Computational Linguistics: Syntax I

Raffaella Bernardi

KRDB, Free University of Bozen-Bolzano

Via della Mostra, Room: 1.06, e-mail: bernardi@inf.unibz.it

Contents

1. Reminder

Main issues of last lecture:

- Different levels of Natural Language

  1. Phonology
  2. Morphology
  3. Syntax
  4. Semantics
  5. Discourse
  6. Pragmatics

- Linguistically motivated computational models. For any topic:

  1. Linguistic Theory
  2. Formal Analysis
  3. Implementation

- Linguistic Theories

  1. Morphology: Stems vs. Affixes; Inflectional and derivational forms.
  2. PoS: classes (categories) of words

- Natural Language as Formal Language

  1. Morphology can be formalized by means of Regular Languages and as such modeled by FSA.
  2. FSA cannot handle "Dependency" ($a^n b^n$, cannot count)

- Implementation

2. Syntax

- **Syntax**: "setting out things together", in our case things are words. The main question addressed here is "*How do words compose together to form a grammatical sentence (s) (or fragments of it)?*"

- **Constituents**: Groups of categories may form a single *unit or phrase* called constituent. The main phrases are noun phrases (*np*), verb phrases (*vp*), prepositional phrases (*pp*). Noun phrases for instance are: "she"; "Michael"; "Rajeev Goré"; "the house"; "a young two-year child".

  Tests like substitution help decide whether words form constituents.

  Another possible test is coordination.

3. Dependency

Dependency: Categories are interdependent, for example

Ryanair services [Pescara]$_{np}$	Ryanair flies [to Pescara]$_{pp}$
*Ryanair services [to Pescara]$_{pp}$	*Ryanair flies [Pescara]$_{np}$

the verbs services and flies determine which category can/must be juxtaposed. If their constraints are not satisfied the structure is ungrammatical.

4. Long-distance Dependencies

Interdependent constituents need not be juxtaposed, but may form long-distance dependencies, manifested by gaps

- What cities does Ryanair service [. . .]?

The constituent what cities depends on the verb service, but is at the front of the sentence rather than at the object position.

Such distance can be large,

- Which flight do you want me to book [. . .]?
- Which flight do you want me to have the travel agent book [. . .]?
- Which flight do you want me to have the travel agent nearby my office book [. . .]?

4.1. Relative Pronouns

Relative Pronoun (eg. who, which): they function as e.g. the subject or object of the verb embedded in the relative clause (*rc*),

- [[the [student [who [. . .] knows Sara]$_{rc}$]$_{n}$]$_{np}$ [left]$_{v}$]$_{s}$.
- [[the [book [which Sara wrote [. . .]]$_{rc}$]$_{n}$]$_{np}$ [is interesting]$_{v}$]$_{s}$.

Can you think of another relative pronoun?

4.2. Coordination

Coordination: Expressions of the same syntactic category can be coordinated via “and”, “or”, “but” to form more complex phrases of the same category. For instance, a coordinated verb phrase can consist of two other verb phrases separated by a conjunction:

- There are no flights [[leaving Denver]$_{vp}$ and [arriving in San Francisco]$_{vp}$]$_{vp}$

The conjuncted expressions belong to traditional constituent classes, *vp*. However, we could also have

- I [[[want to try to write [. . .]] and [hope to see produced [. . .]]] [the movie]$_{np}$]$_{vp}$”

Again, the interdependent constituents are disconnected from each other.

Long-distance dependencies are **challenging phenomena** for formal approaches to natural language analysis. We will study them in the second part of the course (May).

5. Sentence Structures: English

The structure of a sentences can be represented in several ways, the most common are the following notations: (i) brackets or (ii) trees. For instance, “John ate the cat” is a sentence (s) consisting of noun phrase (np) and a verb phrase (vp). The noun phrase is composed of a verb (v) “ate” and an np, which consists of an article (art) “the” and a common noun (n) “cat”.

$$[\text{John}_{np}\ [\text{ate}_{v}\ [\text{the}_{art}\ \text{cat}_{n}]_{np}]_{vp}]_{s}$$

Give the tree representation of this structure.

5.1. Exercises

Now represent in the format you prefer the sentences below:

I like a red shirt

I will leave Boston in the morning.

John saw the man with the telescope.

John thinks someone left.

