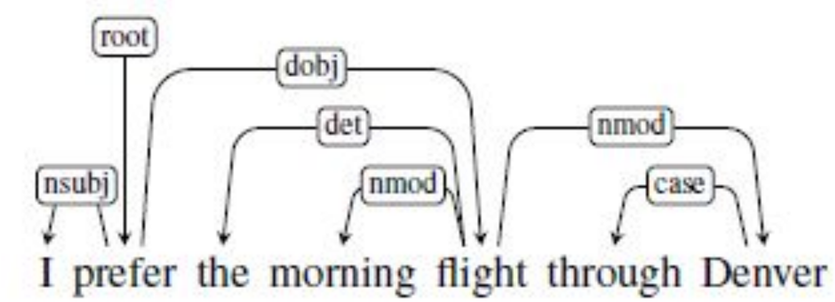
Dependency parsing

Lê Thiện Toàn - ICT2019
Lê Thị Thùy Trang - QCF2019

Outline

I. Introduction

- Dependency Grammars focus on the syntactic structure that involves:
  - Words (or lemmas) in the sentence
  - Directed binary relations between pairs of words
- Standard graphical method for illustrating Dependency structure:



- The relations are shown by directed labeled arcs: (head -> dependent)
- The labels are pre-defined grammatical relations

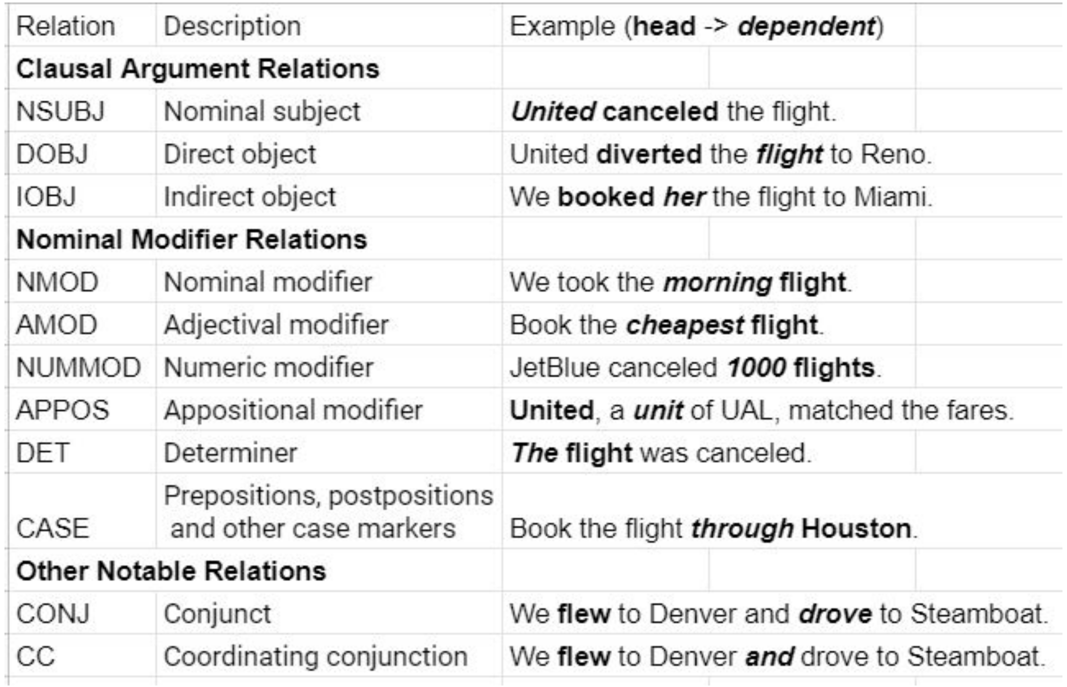
I. Introduction

Advantages of Dependency Grammar:

- The ability to process languages that are morphologically rich and have relatively **free word order** ⇒ abstracting away from word order information, and representing the relations only with the necessary information
- The dependency relations approximate the semantic relations ⇒ the results are directly useful for coreference resolution, question answering and information extraction, etc.

II. Dependency Relations

The Universal Dependencies project provides Universal dependency relation that are linguistically motivated, computationally useful, and cross-linguistically applicable.



Relation	Description	Example (**head** -> ***dependent***)
Clausal Argument Relations		
NSUBJ	Nominal subject	***United*** **canceled** the flight.
DOBJ	Direct object	United **diverted** the ***flight*** to Reno.
IOBJ	Indirect object	We **booked** ***her*** the flight to Miami.
Nominal Modifier Relations		
NMOD	Nominal modifier	We took the ***morning*** **flight**.
AMOD	Adjectival modifier	Book the ***cheapest*** **flight**.
NUMMOD	Numeric modifier	JetBlue canceled ***1000*** **flights**.
APPOS	Appositional modifier	**United**, a ***unit*** of UAL, matched the fares.
DET	Determiner	***The*** **flight** was canceled.
CASE	Prepositions, postpositions and other case markers	Book the flight ***through*** **Houston**.
Other Notable Relations		
CONJ	Conjunct	We **flew** to Denver and ***drove*** to Steamboat.
CC	Coordinating conjunction	We **flew** to Denver ***and*** drove to Steamboat.

