Computational Linguistics
A.Y. 2018/19 - C. Chesi

Lecture 9-10

SYNTACTIC PARSING

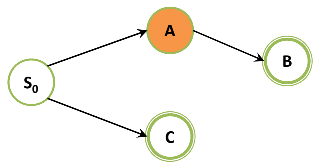
Index

References

- **Essential references** (http://www.ciscl.unisi.it/master/materials.htm)
  - Jurafsky, D. & Martin, J. H. (2009) *Speech and Language Processing. Prentice-Hall*. (2nd edition) http://www.cs.colorado.edu/~martin/slp.html Chapter 13, 15
- **Extended references**
  - Barton G.E., Berwick R. & Ristad E.S. 1987. *Computational Complexity and Natural Language*. MIT Press Shank
  - Hale, J. T. (2011). What a rational parser would do. *Cognitive Science*, *35*(3), 399-443.
  - Van de Koot H. *The Computational Complexity of natural language recognition*. Ms. University College London

Why having a computational model

- Predict possible dysfunctions

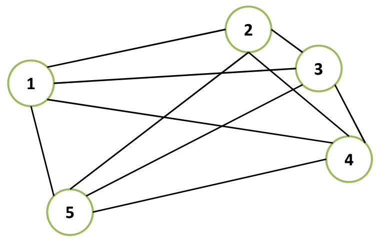


- Calculate the complexity of certain processes…

What's complexity

- **Sorting problem:** order the following 5 numbers

- **travelling salesman problem :** find the shortest path connecting 5 cities

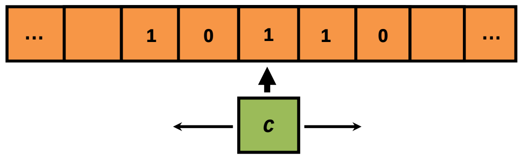


What's computable

- (informally speaking) a **computation** is a relation between an **input** and an **output**. This relation can be defined by various algorithms: a **series of computational states** and **transitions** among them until the final state is reached. A computation attempts at reaching the final state through **legal steps** admitted by the computational model (problem space = set of all possible states the computation can reach).
- Turing-Church thesis (simplified)
  every computation realized by a physical device can be realized by means of an algorithm; if the physical device completes the computation in *n* steps, the algorithm will take *m* steps, **with *m* differing from *n* by, at worst, a polynomial**.
- Some algorithm might take **too much time to find a solution** (e.g. years or even centuries)
- Other algorithms can **not even terminate!**

Turing Machine

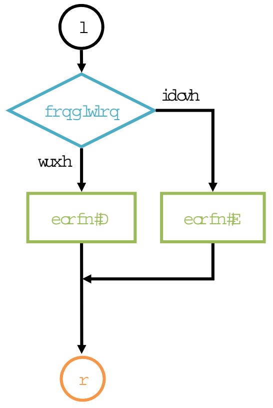
- Infinite tape subdivided in cells
- alphabet ***A*** (e.g. ***A*** ={0, 1})



- cursor ***C*** (that can move right and left, and can read, delete or write a character)
- Finite set of states $\boldsymbol{Q} = (q_0, q_1 \ldots q_n)$
- Finite input ***I*** constituted by a sequence of characters of ***A***
- Finite set of states ***S*** described as 5-tuples $<q_i a b v q_j>$ such that q_i, $q_j \in \boldsymbol{Q}$; $a,b \in \boldsymbol{A}$; $\boldsymbol{v}$ = {right, left}

Flow charts

- Oriented graph**:**
  - An **input** (*i*)
  - One or more exit (*o*)
  - Finite set of instructions blocks such that any instruction is in the form X = Y, X = X+1, X = X-1
  - Finite set of special blocks, named conditions, of the (Boolean) form X = Y?
  - A finite set of **connectors** that links the blocks, such that from every block just one arrow goes outbound and, in case of conditional blocks, 2 arrows go outbound



Modularity

- **Turing Machines** and **flow charts** are equivalent: they express the very same class of function (computable functions)
- Both formalisms guarantee **compositionality** (M1 • M2).
- Hence: “divide et impera” is a programming paradigm that suggests decoupling a problem in smaller sub-problems for which a solution would be easier to be found.

