Prolog can be used for parsing context-free grammars

Here is a simple grammar:

s -> 'a' 'b'
s -> 'a' 'c'
s -> s s

Terminals: a, b
Non-terminals: s

Parsing by consumption

Write a predicate for each non-terminal which consumes as much as the first list which is necessary to match the non-terminal and returns the remaining elements in the second list

e.g.

s([a,b],[]), s([a,b,c,d],[c,d])

A Prolog program which accepts sentences from our grammar

```
s -> 'a' 'b'
s -> 'a' 'c'
s -> s s
```

```
c([X|T],X,T).

s(In,Out) :- c(In,a,In2), c(In2,b,Out).
s(In,Out) :- c(In,a,In2), c(In2,c,Out).
s(In,Out) :- s(In,In2), s(In2,Out).
```

Prolog provides a shortcut syntax for this

```
s --> [a],[b].
s --> [a],[c].
s --> s,s.
```

```
s -> 'a' 'b'
s -> 'a' 'c'
s -> s s
```

This will both test and generate:
s([a,c,a,b],[]) or s(A,[]).

Building a parse tree

```
c([X|T],X,T).

s(ab,In,Out) :- c(In,a,In2), c(In2,b,Out).
s(ac,In,Out) :- c(In,a,In2), c(In2,c,Out).
s(t(A,B),In,Out) :- s(A,In,In2), s(B,In2,Out).

:- s(Result,[a,c,a,b,a,b],[]).
```

Building a parse tree

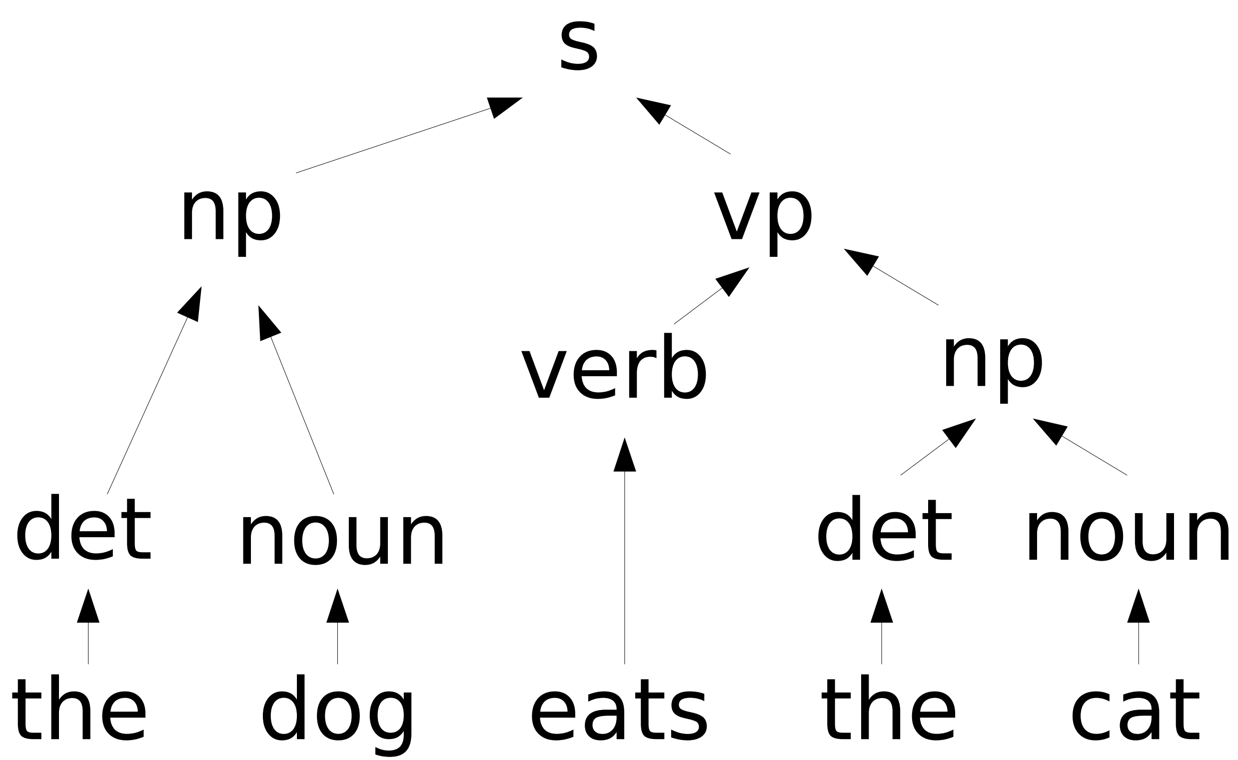
```
s(ab) --> [a],[b].
s(ac) --> [a],[c].
s(t(A,B)) --> s(A),s(B).
```