III. Dependency Formalisms

- Structures G = (V;A)
  - a set of **vertices (node**) V, set of words in a given sentence
  - a set of ordered pairs of vertices A, as **arcs**, captures the head dependent and grammatical function **relationships** between the elements in V
- Dependency tree is a directed graph that satisfies the following constraints:

  1. With the exception of the root node, **each vertex** has exactly **one incoming arc.**

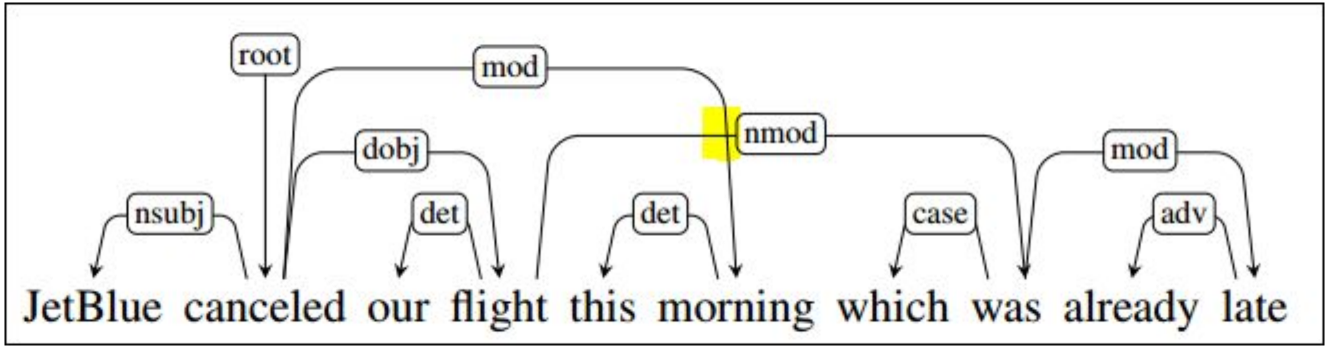
  2. There is **a unique path** from the **root node to each vertex** in V .

  3. There is a single designated **root** node that has **no incoming arcs**.

III. Dependency Formalisms - Projective

Be an **additional condition** in the order of the words in the input.

Arc is projective If there is a path from the head to every word that lies between the head and the dependent in the sentence. A dependency tree is projective if it can be drawn with **no crossing edges**


root
mod
nmod
dobj
mod
nsubj
det
det
case
adv
JetBlue canceled our flight this morning which was already late

Non-projective structures will necessarily contain some errors

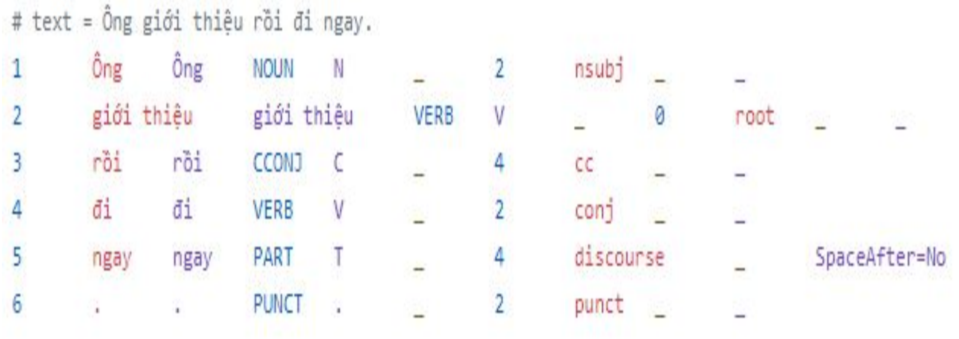
IV. Dependency TreeBanks

- A critical component in developing and evaluating dependency parsers
- Each data contains:
    - a sentence
    - the information and the dependency relations of words
- Need human labeling
- Dependency Treebanks for multiple languages can be found on The Universal Dependencies project (universaldependencies.org)
- Vietnamese Dependency Treebank can be found on Github (github.com/UniversalDependencies/UD_Vietnamese-VTB/tree/master)

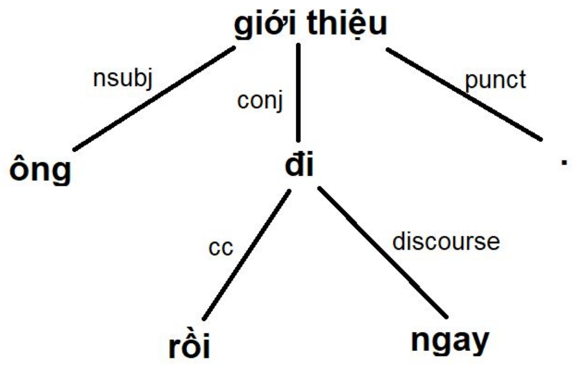
IV. Dependency TreeBanks

The treebanks are in CoNLL-U Format which comprises 10 columns:

1. ID: Word index, integer starting at 1.
2. FORM: Word form or punctuation symbol.
3. LEMMA: Lemma or stem of word form.
4. UPOS: Universal part-of-speech tag.
5. XPOS: Language-specific part-of-speech tag.
6. FEATS: List of morphological features.
7. HEAD: Head of the current word.
8. DEPREL: Universal dependency relation to the HEAD.
9. DEPS: Enhanced dependency graph in the form of a list of head-deprel pairs.
10. MISC: Any other annotation.



```
# text = Ông giới thiệu rồi đi ngay.
1       Ông     Ông     NOUN    N       _       2       nsubj   _       _
2       giới thiệu      giới thiệu      VERB    V       _       0       root    _       _
3       rồi     rồi     CCONJ   C       _       4       cc      _       _
4       đi      đi      VERB    V       _       2       conj    _       _
5       ngay    ngay    PART    T       _       4       discourse       _       SpaceAfter=No
6       .       .       PUNCT   .       _       2       punct   _       _
```

