1 ICL/Chart Parsing CFGs/2005-11-07

Contents

2 Review Top-down Parsing

Parsing
Parsing with a CFG is the task of assigning a correct tree (or derivation) to a string given some grammar. A correct tree is:

- consistent with the grammar, and
- the leaves of the tree cover all and only the words in the input.

There may be a very large number of correct trees for any given input . . .

Problems that arise

- Left Recursion
- Ambiguity
- Inefficiencies due to backtracking

Common substructures

- Despite ambiguity and backtracking there are commons substructures to be taken advantage of.
- Consider parsing the following NP:

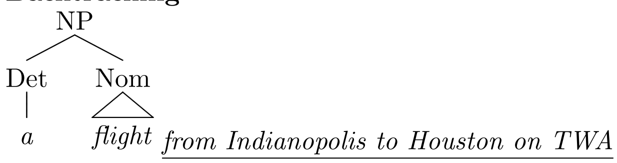
  a flight from Indianapolis to Houston on TWA

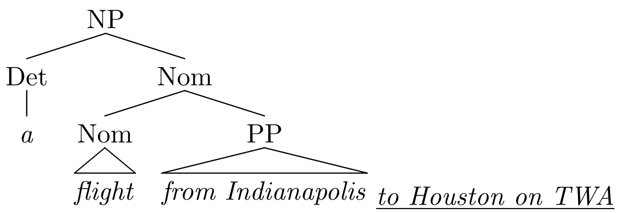
  with the following rules:

  NP → Det Nom
  Nom → Nom PP

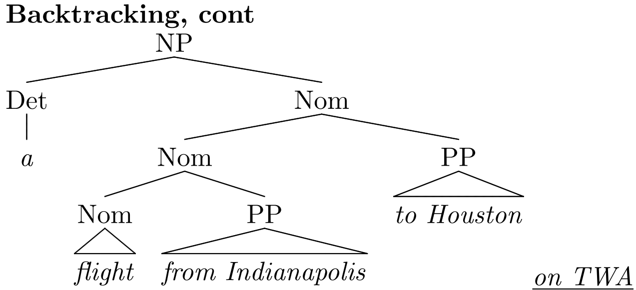
- What happens with a top-down parser?

Backtracking

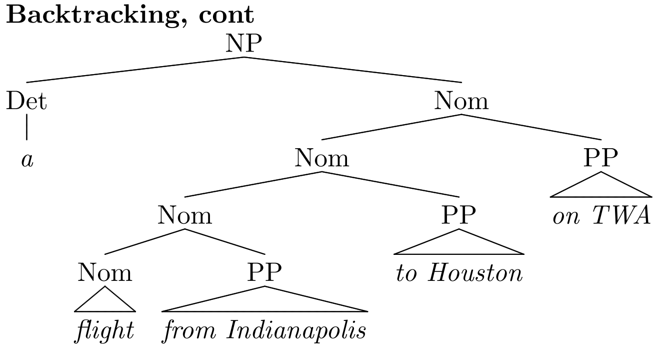




Backtracking, cont



Backtracking, cont



3 Chart Parsing

3.1 Overview

Dynamic Programming

Our current algorithm builds valid trees, discards them during backtracking, then rebuilds them.

- the subtree for *a flight* was derived 4 times.

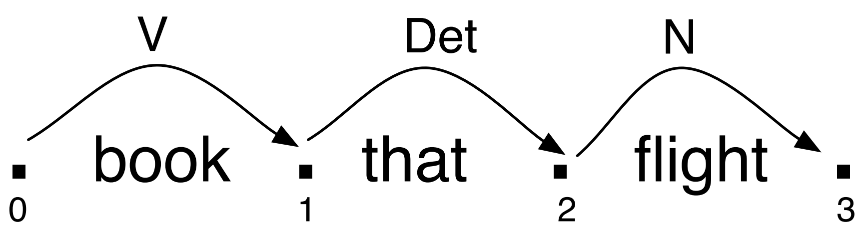
Dynamic programming is one answer to problems that have sub-problems that get solved again and again We'll consider an algorithm that fills a table with solutions to subproblems that:

- does a parallel top-down search with bottom-up filtering
- does not do repeated work
- solves the left-recursion problem

Dynamic Programming and Parsing

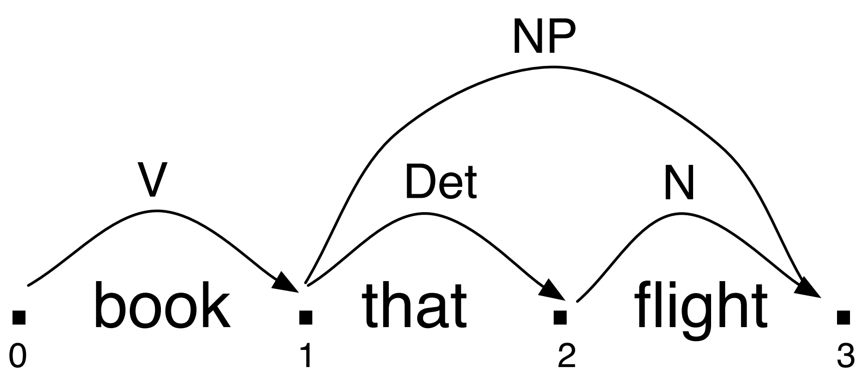
- Systematically fill in tables of solutions to subproblems.
- When complete, the table contains all possible solutions to all of the subproblems needed to solve the whole problem
- For parsing:
  - the table stores subtrees for constituents.
  - Solves reparsing inefficiencies, because subtrees are not reparsed but looked up.
  - Solves ambiguity explosions, because the table *implicitly* stores all parses.
  - Each subtree is represented only once and shared by all parses that need it.