6. Formal Approaches

To examine how the syntax of a sentence can be computed, you must consider two things:

1. **The grammar**: A formal specification of the structures allowable in the language. [Data structures]
2. **The parsing technique**: The method of analyzing a sentence to determine its structure according to the grammar. [Algorithm]

1. Which Grammar do we need to analyse NLs?
2. Which Formal Language can represent NL?

7. Syntax Recognizer

In lecture 1, we have used FSA to recognize/generate natural language morphology, and similarly FSA can be used to concatenate words, i.e. at the syntactic level.

We have said that FSA recognize/generate "Regular Languages". But it has been shown that at the **syntactic level NLs are not regular**.

7.1. NLs are not RL: Example I

1. The cat died.
2. The cat the dog chased died.
3. The cat the dog the rat bit chased died.
4. ...

Let, determiner+noun be in the set $A : \{$ the cat, the dog, $\ldots\}$, and the transitive verbs in $B : \{$ chased, bit, $\ldots\}$. Thus the strings illustrated above are all of the form:

$x^n y^{n-1}$ died, where $x \in A$ and $y \in B$, which can be proved to be not a RL.

7.2. NLs are not RL: Example II

Another evidence was provided by Chomsky in 1956. Let $S_1, S_2, \ldots S_n$ be declarative sentences, the following syntactic structures are grammatical English sentences:

- If S_1, then S_2
- Either S_3, or S_4
- The man who said S_5 is arriving today

In each case there is a lexical dependency between one part of each structure and another. “If” must be followed by “then” “either” must be followed by “or”.

Moreover, these sentences can be embedded in English one in another.

> **If either** the man who said S_5 is arriving today **or** the man who said S_5 is arriving tomorrow, **then** the man who said S_6 is arriving the day after.
> Let

if	$\rightarrow a$
then	$\rightarrow a$
either	$\rightarrow b$
or	$\rightarrow b$
other words	$\rightarrow \epsilon$

The sentence above would be represented as *abba*.

This structure of nested dependencies can be represented more generally by a language like xx^R with $x \in \{a, b\}^*$ and ^{R} denoting the reversal of the string x. (Eg. *abbabbabba*) We can prove via the Pumping Lemma that this language is not in a regular language.

Again, this is an example of open and closed balanced parentheses (or **nested dependencies**) that are not in RL.

8. FSA for syntactic analysis

Finite state methods have been applied to syntactic analysis too. Although they are not expressive enough if a full syntactic analysis is required, there are many applications where a partial syntactic analysis of the input is sufficient.

Such partial analyses can be constructed with cascades of finite state automata (or rather **transducers**) where one machine is applied to the output of another.

Anyway, in order to deal with syntactic analysis of natural language we need **a more powerful device than FSA** (and of their corresponding formal grammars, namely regular (or right linear) grammar (RG).)

9. Formal Grammar: Terminology

Formal Grammars are string **re-write systems**. The re-write rules say that a certain sequence of symbols may be substituted by another sequence of symbols. These symbols are divided into two classes:

- **terminal**: symbols that will appear in the string of the language generated by the grammar.
- **non-terminal**: symbols that will be used only in the re-write process.

Given a string, we want to knows whether it belongs to the (Natural) Language.

10. Context Free Grammars

Formal Grammar **more powerful** than Regular Grammars are Context Free Grammars (CFG).

These grammars are called **context free** because all rules contain only one symbol on the left hand side — and wherever we see that symbol while doing a derivation, we are free to replace it with the stuff on the right hand side. That is, the 'context' in which a symbol on the left hand side of a rule occurs is unimportant — we can always use the rule to make the rewrite while doing a derivation.

There are **more expressive kinds of grammars**, with more than one symbol on the left hand side of the rewrite arrow, in which the symbols to the right and left have to be taken into account before making a rewrite. Such grammars are linguistically important, and we will study them in May.

A language is called context free if it is generated by some context free grammar.

Well known CFG are **Phrase Structure Grammars** (PSG) also known as Context Free Phrase Structure Grammars and they are based on **rewrite rules**. They can be used for both understanding and generating sentences.

11. CFG: Formal Language

Let's start by using simple grammars that generate formal languages. E.g., take the grammar below.