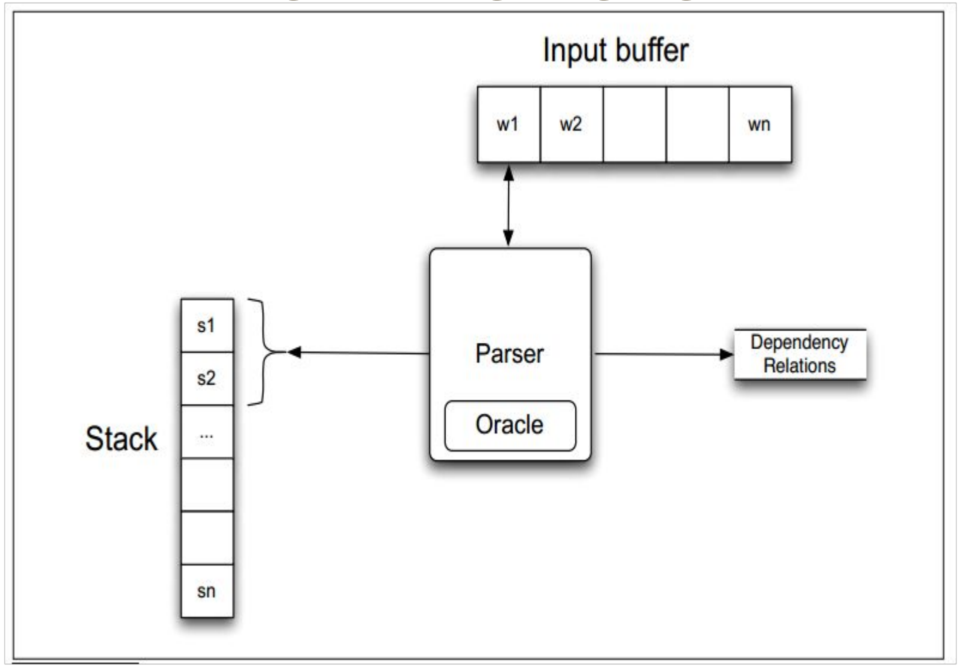


V. Transition-Based Dependency Parsing

Shift-reduce parsing (Aho and Ullman, 1972) Programming language -> NLP

Configuration consists of a stack, an input buffer of words, or tokens, and a set of relations representing a dependency tree.

In the final goal state, the stack and the word list should be empty, and the set of relations will represent the final parse. Root is the only element remaining on the stack.



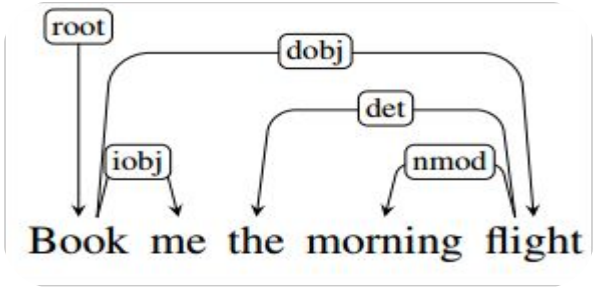
V. Transition-Based Dependency Parsing

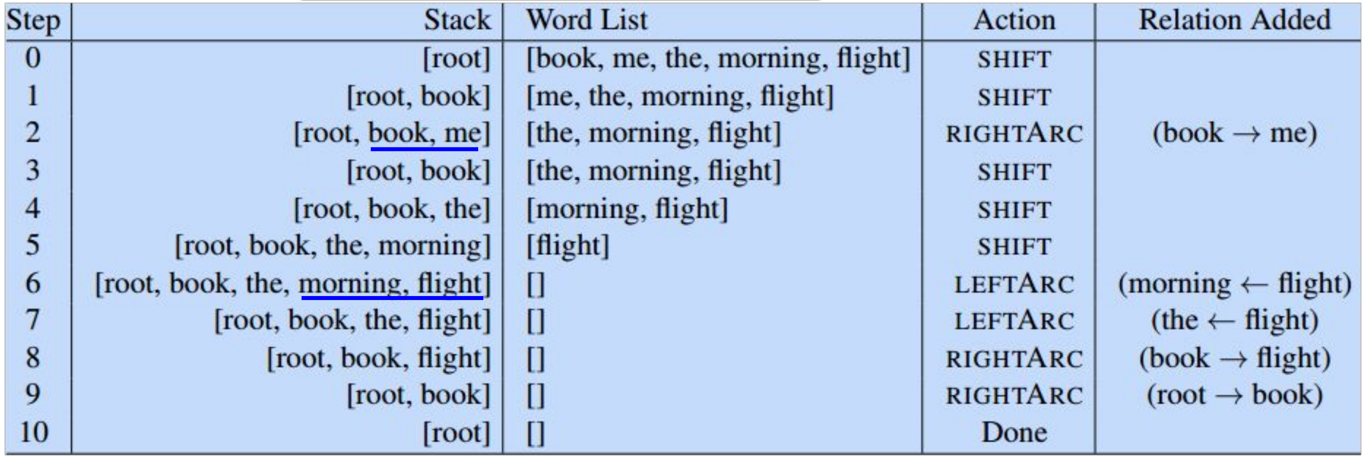
Transition operators:

• *LEFTARC*: Assert a head-dependent relation between the word at the top of the stack and the word directly beneath it; **remove the lower word from the stack.**

• *RIGHTARC*: Assert a head-dependent relation between the second word on the stack and the word at the top; **remove** the word at the **top of the stack;**

• *SHIFT*: Remove the word from the front of the input buffer and push it onto the stack.