Lexical Edges

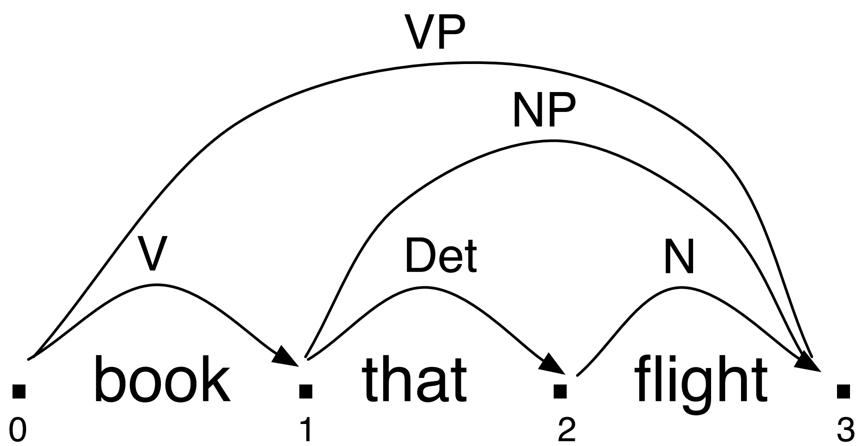


3.2 Charts as Graphs

Nonterminal Edges: NP

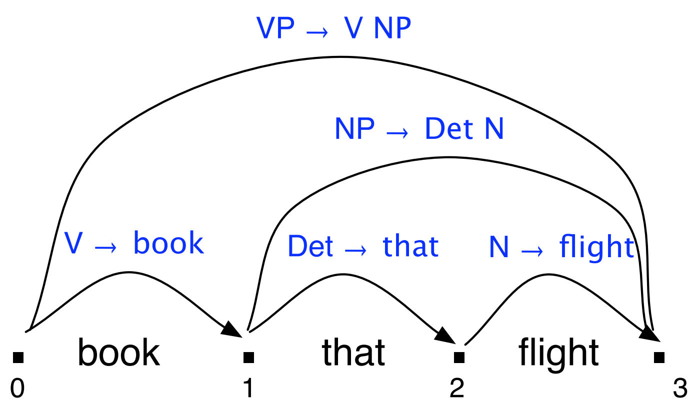


Nonterminal Edges: VP



Nonterminal Edges: Rules

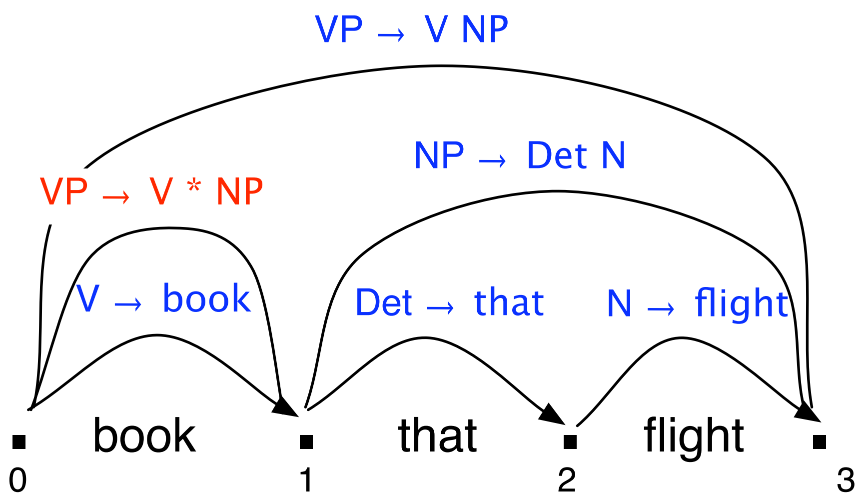
- Useful to label arcs with rules rather than categories:



Incomplete Edges

- Recall that the parser can make predictions on the basis of rules:
  - I'm trying to expand an VP, and I've found a V so I'll start looking for an NP.
- Record incomplete constituents with dotted rules:
  - Dot on the RHS of a rule shows what weve found already: VP → V • NP
  - We can use this as a label for an incomplete constituent.

Incomplete Edges, cont.



3.3 The Basic Idea

Dynamic Programming and Parsing
The Earley algorithm:

- fills a table (the chart) in a single left-to-right pass over the input.
- The chart will be size $N + 1$, where N is the number of words in the input.
- Chart entries are associated with the gaps between the words — like slice indexing in Python.
- For each word position in the sentence, the chart contains a set of edges representing the partial parse trees generated so far.
- So Chart[0] is the set of edges representing the parse before looking at a word.

States

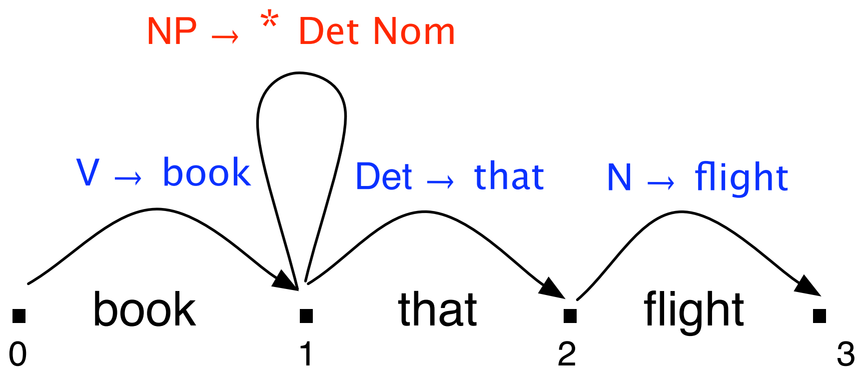
- J&M call the chart entries states.
- The chart entries represent three distinct kinds of things:
  - completed constituents;
  - in-progress constituents; and
  - predicted constituents
- The three kinds of states correspond to different dot locations in dotted rules:

  Completed: VP → V NP •

  In-progress: NP → Det • Nom

  Predicted: S → • VP

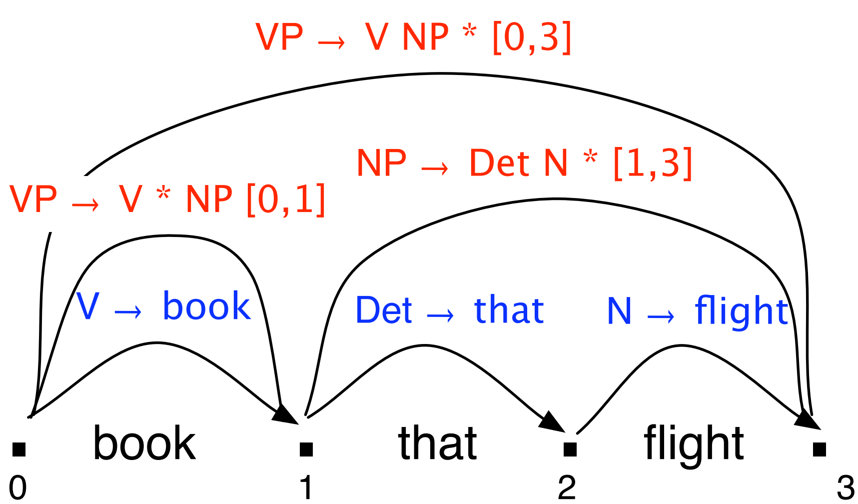
Incomplete NP Edge: Self-loop



States, cont.

- Given dotted rules like those we've just seen, we need to record:
  - where the represented constituent is in the input, and
  - what its parts are.
- So we add a pair of coordinates $[x, y]$ to the state:
  - $A \rightarrow \alpha,\ [x, y]$
  - x indicates the position in input where the state begins
  - y indicates position of dot

Example with coordinates



3.4 Example States

States, cont.
Example states in parsing *Book that flight*:

1. S → • VP, [0,0]

    - First 0 indicates that the constituent begins at the start of the input.
    - Second 0 indicates that the dot also begins at start of input, and thus indicates a top-down prediction.

2. NP → Det • Nom, [1,2]

    - the NP begins at position 1
    - the dot is at position 2
    - Det has been successfully *completed*
    - Nom is predicted next

States, cont.

1. VP → V NP •, [0,3]

    - VP is completed
    - no further predictions from this rule
    - a successful parse that spans the entire input

Success
The final answer is found by looking at the last column of the table. In particular, for an input of N words, if we find the following kind of state in the chart then we've succeeded:

S → α •, [0,N]

4 The Earley Algorithm

Parsing
Parsing is sweeping through the chart creating the three kinds of states as we go. States are never removed, and we never backtrack.

- New *predicted* states are based on existing table entries (predicted, or in-progress) that predict a certain constituent at that spot.
- New *in-progress* states are created by updating older states to reflect the fact that previously expected completed constituents have been located.
- New *completed* states are created when the dot in an in-progress state moves to the end.

More Specifically

1. Predict all the states you can.
2. Read an input.
    - See what predictions you can match.
    - Extend matched states, add new predictions.
    - Go to next state (goto 2)
3. At the end, see if $state[N+1]$ contains a complete S.

Earley Algorithm
The Earley algorithm has three main functions that do all the work:

Predictor: Adds predictions into the chart

Completer: Moves the dot to the right when new constituents are found

Scanner: Reads the input words and enters states representing those words into the chart

Predictor

```
procedure PREDICTOR((A → α • B β, [i, j]))
    for each (B → γ) in GRAMMAR-RULES-FOR(B, grammar) do
        ENQUEUE((B → • γ, [j, j]), chart[j])
    end
```

- Intuition: new states represent top-down expectations.
- Applied when a state has a non-terminal to the right of a dot that is not a part-of-speech.
- Generates one new state for each alternative expansion of the non-terminal in the grammar.
- Adds states to the same chart entry as generating state.

Completer

```
procedure COMPLETER((B → γ •, [j, k]))
    for each (A → α • B β, [i, j]) in chart[j] do
        ENQUEUE((A → α B • β, [i, k]), chart[k])
    end
```

- Intuition: parser has discovered a constituent, so must find and advance states that were looking for this grammatical category at this position in input.
- Applied when dot has reached right end of rule.
- New states are generated by copying old state and advancing dot over expected category.
- Adds new states to same chart entry as generating state.

Scanner

```
procedure SCANNER((A → α • B β, [i, j]))
    if B ∈ PARTS-OF-SPEECH(word[j]) then
        ENQUEUE((B → word[j] •, [j, j + 1]), chart[j+1])
```

- New states for predicted part-of-speech.
- Applicable when part-of-speech is to the right of a dot.
- Adds states to next chart entry.