Rules

Rule 1	$S \rightarrow A\ B$	Rule 2	$S \rightarrow A\ S\ B$
Rule 3	$A \rightarrow a$	Rule 4	$B \rightarrow b$

the above grammar let us rewrite 'S' to 'aabb'. Try it your self!

```
S
ASB Rule 2
aSB Rule 3
aSb Rule 4
aABb Rule 1
aaBb Rule 3
aabb Rule 4
```

Such a sequence is called a **derivation** of the symbols in the last row, in this case, i.e. a derivation of the string 'aabb' ($S \Rightarrow^* aabb$).

11.1. CFG: More derivations

Note that there may be many derivations of the same string. For example,

```
S
ASB Rule 2
ASb Rule 4
aSb Rule 3
aABb Rule 1
aAbb Rule 4
aabb Rule 3
```

is another derivation of ‘aabb’.

12. FG for Natural Languages

Now we will move to see how CFG have been applied to natural language. To this end, it is convenient to distinguish rules from non-terminal to terminal symbols which define the lexical entries (or lexicon).

- Terminal: The terminal symbols are **words** (e.g. sara, dress . . .).
- Non-terminal: The non-terminal symbols are syntactic **categories** (CAT) (e.g. np, vp, . . .).
- Start symbol: The start symbol is the s and stands for **sentence**.

The production rules are divided into:

- Lexicon: e.g. $np \longrightarrow$ sara. They form the set LEX
- Grammatical Rules: They are of the type $s \longrightarrow np\ vp$.

Alternative notation The lexicon and the derivation can also be written as below:

$np \longrightarrow$ sara is also written as $\langle \text{sara}, np\rangle$

A derivation of a sequence of words $w_1, \ldots w_n$ from the start symbol will be represented as,

$\langle w_1 \ldots w_n, s\rangle$

13. PSG: English Toy Fragment

We consider a small fragment of English defined by the following grammar $G = \langle \mathsf{LEX}, \mathsf{Rules} \rangle$, with vocabulary (or alphabet) V and categories CAT.

- $\mathsf{LEX} = V \times \mathsf{CAT}$
  - $V = \{\text{Sara}, \text{dress}, \text{wears}, \text{the}, \text{new}\}$,
  - $\mathsf{CAT} = \{det, n, np, s, v, vp, adj\}$,
  - $\mathsf{LEX} = \{\langle \text{Sara}, np \rangle, \langle \text{the}, det \rangle, \langle \text{dress}, n \rangle, \langle \text{new}, adj \rangle, \langle \text{wears}, v \rangle\}$
- $\mathsf{Rules} = \{s \longrightarrow np\ vp,\ np \longrightarrow det\ n,\ vp \longrightarrow v\ np, n \longrightarrow adj\ n\}$

Among the elements of the language recognized by the grammar, $L(G)$, are

- $\langle \text{the}, det \rangle$ because this is in the lexicon, and
- $\langle \text{Sara wears the new dress}, s \rangle$ which is in the language by repeated applications of rules.

14. English Toy Fragment: Strings

$\langle \text{Sara wears the new dress}, s\rangle$ is in the language. Try to prove it your self.

(1) $\langle \text{new dress}, n\rangle \in L(G)$ because
$n \rightarrow adj\ n \in \textsf{Rules}$,
$\langle \text{new}, adj\rangle \in L(G)$ ($\textsf{LEX}$), and
$\langle \text{dress}, n\rangle \in L(G)$ ($\textsf{LEX}$)

(2) $\langle \text{the new dress}, np\rangle \in L(G)$ because
$np \rightarrow det\ n \in \textsf{Rules}$,
$\langle \text{the}, det\rangle \in L(G)$ ($\textsf{LEX}$), and
$\langle \text{new dress}, n\rangle \in L(G)$ (1)

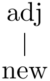
(3) $\langle \text{wears the new dress}, vp\rangle \in L(G)$ because
$vp \rightarrow v\ np \in \textsf{Rules}$,
$\langle \text{wears}, v\rangle \in L(G)$ ($\textsf{LEX}$), and
$\langle \text{the new dress}, np\rangle \in L(G)$ (2)

(4) $\langle \text{Sara wears the new dress}, s\rangle \in L(G)$ because
$s \rightarrow np\ vp \in \textsf{Rules}$,
$\langle \text{Sara}, np\rangle \in L(G)$ ($\textsf{LEX}$), and
$\langle \text{wears the new dress}, vp\rangle \in L(G)$ (3)

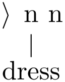
Now try to build the structure of the parsed string.

