Introduction to Parsing

- What is a parser?
- Under what criteria can they be evaluated?
- Parsing strategies
  - top-down vs. bottom-up
  - left-right vs. right-left
  - depth-first vs. breadth-first
- Parsing strategy of Prolog executing DCGs

Parsers and criteria to evaluate them

- Function of a parser:
  - grammar + string $\rightarrow$ analysis trees
- Main criteria for evaluating parsers:
  - correctness
  - completeness
  - efficiency

Correctness

A parser is **correct** iff for every grammar and for every string, every analysis returned by parser is an actual analysis.

Correctness is nearly always required (unless simple post-processor could eliminate wrong analyses)

Completeness

A parser is **complete** iff for every grammar and for every string, every correct analysis is found by the parser.

- In theory, always desirable.
- In practice, essential to find the 'relevant' analysis first (possibly using heuristics).
- For grammars licensing an infinite number of analyses this means: there is no analysis that the parser could not find.

Efficiency

- One can reason about complexity of (parsing) algorithms by considering how it will deal with bigger and bigger examples.
- For practical purposes, the factors ignored by such analyses are at least as important.
  - profiling using typical examples important
  - finding the (relevant) first parse vs. all parse
- Memoization of complete or partial results is essential to obtain efficient parsing algorithms.

Complexity classes

If n is the length of the string to be parsed, one can distinguish the following complexity classes:

- **constant**: the amount of work does not depend on n
- **logarithmic**: the amount of work behaves like $log_k(n)$, for some constant k

Complexity classes (cont.)

- **polynomial**: the amount of work behaves like n^k, for some constant k. This is sometimes subdivided into the cases
  - **linear** $(k = 1)$
  - **quadratic** $(k = 2)$
  - **cubic** $(k = 3)$
  - ...
- **exponential**: the amount of work behaves like k^n, for some constant k.

Complexity and the Chomsky hierarchy

Grammar type	Worst-case complexity of recognition
regular (3)	linear
context-free (2)	cubic (n^3)
context-sensitive (1)	exponential
general rewrite (0)	undecidable

Recognition with type 0 grammars is **recursively enumerable**: if a string x is in the language, the recognition algorithm will succeed, but it will not return if x is not in the language.

Parsing strategies

1. What do we start from?
   - top-down vs. bottom-up
2. In what order is the string or the RHS of a rule looked at?
   - left-to-right, right-to-left, island-driven, . . .
3. How are alternatives explored?
   - depth-first vs. breadth-first

Direction of processing: Top-down

Goal-driven processing is Top-down:

- Start with the start symbol
- Derive sentential forms.
- If the string is among the sentences derived this way, it is part of the language.

Direction of processing: Bottom-up

Data-driven processing is Bottom-up:

- Start with the sentence.
- For each substring σ of each sentential form $\alpha\sigma\beta$, find each grammar rule $N \rightarrow \omega$ to obtain all sentential forms $\alpha N \beta$.
- If the start symbol is among the sentential forms obtained, the sentence is part of the language.

Problem: Epsilon rules ($N \rightarrow \epsilon$).

The order of looking at substrings or a RHS

Left-to-Right

- Use the leftmost symbol first, continuing with the next to its right

Problem for top-down, left-to-right processing: left-recursion (e.g., N' $\rightarrow$ N' PP) leads to non-termination.

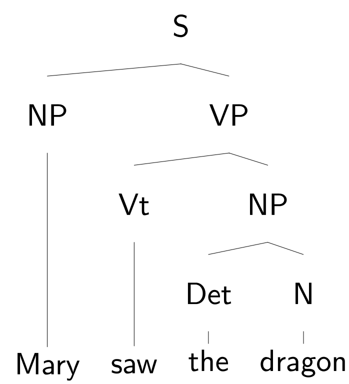
How are alternatives explored? Depth-first

- At every choice point: Pursue a single alternative completely before trying another alternative.
- State of affairs at the choice points needs to be remembered. Choices can be discarded after unsuccessful exploration.
- Depth-first search is generally not complete.

How are alternatives explored? Breadth-first

- At every choice point: Pursue every alternative for one step at a time.
- Requires massive bookkeeping since each alternative computation needs to be remembered at the same time.
- Search is guaranteed to be complete.

A small example



S → NP VP	NP → Det N'
VP → Vi	VP → Vt NP
Vt → saw	Vi → left
Det → the	Det → a
N' → N	N' → N' PP
N → dragon	N → cave
PP → P NP	PP → there
P → in	P → at
NP → Mary	NP → midnight

Compiling and executing DCGs in Prolog

- DCGs are a grammar formalism supporting any kind of parsing regime.
- The standard translation of DCGs to Prolog plus the proof procedure of Prolog results in a parsing strategy which is
  - top-down
  - left-to-right
  - depth-first