Language

Definition 1 (Language)

A language is a subset of the set of all strings over an alphabet Σ.

In the context of NLP, Σ is the vocabulary and a language is the set of all 'legal' sentences in the language.

A legal sentence is one that can be derived from a grammar (see later).

Parsing

Definition 2 (Parsing)

Parsing is the process of identifying relationships between constituents of a legal sentence in a language.

Parsing produces a parse of a sentence which makes explicit the relations between the constituents of the sentence.

If the sentence is not a legal sentence in the language then a parse fails.

For example, these could be dominance relations (in a parse tree), functional relations in a lexical grammar or dependency relations in a dependency parse.

The main reason for a parse is that it allows us to compose the meaning of a sentence by compositional operations on the meanings of the constituents of the sentence.

Types of parsing

There are multiple kinds of parsing. Two main types used in NLP are:

- Constituent structure. A way to chunk words in a sentence often hierarchically. Two major variants are:
    - Phrase structure grammar (PSG) based parses.
    - Lexical grammar based parses - these concentrate much more on the words in a sentence and define relationships between words and word groupings. We will look at one example of a lexical grammar called Combinatory Categorial Grammar (CCG).
- Dependency parsing: Labels pairs of words with binary relations (called dependency relations) chosen from a fixed vocabulary of relations. One set of relations are the universal dependency relations (see http://universaldependencies.org/).

Phrase Structure Grammar: notation

Formally, a grammar is a 4-tuple (N, Σ, R, S) where:

N - finite set of non-terminals or variables. Corresponds to S, NP, VP, etc.
Σ - finite set of terminal symbols. Corresponds to the vocabulary.
R - finite set of rewrite rules of the form $\alpha \to \beta$, where $\alpha, \beta \in (N \cup \Sigma)^{\star}$. Constraints on the types of rules produce different types of grammars.
$S \in N$ - is the start symbol.

Derivation, language generated by grammar

A <u>1-step derivation</u>, written $\gamma \Rightarrow \delta$ is one application of a rule in R to the string γ to produce string δ. A <u>derivation</u> of δ from γ, written $\gamma \overset{\star}{\Rightarrow} \delta$ is chain of 0 or more 1-step derivations. That is: $\gamma \Rightarrow \gamma_1 \Rightarrow \gamma_2 \ldots \Rightarrow \gamma_m = \delta$.

A <u>derivation</u> of a terminal string $w \in \Sigma^\star$, written, $S \overset{\star}{\Rightarrow} w$ is a way for a grammar to derive a terminal string or a terminal sentence. A terminal string or sentence is one which contains only elements from Σ.

The language $L(G)$ generated by grammar G is defined as the set of all terminal sentences derivable from the grammar starting with the start symbol $S \in N$.

$L(G) = \{w \in \Sigma^\star | S \overset{\star}{\Rightarrow} w\}$

Examples of grammars and languages

1. $G = (\{S\}, \{0,1\}, \{S \to 0|1,\ S \to 0S|1S\}, S)$
   $L(G)$ - set of all binary strings.
2. $S \to$ ‘`if`’BE‘`then`’S‘`else`’$S|$‘`if`’BE‘`then`’S
   The above is a rule that generates all conditional statements in a C like language. The strings within single quotes are terminal symbols. BE is a non-terminal that generates boolean expressions.
3. The grammar below is for boolean expressions where VAL is a non-terminal that generates value expressions.
   $S \to VAL$‘`==`’ | ‘`<`’ | ‘`<=`’ | ‘`>`’ | ‘`>=`’ | ‘`<>`’ VAL
   $S \to S$‘ $\wedge$ ’|‘ $\vee$ ’S
   $S \to !S$,
   $S \to$ ‘`true`’ | ‘`false`’

Chomsky hierarchy

In the table below, $\alpha, \beta \in (\Sigma \cup N)^\star$, $\gamma \in (\Sigma \cup N)^+$, $A, B \in N$, $a \in \Sigma$.

Grammar type	**Language type**	**Rule type**
Type-0	Recursively enumerable	$\alpha \to \beta$
Type-1	Context sensitive	$\alpha A \beta \to \alpha \gamma \beta$
Type-2	Context free	$A \to \alpha$
Type-3	Regular	$A \to a,\ A \to aB$

Where do NLs lie in the Chomsky hierarchy?

- Some natural languages have some feature(s) that are not context free. [1] So, NLs are not context free. They appear to be mildly context sensitive. But for the most part they can be considered context free.
- Parsing context sensitive languages is computationally hard so NLs are highly unlikely to be context sensitive. They have some mildly context sensitive features but in practice can be treated as context free since sentence lengths are finite and generally small.
- So, NLP uses algorithms for context free languages for parsing NLs.

[1] Laura Kallmeyer http://user.phil-fak.uni-duesseldorf.de/ kallmeyer/GrammarFormalisms/4nl-cfg.pdf and http://user.phil-fak.uni-duesseldorf.de/ kallmeyer/GrammarFormalisms/4lcfrs-intro.pdf

Normal forms for grammars

Some parsing algorithms require that the CFG be in a particular form. One commonly used form is the Chomsky Normal form or CNF. A CFG is in CNF if all rules satisfy the following (where $A, B, C \in N$ and are distinct from S and $a \in \Sigma$):

$A \rightarrow BC$; $A \rightarrow a$; $S \rightarrow \epsilon$ (only if $\epsilon \in L(G)$)

All CFGs can be converted to CNF by a set of simple transformations[2]. And any CNF grammar is a CFG.

