Parsing Irish with Mojo & Python

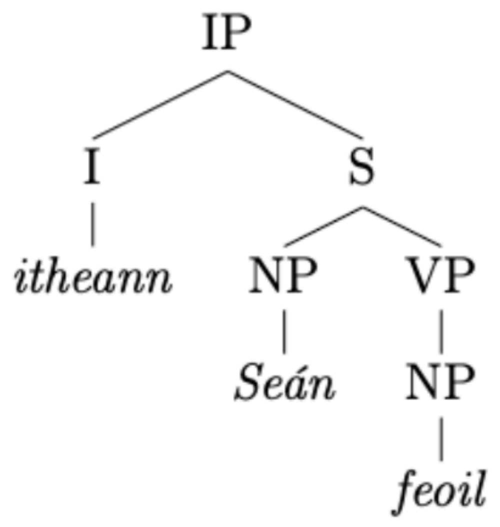
Petr Homola Dublin Sep 11, 2024

Outline

- What is parsing?
- Grammar fragment
- Mojo (“Python++”)
- Data structures for parsing
- Conclusions

What is parsing?

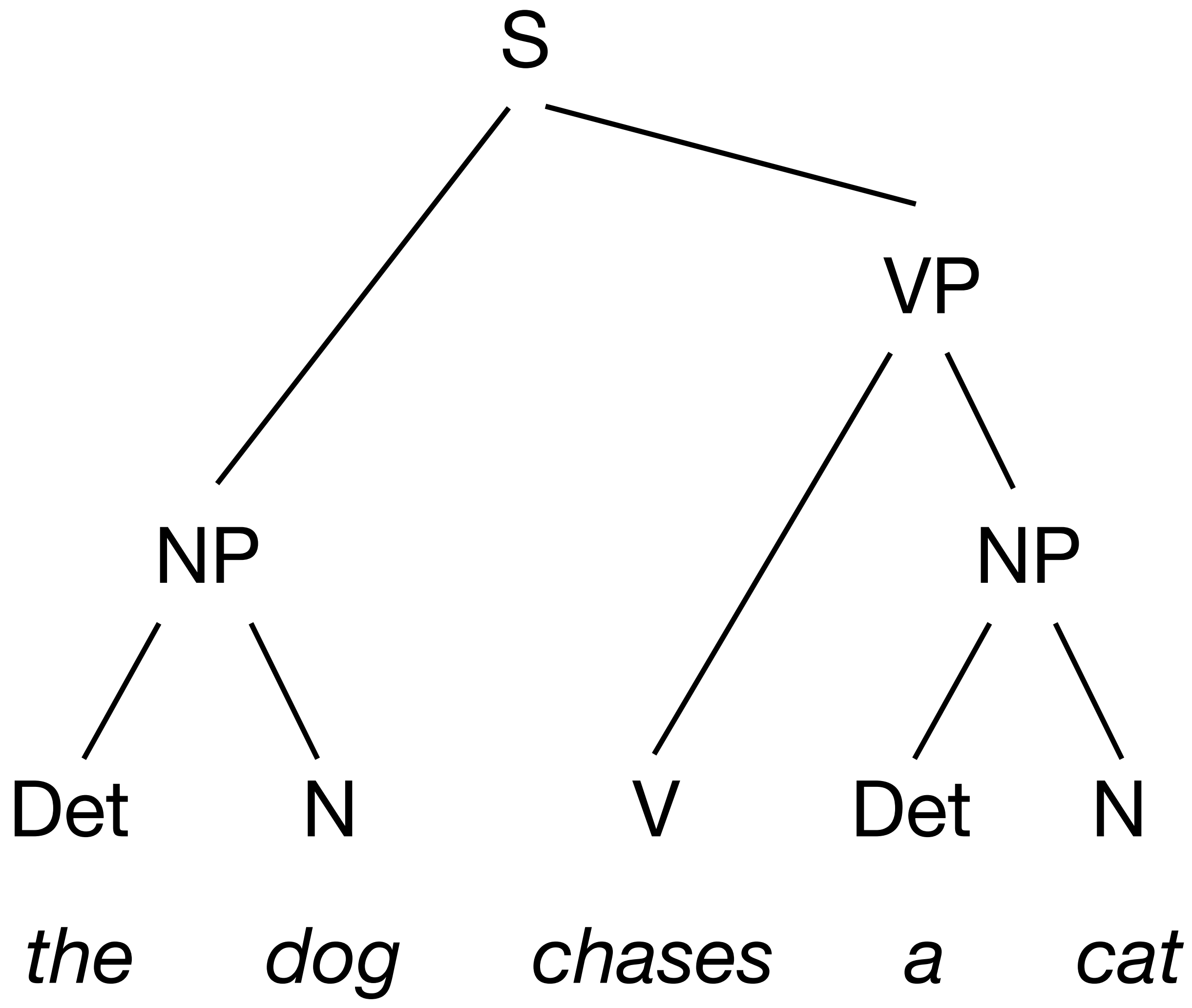
- Input
  - A phrase or sentence
    - *Itheann Seán feoil* “John eats meat”
- Output
  - Syntax tree
  - Feature structure
  - Semantic structure (logical formula)
    - *ith(Seán, feoil) / ith(x, y) & Seán(x) & feoil(y)*