Parsing Natural Language (back to Prolog's roots)

```
s    --> np,vp.
np   --> det,n.
vp   --> v.
vp   --> v,np.

n    --> [cat].
n    --> [dog].
v    --> [eats].
det  --> [the].
```

This is a very limited grammar of English. Things get complicated very quickly – for more see the Natural Language Processing course next year (Prolog is not a pre-requisite)

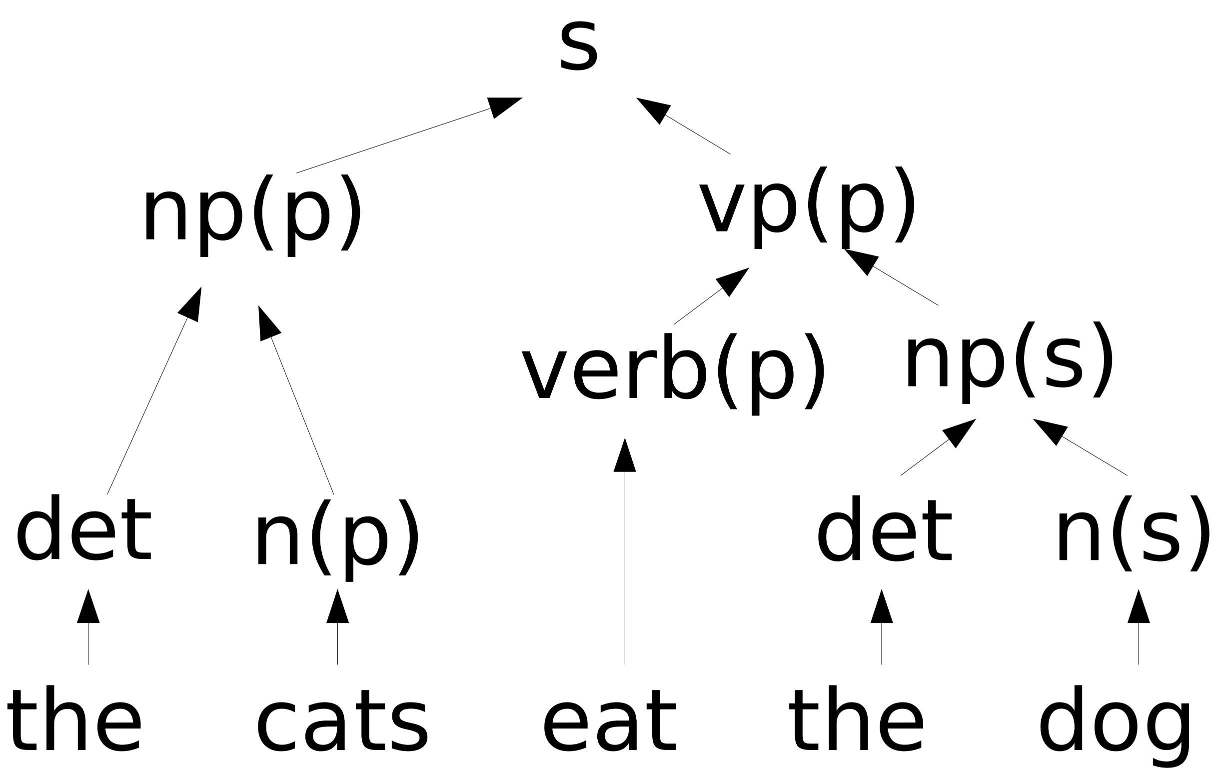

s
np
vp
verb
np
det
noun
det
noun
the
dog
eats
the
cat

We can also handle agreement

```
s(N)    --> np(N),vp(N).
np(N)   --> det,n(N).
vp(N)   --> v(N).
vp(N)   --> v(N),np(_).

n(s)    --> [cat].
n(s)    --> [dog].
n(p)    --> [cats].
v(s)    --> [eats].
v(p)    --> [eat].
det     --> [the].
```

We consider only
third-person constructions


s
np(p)
vp(p)
verb(p)
np(s)
det
n(p)
det
n(s)
the
cats
eat
the
dog

Things get much more complicated very quickly

Ambiguities, special cases and noise all make this hard to scale up to a full language

Closing Remarks

- Declarative programming is different to Functional or Procedural programming
  - Foundations of Computer Science & Programming in Java
- Prolog is built on logical deduction
  - formal explanation in Logic & Proof
- It can provide concise implementations of algorithms such as sorting or graph search
  - Algorithms I & Algorithms II

Closing Remarks

- Foundations of Functional Programming (Part IB)
  - Building computation from first principles
- Databases (Part 1B)
  - Find out more about representing data and SQL
- Artificial Intelligence (Part 1B)
  - Search, constraint programming and more
- C & C++ (Part 1B)
  - Doing useful stuff in the real world
- Natural Language Processing (Part II)
  - Parsing natural language