$, then $\alpha\delta\gamma$ is a sentential form.

(3) a sentential form containing no elements of N is a sentence.

(4) The language L(G) is the set of sentences generated by G

The position of a grammar on the Chomsky heirarchy can be determined by examination of the rules of P. That is, the more constrained the rules are, the higher in the heirarchy is the grammar.

2. Automata

An automaton can be pictured as having three components:

(1) An input tape, divided into squares, each containing a symbol from an input alphabet. There is a tape head, positioned on one of the squares, initially the leftmost.

(2) An auxiliary memory, whose behaviour is characterised by two functions:

- a fetch function, which maps from the set of memory configurations to a set of symbols

- a store function, which maps a memory configuration and a control string to a memory configuration

(3) A finite state control, which mediates between the above two components.

A recogniser operates by making a series of moves. Each move consists of reading the input symbol under the tape head, and probing the auxiliary memory by means of the fetch function. These two items of information, together with the current state of the finite state control, then determine the rest of the move. This consists of shifting the tape head one square left or right, or keeping it stationary; storing information into the memory; and changing the state of the control. as per Turing m/c

Moves are made until the input string has been read (the tape head is at the right hand end), and the control is in one of a designated set of final states. There may also be a condition on the state of the memory. If these requirements are satisfied, then the input string is in the language defined by the recogniser. If not, then it is not.

The position on the Chomsky heirachy to which a particular recogniser corresponds is determined by the type of auxiliary memory that it incorporates. That is, the more restricted are the memory functions, the higher in the heirarchy are the languages that can be recognised.

We can now examine the heirarchy and present the equivalence between the two formal systems for defining languages.

The Chomsky Heirarchy					
Type	Automaton		Grammar		O
	Memory	Name	Rule	Name	
0	Unbounded	Turing Machine	$\alpha \rightarrow \beta$	General Rewrite	?
1	Length $\leqslant$ n	Linear Bounded	$\beta A \gamma \rightarrow \beta \delta \gamma$	Context-sensitive	e^n
2	Stack	Push-down	$A \rightarrow \beta$	Context-free	n^3
3	None	Finite State	$A \rightarrow xB$, $A \rightarrow x$	Right Linear	n

where: n is the length of the input string, A,B $\in$ N, x $\in$ Σ*, $\alpha, \beta, \gamma, \delta \in (N \cup \Sigma)^*$, $\delta \neq \varepsilon$

non-terminals

$\alpha, \beta, \gamma, \delta \in (N \cup \Sigma)*, \delta \neq e$

3. Type 3 devices

You should be very familiar with the type 3 automaton, the **finite state transition network**. The equivalent grammars are those whose rewrite rules have a maximum of one non-terminal symbol, which must be the rightmost symbol, hence the name **right linear grammar**. Formally equivalent are the left linear grammars, in which the single non- terminal must be leftmost.

The languages describable are precisely those that can be expressed as **regular expressions**, which are expressions such as (a b* c) | d which can be glossed as those languages consisting of the single symbol 'd', or 'a' followed by any number of 'b's followed by a 'c'.

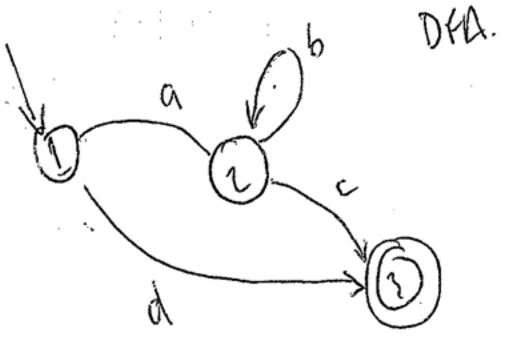
The equivalent right linear grammar would be:

$S \rightarrow d$
$S \rightarrow aB$
$B \rightarrow bB$
$B \rightarrow c$

The equivalent fsa would be:

start state	input symbol	end state
1	d	3
1	a	2
2	b	2
2	c	3

initial state = 1
final state = {3}



4. Type 2 devices

The type 2 grammars are the well-known **context-free grammars**, in which a single non-terminal symbol appears on the left hand side of a rule. The name context-free refers to the fact that the expansion of a non-terminal does not depend on the context in which it appears.

The type 2 automata are the **Push-down automata**, which are finite state automata enriched with a **stack**. In making a transition, a PDA can push a symbol onto the stack, and in determining which transitions are legal, the symbol on the top of the stack may be examined and popped.

A typical context-free language would be $a^n b^n$, that is any number of 'a's followed by the same number of 'b's. A CFG for this language is:

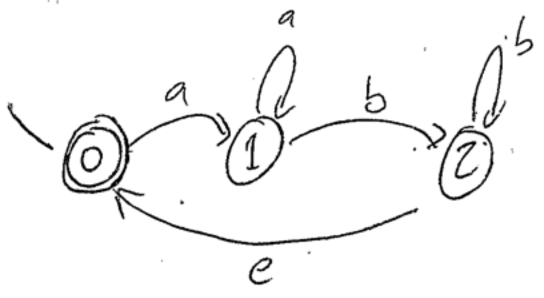
$S \rightarrow aSb$
$S \rightarrow e$

and a PDA:

start state	input symbol	stack before	end state	stack after
0	a	Z	1	aZ
1	a	0...	1	00...
1	b	0...	2	...
2	b	0...	2	...
2	e	Z	0	

0Z

initial state = 0
final states = {0}



5. Type 1 devices

The type 1 grammars are called the **Context-sensitive grammars.** A rule in such a grammar must rewrite at most one non-terminal from the left-hand side, but contextual restrictions may be imposed on this, hence the name. The non-terminal must not rewrite as the empty string 'e'. Notice the effect of this is to ensure that no rewriting can decrease the length of the string, and hence guarantee **decidability.**

The same constraint is in evidence in the definition of the type 1 automaton, the **linear bounded automaton (LBA)**, auxiliary memory must not exceed the length of the input string.

Some typical context-sensitive languages are $a^n b^n c^n$, the language consisting of all strings with a number of 'a's folowed by the same number of 'b's and then the same number of 'c's; and ww, the language consisting of all strings whose first half is the same as their second half.

A grammar for the first of these is:

len(LHS) ≤ len(RHS)

INCORRECT!

S → aSbc
S → abC
bB → bb
bC → bc
cC → cc
CB → CD
CD → ED
ED → EC
EC → BC

6. Type 0 devices

Finally, the type 0 devices are the **General Rewrite Grammar**, in which there are no restrictions on the form of the rules, and the **Turing Machine**, which has an unbounded auxiliary memory. Any language for which there is a recognition procedure can be defined using these devices, but in general, the recognition problem is not decidable. That is, assume we have an arbitrary device of this type, and a string whose membership in the language is to be tested. If the string is in the language, then we will get the answer 'yes' after a finite time. However, if it is not, we may never get an answer.

c.f. "THE HALTING PROBLEM"