$$\begin{bmatrix} \text{LEXSEM} & \text{‘ith’} \\ \text{SUBJ} & \begin{bmatrix} \text{LEXSEM} & \text{‘Seán’} \\ \text{CASE} & \textsc{nom} \end{bmatrix} \\ \text{OBJ} & \begin{bmatrix} \text{LEXSEM} & \text{‘feoil’} \\ \text{CASE} & \textsc{nom} \end{bmatrix} \\ \text{TENSE} & \textsc{pres} \end{bmatrix}$$

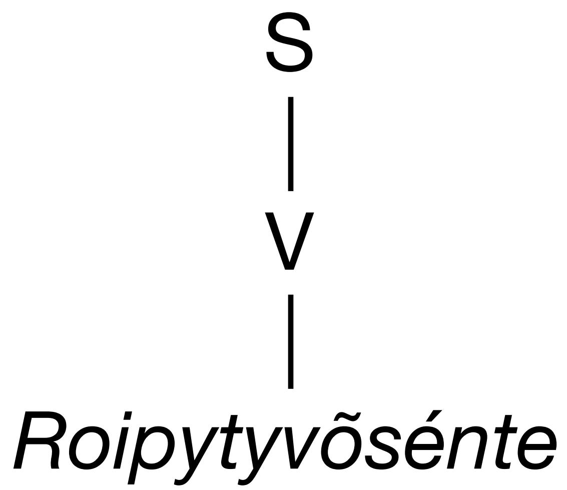
Configurational languages

Syntax more complex than morphology



Non-configurational languages

Morphology more complex than syntax



Roi-pytyvõ-sé-nte

I.thee-help-want-just

“I just want to help you.”

(Guaraní)