Note: Earley parser uses top-down predictions to help disambiguate part-of-speech ambiguities. Only those parts-of-speech of a word that are predicted by some state will find their way into the chart.

4.1 Parsing Example

Mini grammar and lexicon

S → NP VP | Aux NP VP | VP	Det → *that* | *this* | *a*
NP → Det Nom | PropN	N → *book* | *flight* | *meal*
Nom → Nom PP | N Nom	V → *book* | *include* | *prefer*
PP → P NP	Aux → *does*
VP → V | V NP	P → *from* | *to* | *on*
Nom → N PP | N Nom	PropN → *Houston* | *TWA*

Example: Chart[0] and Chart[1]

Chart[0]

$\gamma \rightarrow \bullet\ S$	[0,0]	Dummy start state
$S \rightarrow \bullet\ NP\ VP$	[0,0]	Predictor
$S \rightarrow \bullet\ Aux\ NP\ VP$	[0,0]	Predictor
$S \rightarrow \bullet\ VP$	[0,0]	Predictor
$NP \rightarrow \bullet\ Det\ NOMINAL$	[0,0]	Predictor
$NP \rightarrow \bullet\ Proper\text{-}Noun$	[0,0]	Predictor
$VP \rightarrow \bullet\ Verb$	[0,0]	Predictor
$VP \rightarrow \bullet\ Verb\ NP$	[0,0]	Predictor

Chart[1]

$Verb \rightarrow book\ \bullet$	[0,1]	Scanner
$VP \rightarrow Verb\ \bullet$	[0,1]	Completer
$S \rightarrow VP\ \bullet$	[0,1]	Completer
$VP \rightarrow Verb \bullet NP$	[0,1]	Completer
$NP \rightarrow \bullet\ Det\ NOMINAL$	[1,1]	Predictor
$NP \rightarrow \bullet\ Proper\text{-}Noun$	[1,1]	Predictor

Example: Chart[1] and Chart[2]

Chart[1]

$Verb \rightarrow book\ \bullet$	[0,1]	Scanner
$VP \rightarrow Verb\ \bullet$	[0,1]	Completer
$S \rightarrow VP\ \bullet$	[0,1]	Completer
$VP \rightarrow Verb \bullet NP$	[0,1]	Completer
$NP \rightarrow \bullet\ Det\ NOMINAL$	[1,1]	Predictor
$NP \rightarrow \bullet\ Proper - Noun$	[1,1]	Predictor

Chart[2]

$Det \rightarrow that\ \bullet$	[1,2]	Scanner
$NP \rightarrow Det{\bullet}NOMINAL$	[1,2]	Completer
$NOMINAL \rightarrow \bullet\ Noun$	[2,2]	Predictor
$NOMINAL \rightarrow \bullet\ Noun\ NOMINAL$	[2,2]	Predictor

Example: Chart[3]

Chart[3]

Noun → *flight* •	[2,3]	Scanner
NOMINAL → *Noun* •	[2,3]	Completer
NOMINAL → *Noun* • *NOMINAL*	[2,3]	Completer
NP → *Det NOMINAL* •	[1,3]	Completer
VP → *Verb NP* •	[0,3]	Completer
S → *VP* •	[0,3]	Completer
NOMINAL → • *Noun*	[3,3]	Predictor
NOMINAL → • *Noun NOMINAL*	[3,3]	Predictor

4.2 Left Recursion

Examples: Left Recursion

What about parsing the NP*a flight from Denver to Boston* with the following rules:

NP → NP PP

NP → Det Nom

NP → Proper-Noun

- We construct the state (NP → • NP PP, [0,0]) and add it to *chart*[0]
- The PREDICTOR function then requires us to find a rule which expands the (non-lexical) category immediately to the right of the dot.
- So let's pick the first rule above, and ENQUEUE the state (NP → • NP PP, [0,0]).
- But this is already in the state, so we don't add it again.

5 Reading

Reading

- Read section 10.4 of J&M
- Read the NLTK-Lite Tutorial on Chart Parsing