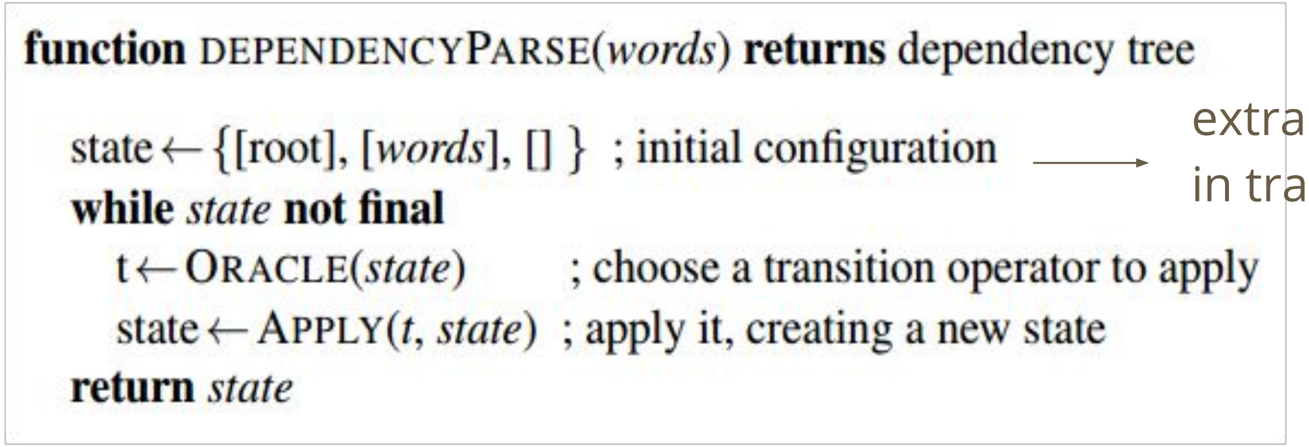




Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning ← flight)
7	[root, book, the, flight]	[]	LEFTARC	(the ← flight)
8	[root, book, flight]	[]	RIGHTARC	(book → flight)
9	[root, book]	[]	RIGHTARC	(root → book)
10	[root]	[]	Done	

V. Transition-Based Dependency Parsing-Creating oracle

As with all supervised machine learning methods (Logistic regression, SVM, neutral network, deep learning)



```
function DEPENDENCYPARSE(words) returns dependency tree

  state ← {[root], [words], [] }  ; initial configuration
  while state not final
     t ← ORACLE(state)          ; choose a transition operator to apply
     state ← APPLY(t, state)  ; apply it, creating a new state
  return state
```

→ extract useful features in training data

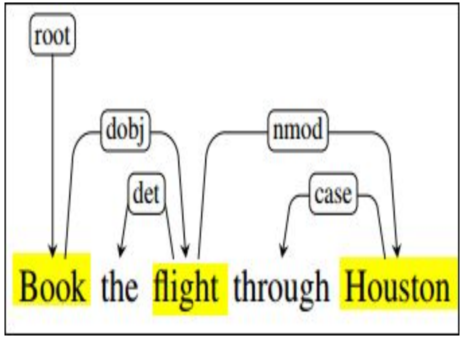
V. Transition-Based Dependency Parsing-Generating training data

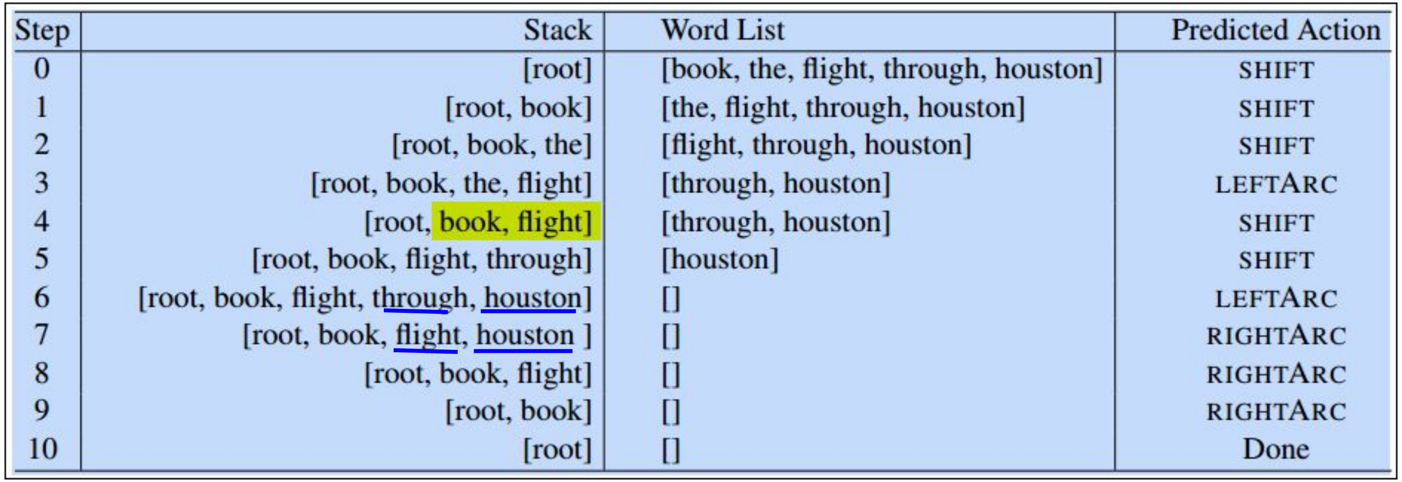
The training oracle proceeds as follows:

• Choose LEFTARC if it produces a correct head-dependent relation given the reference parse and the current configuration,

• Otherwise, choose RIGHTARC if (1) it produces a correct head-dependent relation given the reference parse **and (2) all of the dependents of the word at the top of the stack have already been assigned,**

• Otherwise, choose SHIFT.





Step	Stack	Word List	Predicted Action
0	[root]	[book, the, flight, through, houston]	SHIFT
1	[root, book]	[the, flight, through, houston]	SHIFT
2	[root, book, the]	[flight, through, houston]	SHIFT
3	[root, book, the, flight]	[through, houston]	LEFTARC
4	[root, book, flight]	[through, houston]	SHIFT
5	[root, book, flight, through]	[houston]	SHIFT
6	[root, book, flight, through, houston]	[]	LEFTARC
7	[root, book, flight, houston]	[]	RIGHTARC
8	[root, book, flight]	[]	RIGHTARC
9	[root, book]	[]	RIGHTARC
10	[root]	[]	Done

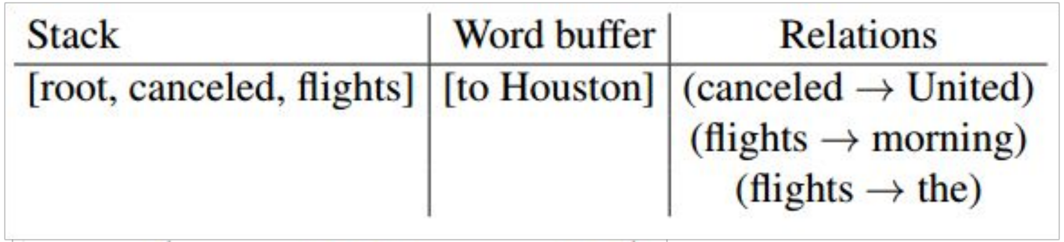
V. Transition-Based Dependency Parsing- Features

Configuration-Transition pairs => Features-Transition pairs

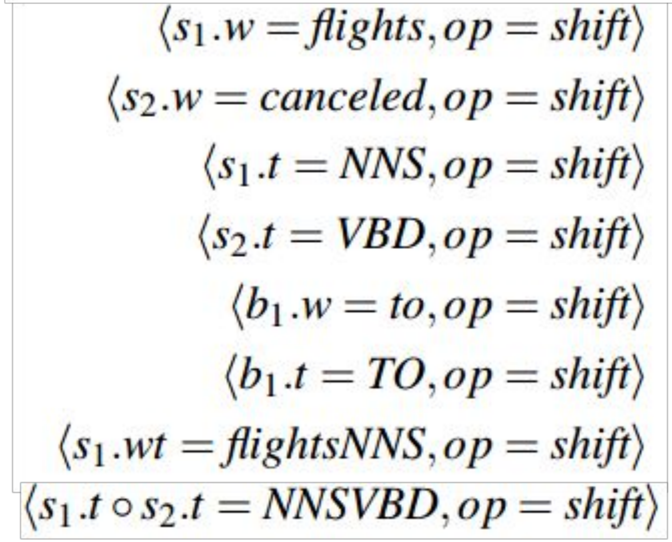
The focus of feature extraction:

- Word forms, lemmas and parts of speech are all powerful features, as are the head, and dependency relation to the head
- On the **top** levels of the stack, the words near the **front** of the buffer, and the dependency relations already associated with any of those elements.

V. Transition-Based Dependency Parsing- Features

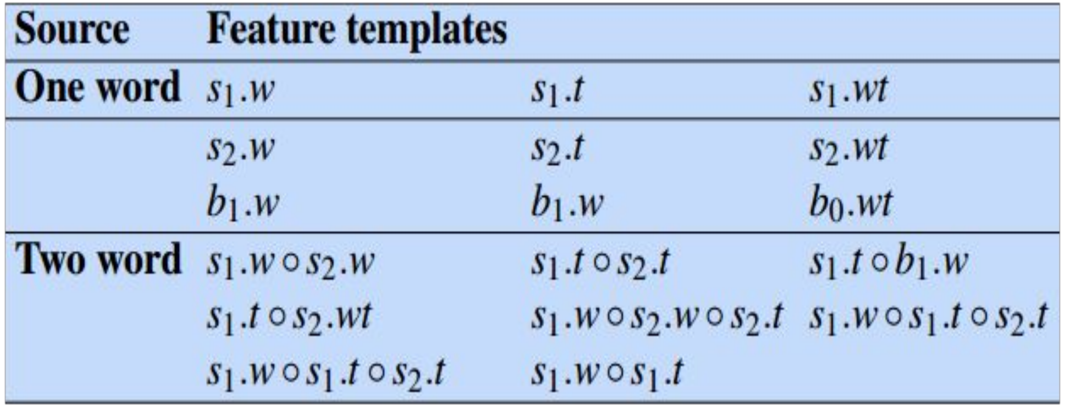


Stack	Word buffer	Relations
[root, canceled, flights]	[to Houston]	(canceled $\rightarrow$ United)
		(flights $\rightarrow$ morning)
		(flights $\rightarrow$ the)