15. English Toy Fragment: Phrase Structure Trees

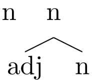
⟨ new, adj ⟩



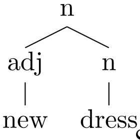
⟨ dress, n ⟩

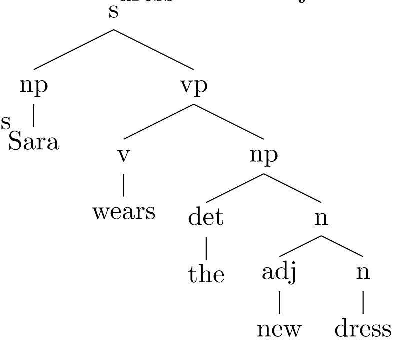


n → adj n



$\langle\langle \text{new}, adj\rangle, \langle \text{dress}, n\rangle, n\rangle$





16. Extending our grammar

Try to extend your grammar so to deal with the sentence you have analyzed before and which are repeated below.

She likes a red shirt.

I will leave Boston in the morning.

John saw the man with the telescope.

John gave Mary a red shirt.

17. Recursion

Which rule do we need to generate the phrases below?

1. the flight to Boston
2. the flight to Boston from Miami
3. the flight to Boston from Miami in February
4. the flight to Boston from Miami in February on Friday

```
np --> np pp
```

I.e. a recursive rule, it contains the same category on the left and the right side.

18. Summing up (I)

We have seen that

- There is a close correspondence between **parse trees** and **derivations**: every derivation corresponds to a parse tree, and every parse tree corresponds to (maybe many) derivations.
- PSG, besides deciding whether a string belongs to a given language, deals with phrase structures represented as trees.
- An important difference between strings and phrase structures is that whereas string concatenation is assumed to be associative, trees are bracketed structures.
- Thus trees preserve aspects of the compositional (constituent) structure or derivation which is lost in the string representations.

19. Summing up (II)

- The **language generated** by a grammar consists of all the strings that the grammar classifies as grammatical.
- A CFG **recognizer** is a program that correctly tells us whether or not a string belongs to the language generated by a PSG.
- A CFG **parser** is a program that correctly decides whether a string belongs to the language generated by a CFG and also tells us what its structure is.
- A **Context Free Language** is a language that can be generated by a CFG.

20. Generative Power

We have seen how to use a formal grammar to recognize natural language strings/phrases.

Every (formal) grammar generates a unique language. However, one language can be generated by several different (formal) grammars.

Formal grammars differ with respect to their **generative power**:

One grammar is of a greater generative power than another if it can recognize a language that the other cannot recognize.

Two grammars are said to be

- **weakly** equivalent if they generate the same string language.
- **strongly** equivalent if they generate both the same string language and the same tree language.

Remark Some of the slides (on Chomsky Hierarchy, etc.) are from Gerhard Jaeger course given at ESSLLI '04.

21. Hierarchy of Grammars and Languages

Based on this observation it's possible to construct a hierarchy of grammars, where the set of languages describable by grammars of grater power subsumes the set of language describable by grammars of less power. The most commonly used hierarchy is the **Chomsky Hierarchy of Languages** introduced in 1959.

Hence, the two questions to ask are:

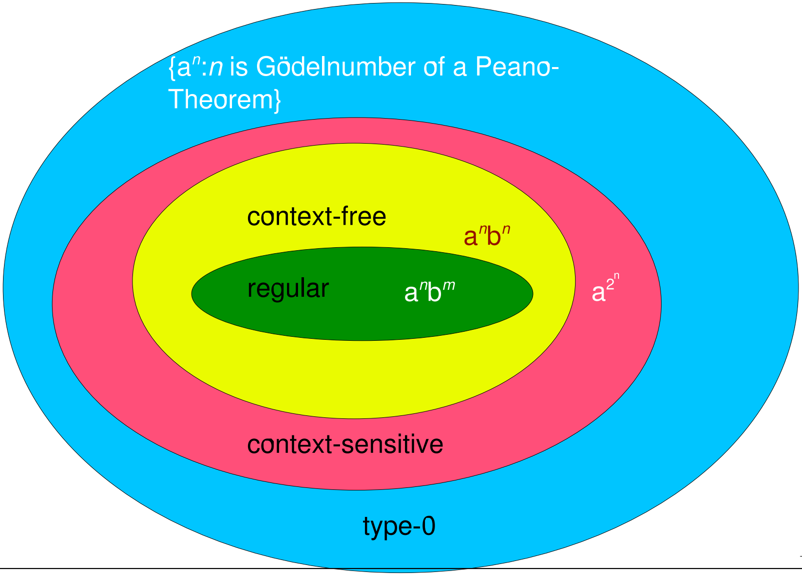
- Where does Natural Language fit in the Hierarchy?
- Which is the generative power of the different Formal Grammars?