Complexity

- Directly proportional to the resource usage:
  - **Time** (time complexity): number of elementary steps needed
  - **Memory** (space complexity): quantity of information to be stored at each step
- Complexity is **directly proportional** to the **problem dimension** (e.g. ordering 1000 words will be more complex that ordering 10 words);
- Grammar complexity should be related to its **generative power**.

Complexity

- The **problem dimension** is expressed in terms of input length to be processed
- The **order of complexity** should be expressed in terms of input length, e.g.:

  $c \cdot n^2$ (example of polynomial time problem complexity)
  n = input length
  c = costant data (depending on the kind of computation)

  In this case we will say that the complexity order of the problem is n^2 since the c constant will be irrelevant with respect to n growing to the infinite.

  Such complexity order is defined as: $O(n^2)$.

Complexity

- We are interested in the **growing rate of the complexity** expressing the mapping between input and output in terms of input dimension
- For **space and time limited problems** (resource usage surely finite) the complexity calculus is irrelevant
- For ***n growing to the infinite***, as in the case of the grammars we want to study, the **growing rate** is crucial for determining the **tractability of the problem**
- A problem is considered **computable/tractable** if a procedure exists and terminates with an answer (positive or negative) in a **finite amount of time**

Complexity

- A problem with **exponential time complexity** (e.g. ***O(2^N)***) will be hardly computable in a reasonable amount of time. To have an idea, assume a device able to deal with **1 million steps per second**, there the calculation for specific input given specific complexity function:

input length → ↓ function	10	20	50	100
N^2	0,0001 second	0,0004 sec.	0,0025 sec.	0,01 sec.
N^5	0,1 sec.	3,2 sec.	5 min. e 2 sec.	2 hours and 8 min.
2^N	0,001 sec.	1 sec.	35 year e 7 months	400 trillions of centuries
N!	3,6 sec.	about 771 centuries	A number of centuries with 48 digits	A number of centuries with 148 digits
N^N	2 hours and 8 minutes	More than 3 trillions of years	A number of centuries with 75 digits	A number of centuries with 185 digits

Complexity of classic problems

- **3SAT problem** (**satisfiability problem** or **SAT**)
  find a value assignment **for all propositional letters** satisfying the formula below:

  $(a \lor \neg b \lor c) \land (\neg a \lor b \lor \neg c) \land (a \lor b \lor c) \land \ldots$

- In the **worst case**, all possible assignments must be evaluated, that is **2^N** (where **2** are the possible assignment values, True and False, and **N** is the number of propositionals **a**, **b**, **c**…).

- The problem has an **exponential time growth complexity function**, but, once solved, can be readily proved: hard to solve, easy to verify!)

Complexity of classic problems

- **Quantified Boolean Formula (QBF) problem**
  find a value assignment **for all propositional letters** satisfying the formula below :

  $Qx_1,Qx_2 \ldots Qx_n\ F(x_1,x_2 \ldots x_n)$
  (with Q = ⍰ or ⍰)

- The problem is hard to be solved, **as 3SAT**, but also **hard to be verified**: (the 3SAT problem is a special case of QBF where all Q are existential

- The universal quantification requires any assignment of values to be verified.

Complexity of classic problems and reducibility

- If a computer **effectively solve** a problem like 3SAT, it will use an **algorithm that is, at worst, polynomial**.

