Implementing parsers

- Data structures: a parser configuration
- Top-down parsing
  - formal characterization
  - Prolog implementation
- Bottom-up parsing
  - formal characterization
  - Prolog implementation

A parser configuration

Assuming a left-to-right order of processing, a **configuration** of a parser can be encoded by a pair of

- the sequence of terminals or non-terminals recognized so far
- the string remaining to be recognized

More formally, for a grammar $G = (N, \Sigma, S, P)$, a parser configuration is a pair $< \alpha, \tau >$ with $\alpha \in (N \cup \Sigma)^*$ and $\tau \in \Sigma^*$

Top-down parsing

- **Start configuration** for recognizing a string ω: $< S, \omega >$
- **Available actions**:
  - **consume**: remove an expected terminal a from the string
    $< a\alpha, a\tau > \mapsto < \alpha, \tau >$
  - **expand**: apply a phrase structure rule
    $< A\beta, \tau > \mapsto < \alpha\beta, \tau >$ if $A \rightarrow \alpha \ \in P$
- **Success configuration**: $< \epsilon, \epsilon >$

A top-down parser in Prolog
(td_parser.pl)

```
% START
td_parse(String) :-
      td_parse([s],String).

% SUCCESS
td_parse([],[]).
```

```
% CONSUME
td_parse([H|T],[H|R]) :-
        td_parse(T,R).

% EXPAND
td_parse([A|Beta],String) :-
        (A ---> Alpha),
        append(Alpha,Beta,Stack),
        td_parse(Stack,String).
```

Bottom-up parsing

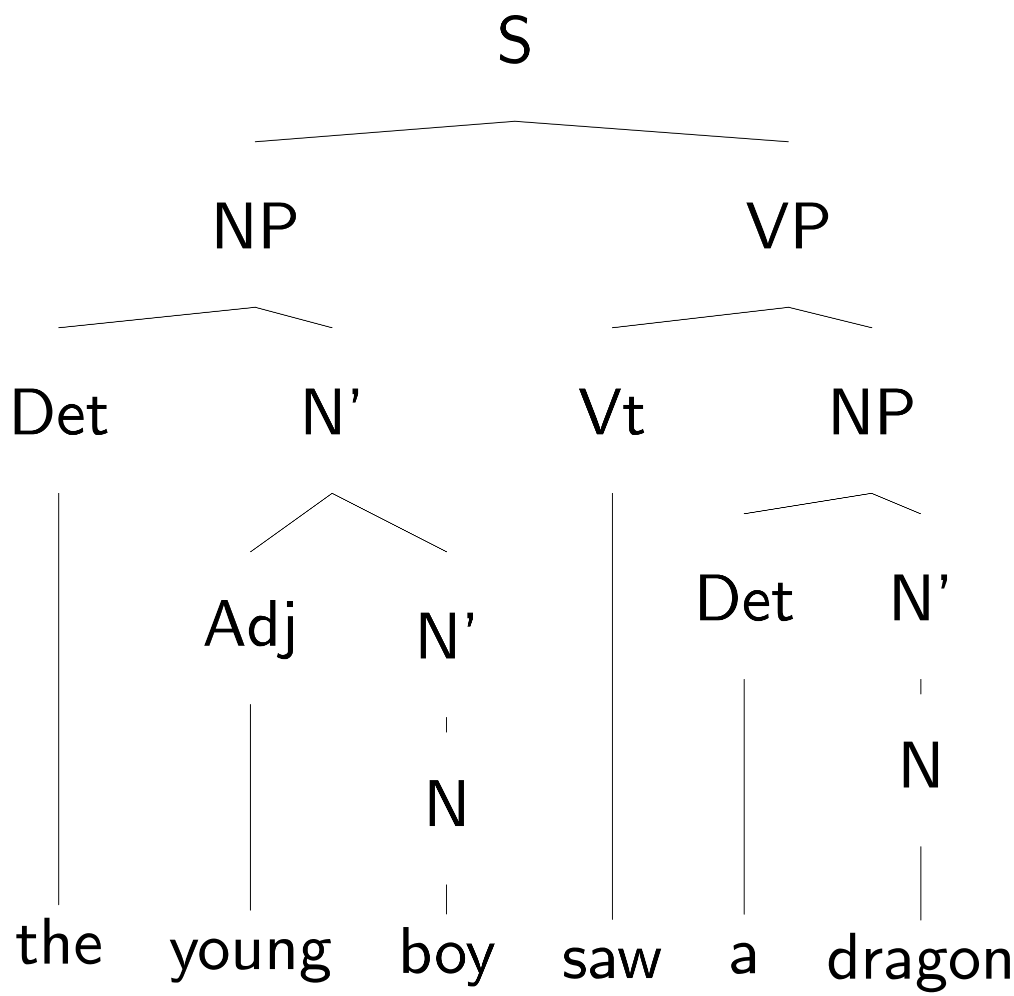
- **Start configuration** for recognizing a string ω: $< \epsilon, \omega >$
- **Available actions**:
  - **shift**: turn to the next terminal a of the string
    $< \alpha, a\tau > \mapsto < \alpha a, \tau >$
  - **reduce**: apply a phrase structure rule
    $< \beta\alpha, \tau > \mapsto < \beta A, \tau >$ if $A \rightarrow \alpha \in P$
- **Success configuration**: $< S, \epsilon >$