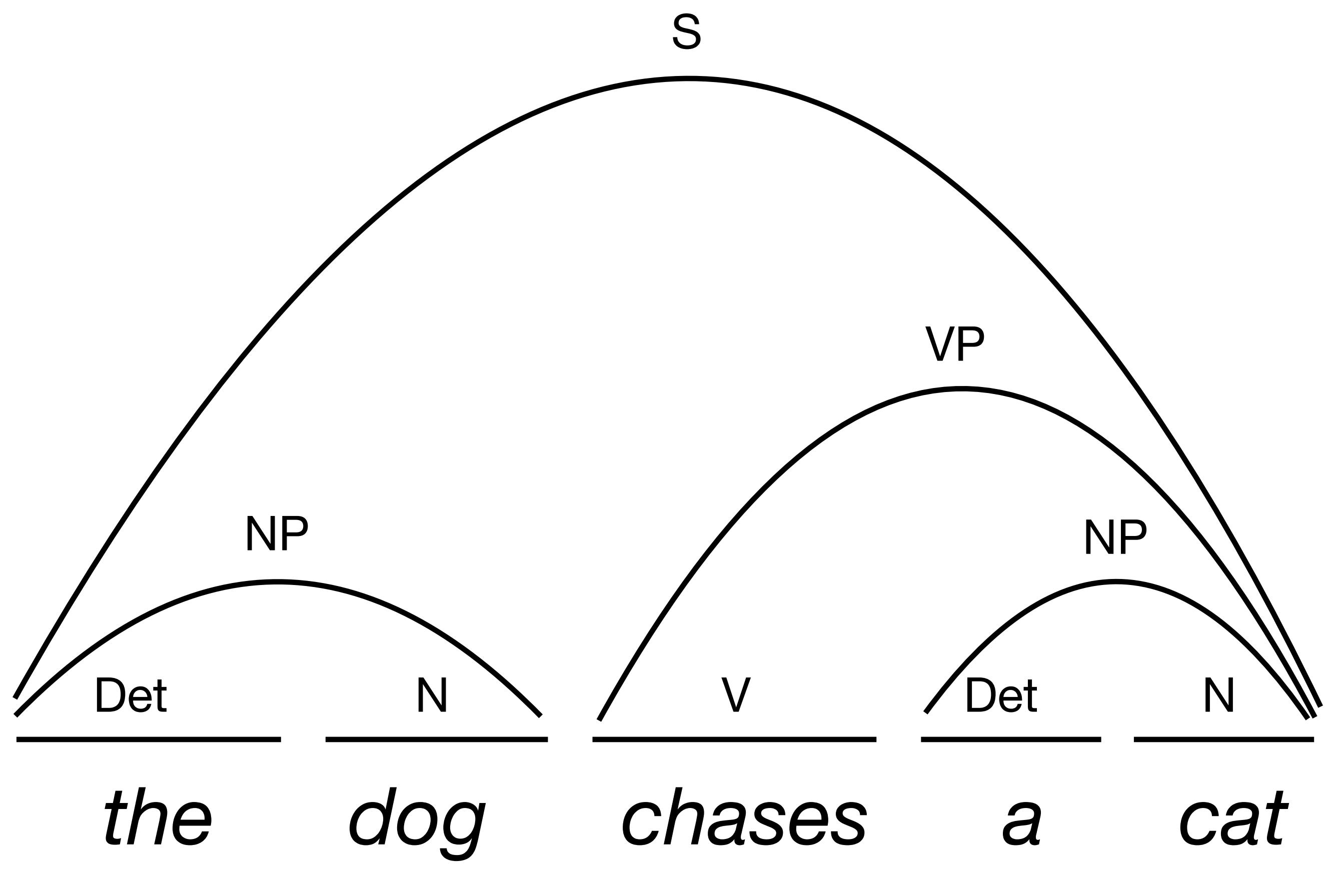
Feature structures

Attribute-value matrices (AVM)

- Recursive (values are either constants or “embedded” AVMs)
- May be unified (like sets)
  - Unification may fail

$$\begin{bmatrix} \text{LEMMA ‘bark’} \end{bmatrix} \sqcup \begin{bmatrix} \text{SUBJ} & \begin{bmatrix} \text{LEMMA ‘dog’} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \text{LEMMA ‘bark’} & \\ \text{SUBJ} & \begin{bmatrix} \text{LEMMA ‘dog’} \end{bmatrix} \end{bmatrix}$$

Chart parsing



The algorithm

- Try to apply all the rules to the chart
- Add all new edges to the chart
- Repeat until no rules can be applied
- Output the tree and AVM of the edge(s) spanning the input

Grammar fragment for Irish

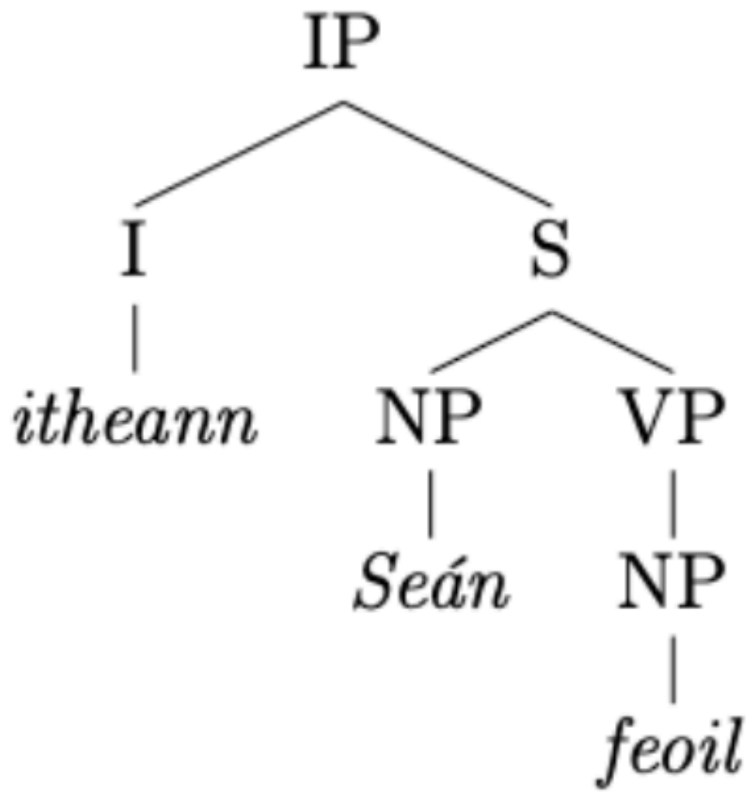
- Context-free grammar with annotations for AVMs
- `mojo parser.mojo ga.gr ga.in`

```
NP > N (=).

VP > NP (>obj obj.case=nom).

V' > V (= obj.case=gen) NP (>obj).

VP > P (= prep=ag) V' (=).

S > NP (>subj) VP (= subj.case=nom).

IP > I (=) S (=).
```