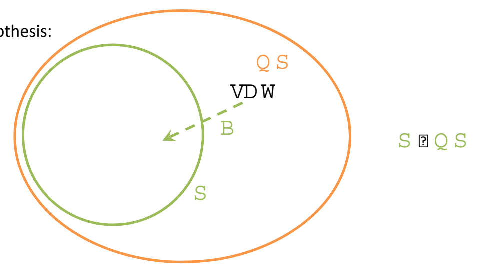
- Because of the problem structure/space, such algorithm should be **necessarily non-deterministic.**

- We call the complexity of this king of problems **NP**:
  Non-deterministic Polynomial time

- Problem with complexity **P** are **deterministic and polynomial**. Problems with an order **P** of complexity are (probably) included in problems with a NP complexity order (no proof of reducibility from NP to P exists… yet).

Complexity of classic problems and reducibility

- Hypothesis:



- Problems like **SAT** are dubbed **NP-hard** (same difficulties, i.e. problem structure/space with respect to NP class problems).

What's Parsing

- Given a Grammar ***G*** and an input ***i*** , parsing ***i*** means applying a function ***p(G, i)*** able to:

1. Accept/Reject ***i***
2. Assign to ***i*** an adequate descriptive structure (e.g. syntactic tree)

Universal Recognition Problem (URP) and reduction

- **Universal Recognition Problem** (**URP**)
  Given a Grammar G (in any grammatical framework) and a string x, x belongs to the language generable by G?

- ***Reduction***
  is there any efficient mapping from this problem to a another well know problem for which we can easily evaluate the complexity?
  YES… **SAT problem**!

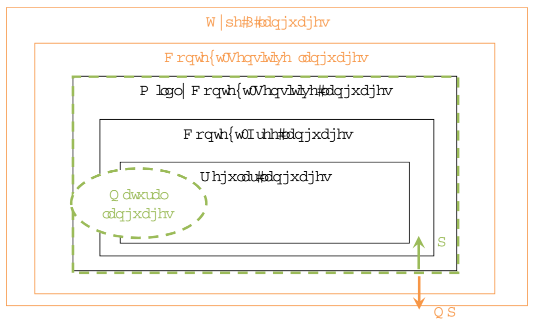
Universal Recognition Problem (URP) and reduction

- **URP** is a generalized parsing problem that can be reduced to **SAT** in its core critical structure

- In a nutshell: a string ***x***, as a proposizional ***a*** in a **SAT formula**, can receive an **ambiguous value assignment** (for instance *"vecchia"* in Italian can both be a *noun* and an *adjectival*, while *a* can be *true* or *false*).
  We then need to keep the assignment coherent in ***x*** (to evaluate the correctness of the final outcome) as in a SAT formula.

- We conclude that URP is at least as complex as SAT, that is, **NP-hard**!

Chomsky’s hierarchy and complexity



Psycholinguistic complexity

- **Complexity = difficulty** in processing a sentence
- **Hypothesis 1:** formal complexity = psycholinguistic complexity
- **Hypothesis 2:** limited processing memory
  - On the one hand, **memory buffer capacity** could be sufficient to store only N structures;
  - On the other, using the memory for storing similar incomplete structures might create **confusion**.

Psycholinguistic complexity

- **Hypothesis 1**
  processing **non context-free structures** causes major difficulties
  (Pullum e Gazdar 1982)
- **Hypothesis 2**
  Limited-size Stack (Yngve 1960)
  linguistic processing uses a stack to store partial analyses. The more partial phrases are stored in the stack, the harder the processing will be.
  - **Syntactic Prediction Locality Theory** (SPLT, Gibson 1998) **total memory load** is proportional to the sum of required an integration + referentiality needs:
    1. DPs required VPs (in SVO languages):
       DP DP DP VP VP VP... Is harder than DP VP
    2. A **pronoun** referring to an already introduced referential entity **is less complex than a new referent (pro < full DPs)**.

Grammar and Parsing

- Grammars (generally) are **declarative devices** that does not specify algorithmically how an input must be analyzed.
- **non-determinism** (multiple options all equally suitable in a given context) and **recursion** are critical in parsing: not all rules lead to a grammatical tree-structure in the end... And sometimes some algorithm could not even terminate!