A shift-reduce parser in Prolog
(sr_parser.pl)

```
% START
sr_parse(String) :-
      sr_parse([],String).

% SUCCESS
sr_parse([s],[]).
```

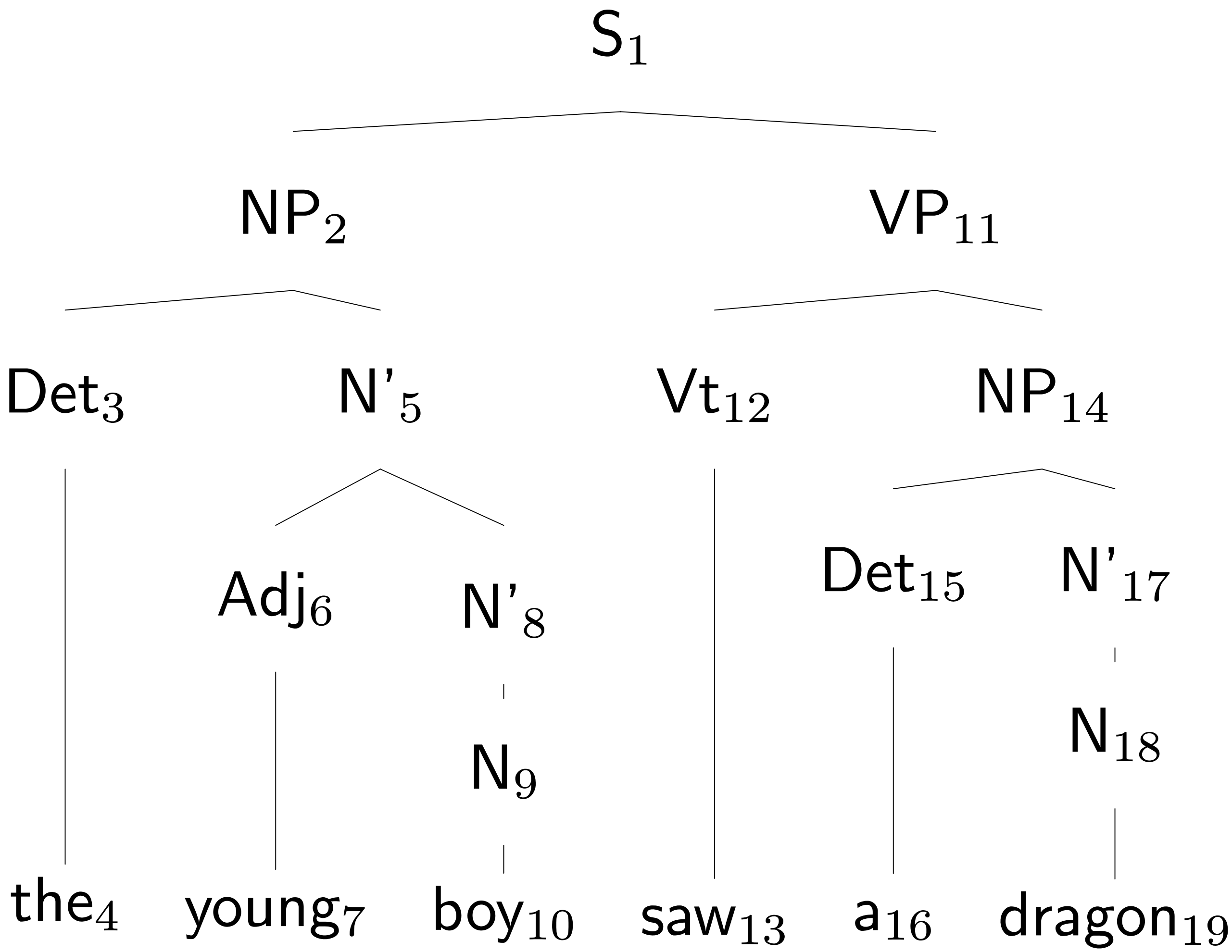
```
% REDUCE
sr_parse(Stack,String) :-
        append(Beta,Alpha,Stack),
        (A ---> Alpha),
        append(Beta,[A],NewStack),
        sr_parse(NewStack,String).

% SHIFT
sr_parse(Stack,[Word|String]) :-
        append(Stack,[Word],NewStack),
        sr_parse(NewStack,String).
```