$$\langle s_1.w = flights, op = shift \rangle$$
$$\langle s_2.w = canceled, op = shift \rangle$$
$$\langle s_1.t = NNS, op = shift \rangle$$
$$\langle s_2.t = VBD, op = shift \rangle$$
$$\langle b_1.w = to, op = shift \rangle$$
$$\langle b_1.t = TO, op = shift \rangle$$
$$\langle s_1.wt = flightsNNS, op = shift \rangle$$
$$\langle s_1.t \circ s_2.t = NNSVBD, op = shift \rangle$$

- s denotes the stack,
- b the word buffer
- r the set of relations.
- w for word forms,
- l for lemmas
- t for part-of-speech

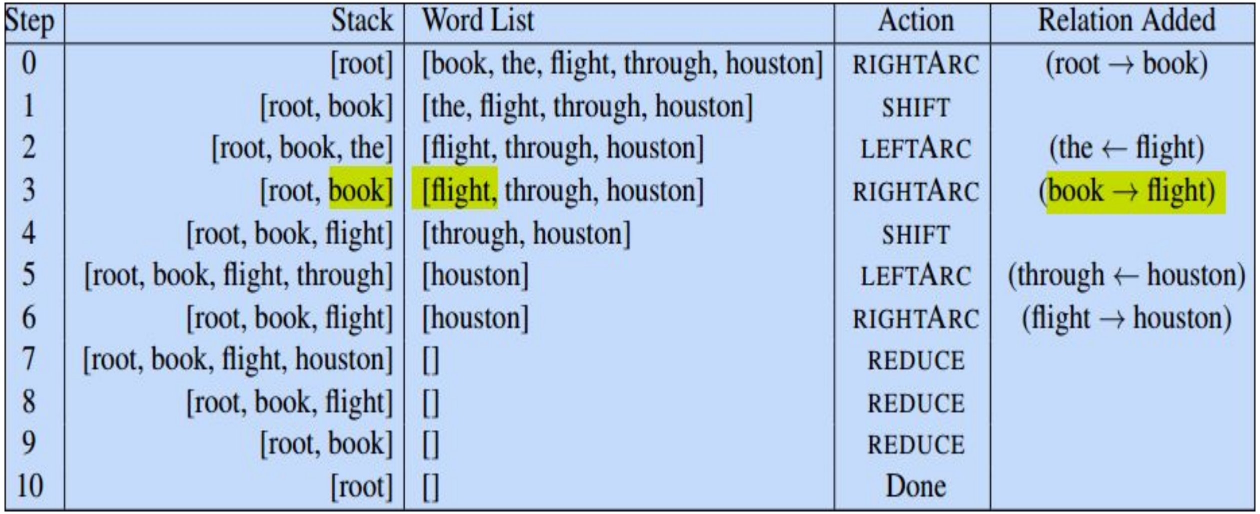


Source	Feature templates		
One word	$s_1.w$	$s_1.t$	$s_1.wt$
	$s_2.w$	$s_2.t$	$s_2.wt$
	$b_1.w$	$b_1.w$	$b_0.wt$
Two word	$s_1.w \circ s_2.w$	$s_1.t \circ s_2.t$	$s_1.t \circ b_1.w$
	$s_1.t \circ s_2.wt$	$s_1.w \circ s_2.w \circ s_2.t$	$s_1.w \circ s_1.t \circ s_2.t$
	$s_1.w \circ s_1.t \circ s_2.t$	$s_1.w \circ s_1.t$	

V. Transition-Based Dependency Parsing- Advanced Methods in Transition

Alternative Transition Systems (arc eager): assert rightward relations much sooner than in the arc standard approach

- LEFTARC: Assert a head-dependent relation between the word at the front of the input buffer and the word at the top of the stack; pop the stack.
- RIGHTARC: Assert a head-dependent relation between the word on the top of the stack and the word at front of the input buffer; shift the word at the front of the input buffer to the stack.
- SHIFT: Remove the word from the front of the input buffer and push it onto the stack.
- REDUCE: Pop the stack



Step	Stack	Word List	Action	Relation Added
0	[root]	[book, the, flight, through, houston]	RIGHTARC	(root $\rightarrow$ book)
1	[root, book]	[the, flight, through, houston]	SHIFT	
2	[root, book, the]	[flight, through, houston]	LEFTARC	(the $\leftarrow$ flight)
3	[root, book]	[flight, through, houston]	RIGHTARC	(book $\rightarrow$ flight)
4	[root, book, flight]	[through, houston]	SHIFT	
5	[root, book, flight, through]	[houston]	LEFTARC	(through $\leftarrow$ houston)
6	[root, book, flight]	[houston]	RIGHTARC	(flight $\rightarrow$ houston)
7	[root, book, flight, houston]	[]	REDUCE	
8	[root, book, flight]	[]	REDUCE	
9	[root, book]	[]	REDUCE	
10	[root]	[]	Done	

V. Transition-Based Dependency Parsing- Advanced Methods in Transition

Beam Search: systematically explore multiple decision sequences

- by combining breadth-first search approach with heuristic filtering
- the maximum number of sequences considered at a time is defined by the beam width