Problem Space and searching strategies

- Given a **sentence** and a **grammar** the parser should tell us if the sentence is generable by the grammar (**URP**, **Universal Recognition Problem**) and, in the affirmative case, provide an adequate tree structure
- The problem space is the complete forest of trees and subtrees that can be legally generated by applying the grammatical rules in a given context

Problem Space and searching strategies

- Italian sentence:
  la vecchia legge la regola
  («the old rule regulates it» vs. «the old woman reads the rule»)
- Grammar:

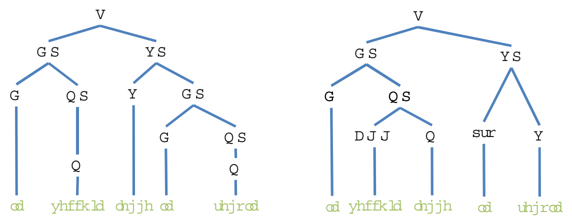
→ +qrq#whup lqdov,	→ +whup lqdov,
V#→ G S#Y S	sur#→ od
Y S#→ Y#G S	G #→ od#
Y S#→ sur#Y	D J J #→ yhffkld
G S#→ G #Q S	Q #→ yhffkld
Q S#→ +D J J ,#Q >	Q #→ dnjjh
	Q → uhjrod#
	Y #→ dnjjh
	Y → uhjrod#

Problem Space and searching strategies

- Two main constraints:
  1. Grammatical rules predicts that from a root node S certain expansion will lead to terminals;
  2. The words in the sentence, indicates how the S expansions must terminate
- We can start from the **root node S** for generating the structure:
  Top-Down or goal-driven algorithm
- We can start from single words, trying to combine then in phrases up to the root node S:
  Bottom-Up, or data-driven algorithm

Top-Down Parsing Algorithm

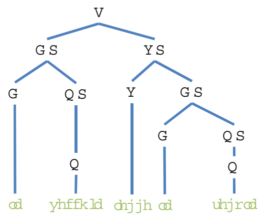
- A simple (blind) **top-down** algorithm explores all possible expansion of S offered by the grammar (assuming parallel expansions affects memory usage).

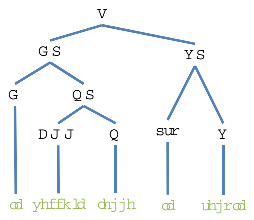


- Notice that “la regola regola la regola”, “la legge legge la vecchia legge”… will be plausible analysis proposed by the Top-Down algorithm.

Bottom-Up Parsing Algorithm

- Historically, the first parsing algorithm (Yngve 55) and possibly the most common (e.g. in programming languages parsers). It starts from lexical elements, that are terminal symbols, and, phrase by phrase, up to S:





What's better?

- Top-Down strategy doesn't loose time generating ungrammatical trees, but it generates sentences without considering the input till the end.
- Bottom-Up strategy, will be locally consistent with the input, but it will generate ungrammatical phrases unable to be rejoined under the root node S**S**.
- Both blind strategies are complete, then roughly equivalent, but:

  a. Consider starting from the side with the **most precise** (unambiguous) **information**

  b. Explore the tree trying to be guided by the **smallest possible ramification factor**.

LEFT CORNER Algorithm

- **Basic idea**
  combination of a Top-Down strategy, filtered by Bottom-Up considerations.
- **Left-corner rule**
  Every non-terminal category will be rewritten at some point by a word in the input
  Then B if the «left-corner» of the A category
  IFF A →* B → α.
- Off-line table of left corner given a standard grammar:

fdwhjru|	V	GS	YS
dhiw0fruqhu	G/Q $_{surshu}$/Y	G/Q $_{surshu}$	dx{/Y

Unresolved problems

- **Left-recursion**
  A →* Aα (es. DP → DP PP)
  how do we stop?
- **ambiguity**
  - **PP attachment** (I saw a man with the binocular)
  - **coordination** («papaveri e paperi rossi», *red poppies and ducks*)

  exponential growth of alternatives (Church e Patil 82) with respect to the number of PPs (3 PPs up to 5 possible analyses, 6 PPs up to 469 possible analyses… 8 PPs … 4867 possible analyses!).