An Example



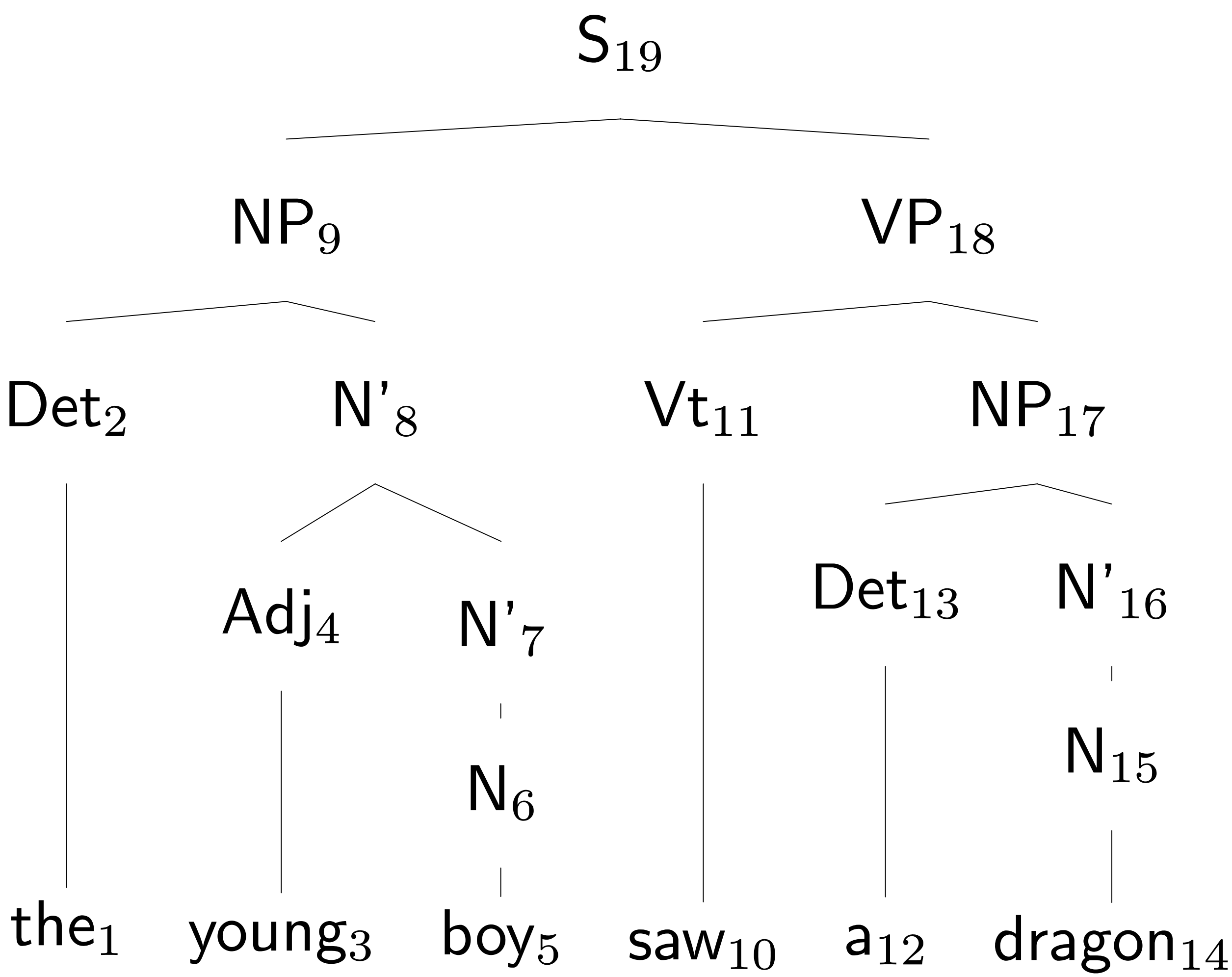
S → NP VP
VP → Vt NP
NP → Det N'
N' → N
N' → Adj N'
Vt → saw
Det → the
Det → a
N → dragon
N → boy
Adj → young

Top-Down, left-right, depth-first tree traversal



S $\longrightarrow$ NP VP
VP $\longrightarrow$ Vt NP
NP $\longrightarrow$ Det N'
N' $\longrightarrow$ N
N' $\longrightarrow$ Adj N'
Vt $\longrightarrow$ saw
Det $\longrightarrow$ the
Det $\longrightarrow$ a
N $\longrightarrow$ dragon
N $\longrightarrow$ boy
Adj $\longrightarrow$ young

Bottom-up, left-right, depth-first tree traversal



S $\rightarrow$ NP VP
VP $\rightarrow$ Vt NP
NP $\rightarrow$ Det N'
N' $\rightarrow$ N
N' $\rightarrow$ Adj N'
Vt $\rightarrow$ saw
Det $\rightarrow$ the
Det $\rightarrow$ a
N $\rightarrow$ dragon
N $\rightarrow$ boy
Adj $\rightarrow$ young