[2] See M Sipser, Intro to Theory of Computation.

CYK - bottom up parsing

The Cocke, Younger, Kasami (CYK) algorithm is a dynamic programming based, bottom-up algorithm. Let $w = w_1, \ldots, w_n$ be the input string or sequence. The CYK alogorithm fills in a matrix T where:

$T_{i,j}$ is the set of non-terminals $A \in N$ such that $A \overset{\star}{\Rightarrow} w_{i+1,\ldots,w_j}$ where $0 \leq i < j \leq n$. If start symbol S is in $T_{0,n}$ then sentence w is legal and derivations can be extracted from T by using back pointers.

The version of the CYK algorithm we discuss needs the grammar to be in Chomsky Normal Form.

Only a triangular part of T is filled up. The algorithm uses the following recurrence to fill-up T.

$$\begin{aligned} T_{i-1,i} &= \{A|A \rightarrow w_i\} \\ T_{i,k} &= \{A|A \rightarrow BC, i < j < k, \quad B \in T_{i,j} \text{ and } C \in T_{j,k}\} \end{aligned}$$

$w = w_1, \ldots, w_n$ has a parse if $S \in T_{0,n}$.

CYK Example

Grammar: $S \rightarrow AB|BC \quad A \rightarrow BA|a \quad B \rightarrow CC|b \quad C \rightarrow AB|a$

String: baaba

0	1	2	3	4	5
	b	a	a	b	a
	$\{B\}_{(0,1)}$	$\{A, C\}_{(1,2)}$	$\{A, C\}_{(2,3)}$	$\{B\}_{(3,4)}$	$\{A, C\}_{(4,5)}$
	$\{S, A\}_{(0,2)}$	$\{B\}_{(1,3)}$	$\{S, C\}_{(2,4)}$	$\{S, A\}_{3,5}$	
	-$_{(0,3)}$	$\{B\}_{(1,4)}$	$\{B\}_{(2,5)}$		
	-$_{(0,4)}$	$\{S, A, C\}_{(1,5)}$			
	$\{S, A, C\}_{(0,5)}$				

The CYK algorithm

Algorithm 0.1: $\mathrm{CYK}(T, w = w_1, \ldots, w_n)$

```
for each $(i, j \in (0 \leq i, j \leq n))$    $T_{i,j} \leftarrow \emptyset$
for $i \leftarrow 1$ to $n$
  do for each rule $A \rightarrow w$
       do if $w == w_i$
            then $T_{i-1,i} \leftarrow T_{i-1,i} \cup A$
for $k \leftarrow 2$ to $n$
  do for $i \leftarrow k - 2$ to $0$
       do for $j \leftarrow i + 1$ to $k - 1$
            do for each rule $A \rightarrow BC$
                 do if $B \in T_{i,j}$ and $C \in T_{j,k}$
                      then $T_{i,k} \leftarrow T_{i,k} \cup A$
if $S \in T_{0,n}$
  then return ( true )
  else return ( false )
```

Earley's algorithm - notation

Earley's algorithm (top-down alg.) can work with a CFG in any form.

$A, B, X, Y \in N$; $\alpha,\ \beta,\ \gamma \in (\Sigma \cup N)^\star$

State: $(A \rightarrow \alpha \bullet \beta, i)$

- $(A \rightarrow \alpha\beta) \in R$
- $\bullet$ indicates the point to which parsing has progressed in the rule. α has been seen and β is expected.
- i is the point in the sentence/string being parsed at which parsing began for this rule.

$\mathcal{S}(k)$: set of states at input position k.

Parser starts with $\mathcal{S}(0)$ consisting of only the top level rule. This can be easily ensured by using a dummy start symbol S_0 and a rule $S_0 \rightarrow S$. So, $\mathcal{S}(0) = \{S_0 \rightarrow \bullet S, 0\}$

3 main operations in Earley's algorithm

Prediction: $\forall s \in \mathcal{S}(k)$ of form $(A \to \alpha \bullet B\beta, j)$ add $(B \to \bullet\gamma, k)$ to $\mathcal{S}(k)$ for every rule in R with B on LHS.

Scanning: If a is the next symbol in the input, for every state in $\mathcal{S}(k)$ of the form $(A \to \alpha \bullet a\beta, j)$ add $(A \to \alpha a \bullet \beta, j)$ to $\mathcal{S}(k+1)$.

Completion: For every state in $\mathcal{S}(k)$ of the form $(A \to \gamma\bullet, j)$ find states in $\mathcal{S}(j)$ of the form $B \to \alpha \bullet A\beta, i$ and add $B \to \alpha A \bullet \beta, i$ to $\mathcal{S}(k)$.

A state is not added to a state set if it is already in the set.