Example of analysis

- *Itheann Seán feoil* “John eats meat”

```
-1- I [lemma: "ith" tense: "pres"] -2-
-2- N [lemma: "Seán" case: "nom"] -3-
-3- N [lemma: "feoil" case: "nom"] -4-
```

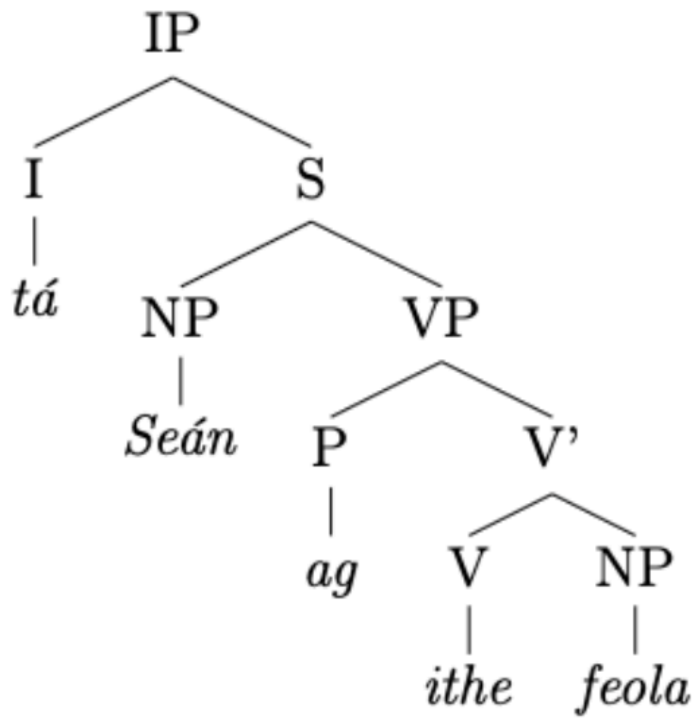


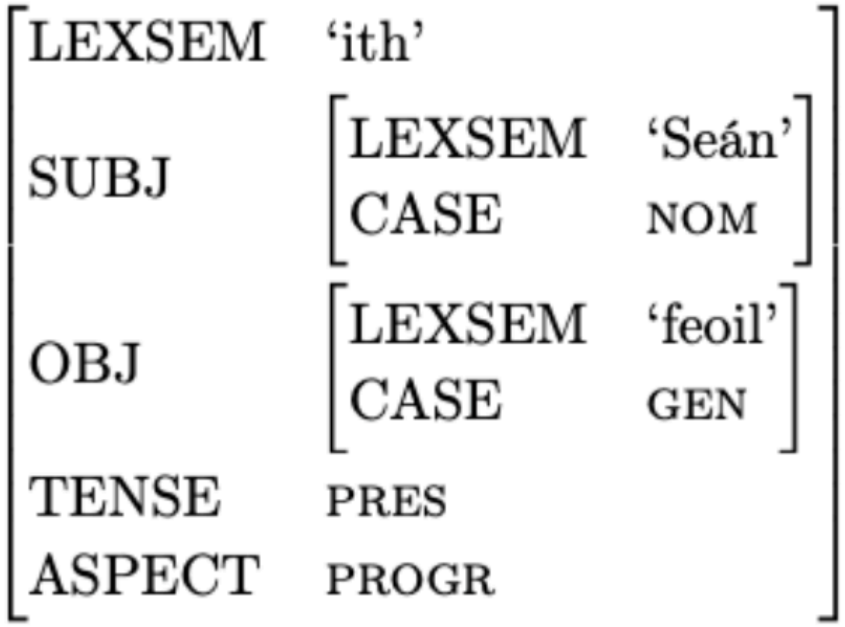
$$\begin{bmatrix} \text{LEXSEM} & \text{‘ith’} \\ \text{SUBJ} & \begin{bmatrix} \text{LEXSEM} & \text{‘Seán’} \\ \text{CASE} & \textsc{nom} \end{bmatrix} \\ \text{OBJ} & \begin{bmatrix} \text{LEXSEM} & \text{‘feoil’} \\ \text{CASE} & \textsc{nom} \end{bmatrix} \\ \text{TENSE} & \textsc{pres} \end{bmatrix}$$