Unresolved problems

c. Inefficiency in **subtrees analysis**
backtracking is not needed in certain analysis:
a flight from Rome to Milano at 7:00PM with a Boeing 747

DP → D N	*(ok, but incomplete…)*
DP → D N PP	*(ok, but incomplete…)*
DP → D N PP PP	*(ok, but incomplete…)*
…	

Dynamic Programming

- **dynamic programming** reuse useful analysis by storing them in tables (or charts).
- Once sub-problems are resolved (sub-trees in parsing), a global solution is attempted by merging partial solutions together.

Dynamic Programming: Earley Algorithm

- **Earley Algorithm** (Earley 1970) is a classic example of Top-Down, Parallel, Complete dynamic programming approach.
- The problem complexity (remember that generalized parsing is **NP-hard**) is reduced to **Polynomial complexity**. In the worst case: **$O(n^3)$**.
- One input pass, from left-to-right, partial analyses are stored in **chart** with *n*+1 entries, with *n* equals to the input length .

Dynamic Programming: Earley Algorithm

- Each **chart entry** will include three levels of information:
  - A **subtree** corresponding to **one single grammatical rule**
  - the **progress** in the completion of the rule (we use a dot • indicating the processing step, the rule is then dubbed "**dotted rule**")
  - the **position** of the subtree with respect to the input position (two numbers indicating where the rule began and where the rule is applied now;
    e.g. DP → D • NP [0,1] the rule started at the beginning of the input (position 0) and it is waiting between the first and the second word (position 1))

Dynamic Programming: Earley Algorithm

- Three fundamental operations are combined in Earley Algorithm:
  - **Predictor**
    add new rules in the chart, representing top-down expectations in the grammar; every rule in the grammar that is an expansion of a non-terminal or pre-terminal node to the right of the dot will be added here.
    e.g. S → • DP VP [0,0] DP → • D NP [0,0]
  - **Scanner**
    check the input, in the expected position, and trigger an advancement when the word is recognized as belonging to the expected POS. A correct scan introduce a new rule in the next position of the chart.
    e.g. DP → • D NP [0,0] iff D → article, then DP → D • NP [0,1]
  - **Completer**
    when the dot reached the end of the rule, the algorithm informs the chart that at the rule starting position, the category has been recognized, hence advancing the rules with the dot to the left of the relevalt category:
    e.g. NP → AGG N • [1,3] will advance the rule in the [1] position,
    DP → D • NP [0,1] adding DP → D NP • [0,3];

Some consideration on efficiency and plausibility

- A grammar can avoid considering **space/time limits** while focusing only on descriptive adequacy;
- the **parser** should take into consideration such limits. It happens than one grammar can be used by different parsing algorithms.
  The adequacy of the parser can be a matter of computational performance or psycholinguistic plausibility
- **token transparency** (Miller e Chomsky 63) or **strict isomorphism** (is the null hypothesis) the parser implements exactly the derivation suggested by the grammar.
- **type transparency** (Bresnan 78) suggests that, overall, the parsers implements different derivations with respect to the grammar, but overall, the same phenomena (e.g. passive constructions) are processed, globally, in a coherent way.

Some consideration on efficiency and plausibility

- **covering grammars** (Berwick e Weinberg 83, 84) parse and grammar must cover the same phenomena. But the parser should be psycholinguistically plausible or computationally efficient then implementing derivations that are not included in the grammar.

Next lecture

- **Lab time (please bring your own laptop)**
  - Writing Context-Free Grammars
  - Using parsing algorithms to test efficiency/plausibility