At each time step:

- apply all possible operators to each configuration in the agenda
- score all configurations and keep only “beam width” best ones

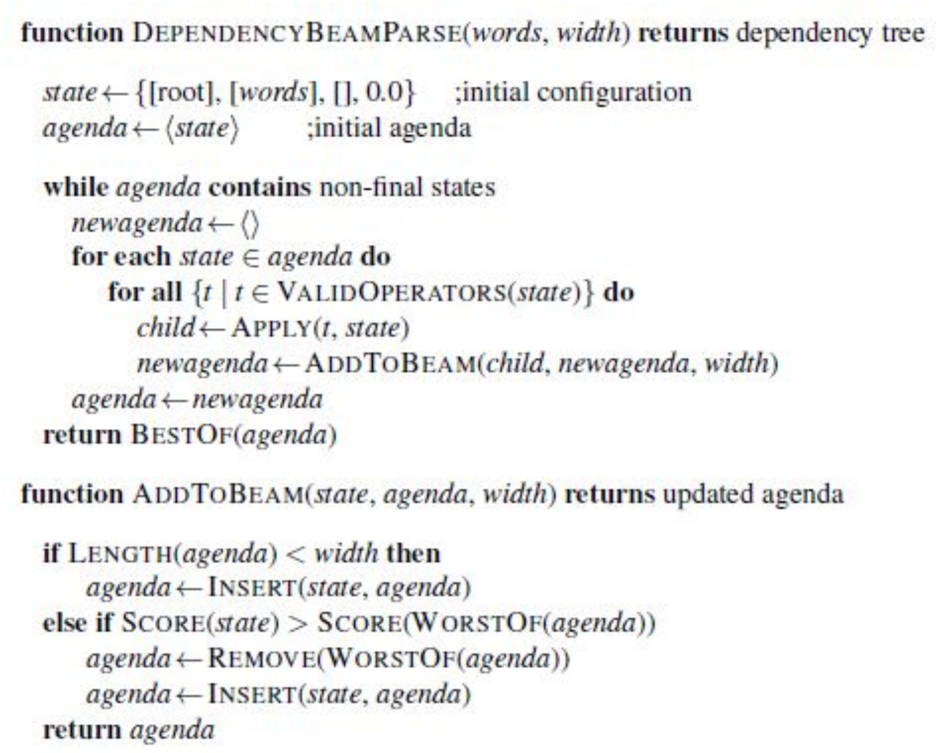
$$\begin{aligned} ConfigScore(c_0) &= 0.0 \\ ConfigScore(c_i) &= ConfigScore(c_{i-1}) + Score(t_i, c_{i-1}) \end{aligned}$$

V. Transition-Based Dependency Parsing- Advanced Methods in Transition

Beam search is completed when all considered configurations in the Beam are in final state

The final result is the best scored tree

$$\hat{T}(c) = argmaxScore(t, c)$$



```
function DEPENDENCYBEAMPARSE(words, width) returns dependency tree

  state ← {[root], [words], [], 0.0}     ;initial configuration
  agenda ← ⟨state⟩        ;initial agenda

  while agenda contains non-final states
    newagenda ← ⟨⟩
    for each state ∈ agenda do
      for all {t | t ∈ VALIDOPERATORS(state)} do
        child ← APPLY(t, state)
        newagenda ← ADDTOBEAM(child, newagenda, width)
    agenda ← newagenda
  return BESTOF(agenda)

function ADDTOBEAM(state, agenda, width) returns updated agenda

  if LENGTH(agenda) < width then
     agenda ← INSERT(state, agenda)
  else if SCORE(state) > SCORE(WORSTOF(agenda))
     agenda ← REMOVE(WORSTOF(agenda))
     agenda ← INSERT(state, agenda)
  return agenda
```

VI. Graph-Based Dependency Parsing

- Search through possible trees for a tree that maximize some score:

$$\hat{T}(S) = \underset{t \in \mathcal{G}_S}{\operatorname{argmax}}\, score(t, S)$$

- The score of a tree is determined by the scores of the edges:

$$score(t, S) = \sum_{e \in t} score(e)$$

- Motivations for using Graph-based approach:
    - Capable of producing non-projective trees
    - Improving parsing accuracy, with respect to longer dependencies

VI. Graph-Based Dependency Parsing

The process:

1. Given a sentence, build a fully-connected, weighted, directed graph:
    - Vertices are words
    - Directed edges are all possible head-dependent relations
    - A ROOT node with outgoing edges to all other vertices
    - The weights of edges are scored by a model
2. Apply maximum spanning tree algorithm to find the best tree

*Note: This algorithm doesn't necessarily produces a tree as the graph can contain cycles. So we need recursive cleanup phase to remove cycles.