If we know where NL fit, we would know

- which formal language can represent NL;
- which rules to use in writing formal grammars for NL.

22. Chomsky Hierarchy of Languages

The Chomsky Hierarchy



23. Dissenting Views

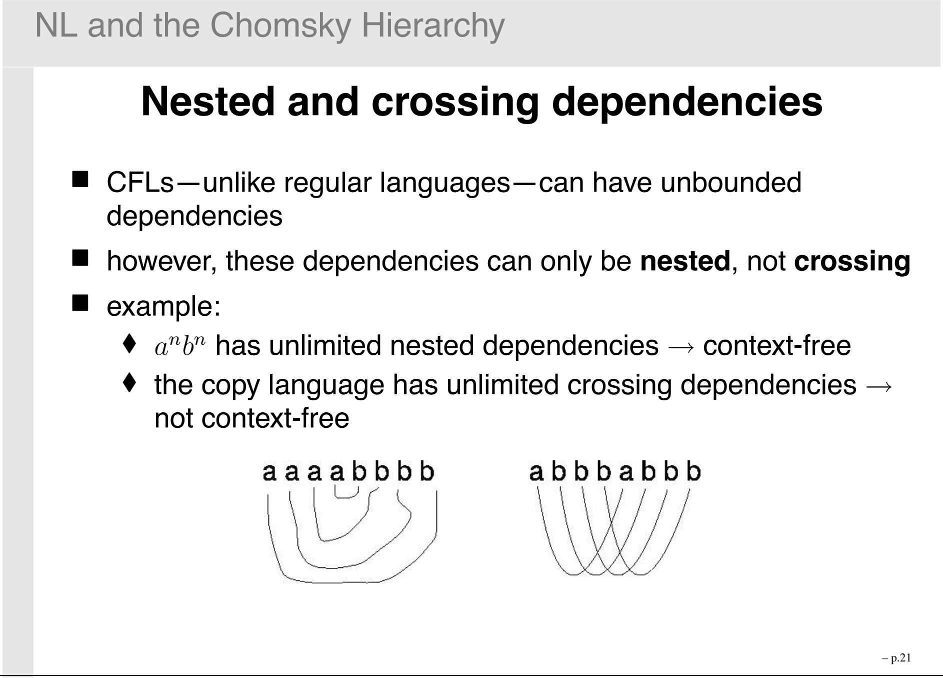
Claim: NL are not RL.

- all arguments to this effect use center-embedding
- humans are extremely bad at processing center-embedding
- notion of competence that ignores this is dubious
- natural languages are regular after all.

23.1. Are NL Context Free (CF)?

History of the problem:

1. Chomsky 1957: conjecture that natural languages are not CF
2. sixties, seventies: many attempts to prove this conjecture
3. Pullum and Gazdar 1982:
    - all these attempts have failed
    - for all we know, natural languages (conceived as string sets) might be context-free
4. Huybregts 1984, Shieber 1985: proof that Swiss German is not context-free
5. Culy 1985: proof that Bambara is not context-free



NL and the Chomsky Hierarchy

Nested and crossing dependencies

- CFLs—unlike regular languages—can have unbounded dependencies
- however, these dependencies can only be **nested**, not **crossing**
- example:
  - $a^n b^n$ has unlimited nested dependencies → context-free
  - the copy language has unlimited crossing dependencies → not context-free

23.3. English & Copy Language

Bar-Hillel and Shamir (1960): "English contains copy-language. Hence it cannot be context-free". E.g. Consider the sentence

> John, Mary, David, ... are a widower, a widow, a widower, ..., respectively.

Claim the sentence is only grammatical under the condition that if the nth name is male (female) then the nth phrase after the copula is a widower (a widow).