Example of analysis

- *Tá Seán ag ithe feola* “John is eating meat”

```
-1- I [tense: "pres" aspect: "prog"] -2-
-2- N [lemma: "Seán" case: "nom"] -3-
-3- P [prep: "ag"] -4-
-4- V [lemma: "ith"] -5-
-5- N [lemma: "feoil" case: "gen"] -6-
```





Mojo (“Python++”)

- Superset of Python
- Statically typed
    - Safe & fast
- Seamless interop with Python
- Code runs natively on both CPUs and GPUs (via MLIR)

```
struct AVM:
    features: Dict[String, Variant[String, AVM]]
```

Memory management in Mojo

- Arguments passed by value
- Copy/move semantics
- References with lifetimes
- Memory released on last use

```
fn f1(borrowed x: MyType):
    …
fn f2(owned x: MyType):
    …
fn f3(x: Variant[String, AVM]):
    if x.isa[String]():
        print(x[String])
```

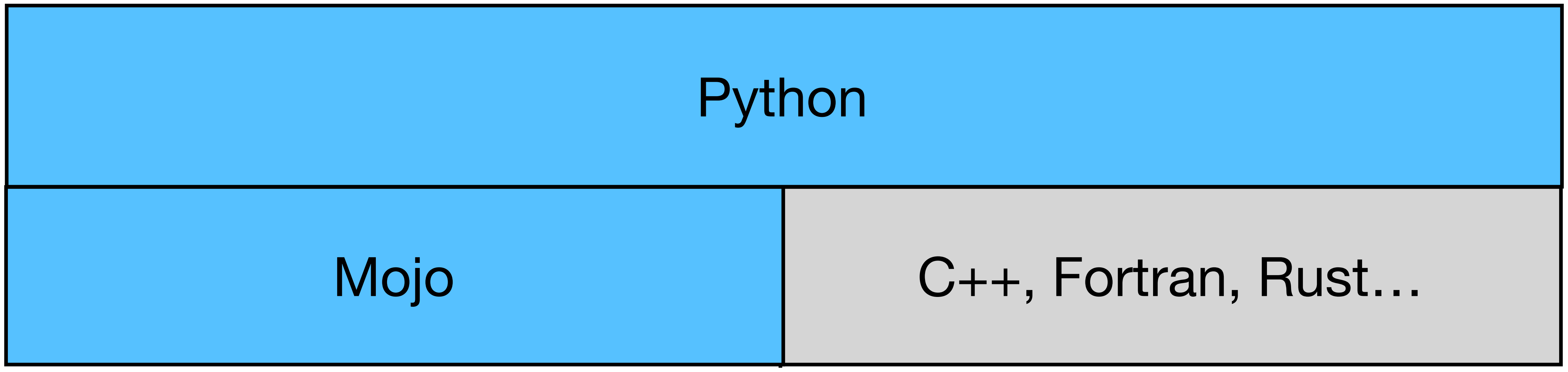
References

- “Safe pointers”, e.g. to elements of collections
- Explicit dereferencing (via `[]`)
- Checked lifetimes

```
var list = List(1, 2, 3, 4)
for n in list:
    print(n[])
    n[] += 1
```

Typical use of Mojo

- Seamless interop with Python (like C++/CLI, C++ ↔ .NET)



Data structures for parsing

```
struct AVM:
    features: Dict[String, Variant[String, AVM]]

struct Chart:
    var edges: Dict[Int, List[Edge]]

struct Rule:
    var lhs: String
    var rhs: List[String]
    var avmfn: fn(List[AVM]) -> Optional[AVM]
```

Conclusions

- Mojo is much like Python syntactically
- Mojo is like Rust under the hood
- Fast & good for symbolic computation
- `https://github.com/phomola/mojolibs`

Thank you

Go raibh maith agaibh