VI. Graph-Based Dependency Parsing

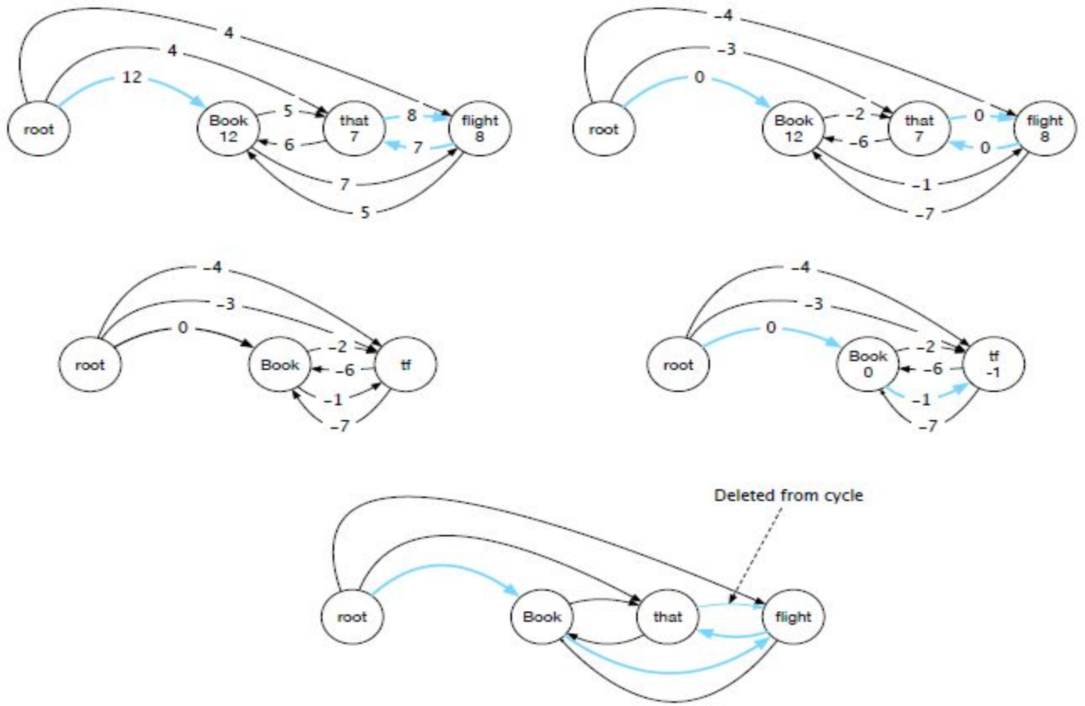
```
function MaxSpanningTree($G=(V,E)$, $root$, $score$) returns spanning tree

    $F \leftarrow []$
    $T' \leftarrow []$
    $score' \leftarrow []$
    for each $v \in V$ do
        $bestInEdge \leftarrow \operatorname{argmax}_{e=(u,v) \in E}\ score[e]$
        $F \leftarrow F \cup bestInEdge$
        for each $e=(u,v) \in E$ do
            $score'[e] \leftarrow score[e] - score[bestInEdge]$

        if $T=(V,F)$ is a spanning tree then return it
        else
            $C \leftarrow$ a cycle in $F$
            $G' \leftarrow$ Contract($G, C$)
            $T' \leftarrow$ MaxSpanningTree($G', root, score'$)
            $T \leftarrow$ Expand($T', C$)
            return $T$

function Contract($G, C$) returns contracted graph

function Expand($T, C$) returns expanded graph
```



VI. Graph-Based Dependency Parsing

For the scoring model, we aim to find parameters w so that when applied to features f of edge will produce the edge's score:

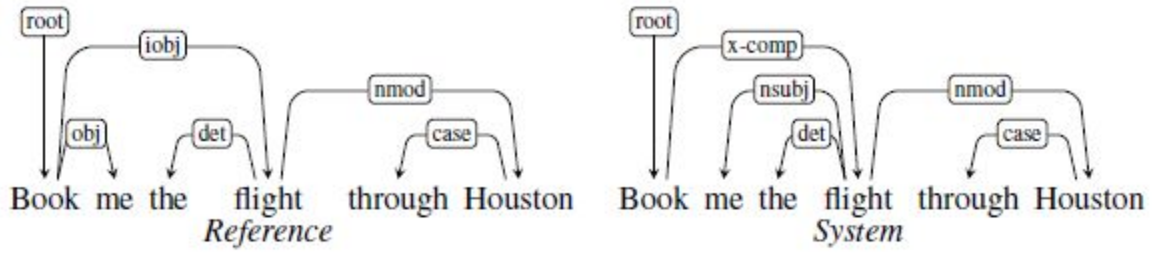
$$score(S,e) = w \cdot f$$

such that scores are higher for correct tree than to incorrect ones.

- The features extracted from edge can be words' forms, lemmas, POSs, direction of relation, head-dependent distance, etc.
- The framework for finding w is inference-based learning combined with perceptron learning rule (State-of-the-art algorithms are RNN based)

VII. Evaluation

- Evaluating the results of dependency parsers to the reference parses in test set.
- Common metrics:
  - Labeled attachment score (LAS) = % of correct directed edges with correct labels
  - Unlabeled attachment score (UAS) = % of correct directed edge regardless of labels
  - Label accuracy score (LS) = % of dependent tokens with correct labels
- Example with LAS=67% and UAS=83%


root
iobj
obj
det
nmod
case
Book me the flight through Houston
Reference
root
x-comp
nsubj
det
nmod
case
Book me the flight through Houston
System

Summary

- Dependency parsing focus on dependency relations between words
- The relations are shown as directed labeled edges (head -> dependent)
- Useful input for coreference resolution, semantic parsing, etc.
- Transition-based parsing use a greedy stack-based algorithm
- Graph-based parsing use maximum spanning tree algorithm
- Both approaches need supervised machine learning model
- Treebanks are the data source for training the model
- Evaluation use metrics such as LAS, UAS, LS

Thank you for listening!