Counterargument :

- crossing dependencies triggered by respectively are semantic rather than syntactic.

compare above example to

> (Here are John, Mary and David.) They are a widower, a widow and a widower, respectively.

23.4. Cross-serial dependencies in Dutch

Huybregt (1976):

- Dutch has copy-language like structures
- thus Dutch is not context-free

(1) dat Jan Marie Pieter Arabisch laat zien schrijven
THAT JAN MARIE PIETER ARABIC LET SEE WRITE
tr. "that Jan let Marie see Pieter write Arabic"

Counterargument

- crossing dependencies only concern argument linking, i.e. semantics
- Dutch has no case distinctions
- as far as plain string are concerned, the relevant fragment of Dutch has the structure

$$NP^nV^n$$

which is context-free

23.5. Cross-serial dependencies Swiss German

Today many theorists believe natural languages are not context-free. (Huybregts 1984, Shieber 1985).

Evidences are given by **cross-serial dependencies** found in Swiss German where verbs and their argument can be orderded cross-serially.

A sentence can have a string of

dative nouns followed by a string of accusative nouns, followed by a string of dative-taking verbs, followed by a string of accusative-taking verbs. E.g.

mer em Hans es huus halfed aastriiche
we Hans/Dat the house/Acc helped paint.

tr. we helped Hans paint the house.

The number of verbs requaring dative objects must equal the number of dative NPs and similarly for accusatives, and this number can be arbitrary. Hence, the language representing this phenomenon is $wa^nb^mxc^nd^my$ which is not Context Free (CF).

However, notice that those construction types used to prove that NLs is not CF appear to be hard to understand for humans too.

24. Mildly Context Free Grammars

However, how large NL are continues to be a less simple matter. There are two main non-compatible views:

1. Natural Language forms a class of languages that **includes the CF** family, but is larger than it.
2. Natural Language occupies a position eccentric with respect to that hierarchy, in such a way that it does not contain any whole family in the hierarchy but is **spread along all of them**

The first view gave rise to a new family of languages which is of clear linguistic interest, Mildly Context-sensitive Languages.

25. Mildly Context-sensitive Languages (MSC)

A concept motivated by the intention of characterizing a narrow class of formal grammars which are **only slightly more powerful than CFGs**, and which nevertheless allow for descriptions of natural languages in a linguistically significant way (Joshi 1985).

According to Joshi (1985, p. 225) a mildly context-sensitive language, L, has to fulfill three criteria, to be understood as a rough characterization. Somewhat paraphrased, these are:

1. the parsing problem for L is solvable in **polynomial time** (see later),
2. L has the constant **growth property** (i.e. the distribution of string lengths should be linear rather than supralinear.), and
3. there is a finite **upper bound** for L limiting the number of different instantiations of factorized **cross-serial dependencies** occurring in a sentence of L.

26. Where do the different Formal Grammars stand?

The interest in the frameworks is tied to their generative power, . . . as well as their **destiny**.

Chomsky's formal language theory made it possible to ask for the generative strength of a grammar.

After the discovery of languages which require cross-serial dependencies, grammars that were proved to be Context Free **lost their appeal**. Since CFGs were shown to be inadequate to model those natural languages.

27. Complexity Issue

For any computational problem we face (hence for parsing a NL sentence too), we are interested in **algorithms** (step-by-step procedures) that can be used to solve the problem (hence we are interested in the parsers).

For these algorithms, we are interested in how **efficient** the algorithm is in terms of its run time or its use of memory (or other system resources such as database accesses).

In its broadest sense, the notion of efficiency involves all the various computing resources needed for executing an algorithm. However, by the most efficient algorithm one normally means the **fastest**. Time requirements are often a dominant factor determining whether or not a particular algorithm is efficient enough to be useful in practice.

Time complexity is determined from the corresponding execution time and input length.

27.1. Input length

The time requirements of an algorithm are conveniently expressed in terms of a single variable, the “size” of a problem instance, which is intended to reflect the **amount of input** data needed to describe the instance.

The time complexity function for an algorithm expresses its time requirements by giving, for each possible input length, the **largest amount of time needed by the algorithm to solve a problem instance of that size**.

We are also interested in how the algorithms fare as their **input load gets higher**; if a grammar intended for a fragment of English is extended to full texts, what impact will this have on the run time of the parser?

27.2. Complexity of a Problem

When we need to solve a problem, we are interested in the most efficient algorithm that solves it.