Earley example

Grammar: $\mathtt{S} \to \mathtt{S} + \mathtt{E} | \mathtt{E} \quad \mathtt{E} \to \mathtt{E} \times \mathtt{T} | \mathtt{T} \quad \mathtt{T} \to 1|2|3$

String: $1 + 2 \times 3$

Here is one derivation:

$$\mathtt{S} \Rightarrow \mathtt{S} + \mathtt{E} \Rightarrow \mathtt{E} + \mathtt{E} \Rightarrow \mathtt{E} + \mathtt{E} \times \mathtt{T} \Rightarrow \mathtt{T} + \mathtt{E} \times \mathtt{T} \Rightarrow \mathtt{T} + \mathtt{T} \times \mathtt{T}$$
$$\Rightarrow 1 + \mathtt{T} \times \mathtt{T} \Rightarrow 1 + 2 \times \mathtt{T} \Rightarrow 1 + 2 \times 3.$$

Earley example

$S_0 \rightarrow S$

$S \rightarrow S + E | E$

$E \rightarrow E \times T | T$

$T \rightarrow 1|2|3$

String: $1 + 2 \times 3$

S(0): $\bullet 1 + 2 \times 3$

No	State	Remark
1	$S_0 \rightarrow \bullet S, 0$	start
2	$S \rightarrow \bullet S + E, 0$	Pr from 1
3	$S \rightarrow \bullet E, 0$	Pr from 1
4	$E \rightarrow \bullet E \times T, 0$	Pr from 3
5	$E \rightarrow \bullet T, 0$	Pr from 3
6	$T \rightarrow \bullet 1, 0$ $T \rightarrow \bullet 2, 0$ $T \rightarrow \bullet 3, 0$	Pr from 5

S(1)

S(0)

No	State
1	$S_0 \rightarrow \bullet S, 0$
2	$S \rightarrow \bullet S + E, 0$
3	$S \rightarrow \bullet E, 0$
4	$E \rightarrow \bullet E \times T, 0$
5	$E \rightarrow \bullet T, 0$
6	$T \rightarrow \bullet 1, 0$

S(1): $1 \bullet + 2 \times 3$

No	State	Remark
1	$T \rightarrow 1 \bullet, 0$	Sc S(0)-6
2	$E \rightarrow T \bullet, 0$	Co from 1, S(0)-5
3	$E \rightarrow E \bullet \times T, 0$	Co from 2, S(0-4)
4	$S \rightarrow E \bullet, 0$	Co from 2, S(0)-3
5	$S \rightarrow S \bullet + E, 0$	Co from 4, S(0)-2
6	$S_0 \rightarrow S \bullet, 0$	Co from 4, S(0)-1

S(2)

S(1)

5 $S \rightarrow S \bullet, 0$

S(2): $1 + \bullet 2 \times 3$

No.	State	Remark
1	$S \rightarrow S + \bullet E, 0$	Sc from S(1)-5
2	$E \rightarrow \bullet E \times T, 2$	Pr from 1
3	$E \rightarrow \bullet T, 2$	Pr from 1
4	$T \rightarrow \bullet 2, 0$	Pr from 3

S(3)

S(2):

No.	State
1	$S \rightarrow S + \bullet E, 0$
2	$E \rightarrow \bullet E \times T, 2$
3	$E \rightarrow \bullet T, 2$
4	$T \rightarrow \bullet 2, 0$
S(0)-2	$S \rightarrow \bullet S + E, 0$
S(0)-1	$S_o \rightarrow \bullet S, 0$

S(3): $1 + 2 \bullet \times 3$

No.	State	Remark
1	$T \rightarrow 2\bullet, 2$	Sc from S(2)-4
2	$E \rightarrow T\bullet, 2$	Co from 1, S(2)-3
3	$E \rightarrow E \bullet \times T, 2$	Co from 2, S(2)-2
4	$S \rightarrow S + E\bullet, 0$	Co from 2, S(2)-1
5	$S \rightarrow S \bullet + E, 0$	Co from 4, S(0)-2
6	$S_0 \rightarrow S\bullet, 0$	Co from 4, S(0)-1

Termination

- Algorithm terminates when $S_0 \rightarrow S\bullet, 0$ in $S(\mathit{length}(\mathit{sent}))$.
- At this point point the $\bullet$ is at the extreme right of the sentence to be parsed.

Earley's algorithm

Algorithm 0.2: EarleyAlgorithm(*sent*, *gr*)

```
$\mathcal{S}(0) \leftarrow \{S_0 \rightarrow \bullet S, 0\}$;
for $i \leftarrow 0$ to $length(sent)$
  do for each $s \in \mathcal{S}(i)$
       do if ($isIncomplete(s)$)
            then if ($nextSymbol(s) \in N$)
                   then $predictor(s, i, gr)$;
                   else $scanner(s, i, gr)$;
            else $completer(s, i, gr)$;
return ($all\ \mathcal{S}()$)
```

State $(S_0 \rightarrow S\bullet, 0)$ in $\mathcal{S}(length(sent))$ represents the completed parse. The parse tree can be generated by putting in back-pointers in the algorithm with a little extra code.