The complexity of a problem is the complexity of such an algorithm.

Classes We distinguish between

- polynomial time problems (PTIME)
- problems believed to be exponential time (e.g. NP, PSPACE)
- problems known to be exponential time or more difficult (e.g. EXPTIME)

See Diego Calvanese's course: Theory of Computing.

27.3. Complexity w.r.t. Chomsky Hierarchy

We are interested in the problem of determining whether a string is in the language generated/recognized by a grammar of a certain type.

- For Context Free Language the problem is polynomial.
- the same holds for Mildly CFL.
- whereas, for Context Sensitive Languages the problem is PSPACE-complete

If NLs were CSL, this would a bad news for CL!

28. Conclusions

Next time we will look at some problems of CFG, and at Formal Grammars tailored for analysing Natural Language.

- ► If you have not given me the filled in form, please do it.

We will start speaking about the projects on the 9th of April.

29. Formal Grammars: Definition

A Formal Grammar (FG) is a formalism to give a finite representation of a Language.

A Grammar, G, is a tuple: $G = (V_T, V_N, S, P)$, such that:

- V_T is the finite set of Terminal Symbols.
- V_N is the finite set of Non-Terminal Symbols.
- Terminal and Non-Terminal symbols give rise to the alphabet: $V = V_T \cup V_N$.
- Terminal and Non-Terminal symbols are disjoint sets: $V_T \cap V_N = \{\}$.
- S is the start symbol (Scope) of the Language, and $S \in V_N$.
- P is the finite set of Productions, $P = \{\alpha \rightarrow \beta \mid \alpha \in V^+ \wedge \beta \in V^*\}$.

29.1. Derivations

To characterize a Language starting from a Grammar we need to introduce the notion of Derivation.

- The notion of Derivation uses Productions to generate a string starting from the Start symbol S.
- Direct Derivation (in symbols $\Rightarrow$)). If $\alpha \rightarrow \beta \in P$ and $\gamma, \delta \in V^*$, then $\gamma\alpha\delta \Rightarrow \gamma\beta\delta$).
- Derivation (in symbols $\Rightarrow^*$)). If $\alpha_1 \Rightarrow \alpha_2$, $\alpha_2 \Rightarrow \alpha_3$, $\ldots, \alpha_{n-1} \Rightarrow \alpha_n$, then $\alpha_1 \Rightarrow^* \alpha_n$.

29.2. Formal Languages and FG

A string belongs to a Language if and only if:

1. The string is made only of Terminal Symbols;
2. The string can be Derived from the Start Symbol, S, of the Language.

Generative Definition of a Language We say that a Language L is **generated** by the Grammar G, in symbols $L(G)$, if:

$$L(G) = \{w \in V_T^* \mid S \Rightarrow^* w\}.$$

29.3. FG and Regular Languages

We have said that the languages generated/recognized by a FSA are called “Regular Languages”. The formal grammars that generate/recognize these languages are known as “Regular Grammar” (RG) or Right Linear Grammars. (or Left Linear Grammar).

Regular Grammars have rules of the form:

- $A \longrightarrow xB$
- $A \longrightarrow x$

where A and B are non-terminal symbols and x is any string of terminals (possibly empty). Moreover, a rule of the form: $S \longrightarrow \epsilon$ is allowed if S does not appear on the right side of any rule.

29.4. FSA and RG

The association between FSA and RG is straight:

RG	FSA
$A \rightarrow xB$	from state A to state B reading x
$A \rightarrow x$	from state A reading x to a designed final state.
start symbol	initial state.

As in FSA, the string already generated/recognized by the grammar has no influence on the strings to be read in the future (no memory!).

See Artale's Course "Compiler" for more details.

29.5. CFG: Language Generated

The above grammar generates the language $a^n b^n - \epsilon$ (the language consisting of all strings consisting of a block of a's followed by a block of b's of equal length, except the empty string).

If we added the rule $S \rightarrow \epsilon$ to this grammar we would generate the language $a^n b^n$. Therefore, these two languages **are context free**.

On the other hand, $a^n b^n c^n$ **is not**. That is, no matter how hard you try to find CFG rules that generate this language, you won't succeed. No CFG can do the job. The same holds for, e.g. $a^n b^m c^n d^m$.

Again, there are formal ways to prove whether a language is or is not context free.