Bài 6: Phân tích cú pháp phụ thuộc

Thông tin giảng viên

- Phd Nguyen Kiem Hieu
- Computer science department, School of Information and Communication Technology, HUST
- Email: hieunk@soict.hust.edu.vn

Content

1. **Overview**
    - Introduction
    - Applications
    - Properties
2. Approaches
    - Transition-based
    - Graph-based
    - Current approaches
3. Some results

Introduction

- Increasing interest in dependency-based approaches to syntactic parsing in recent years
- Dependency-based methods still less accessible for the majority of researchers and developers than the more widely known constituency-based methods

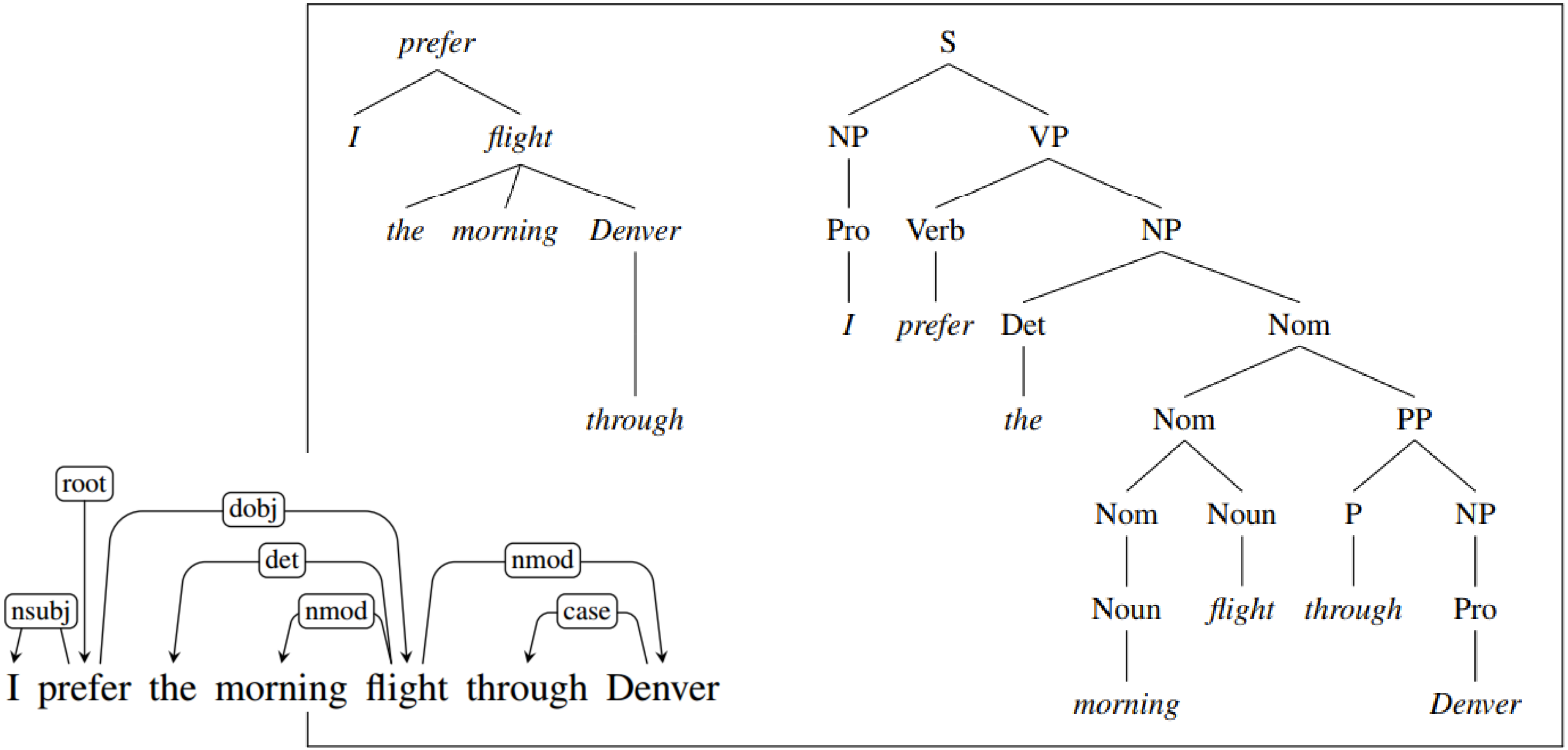
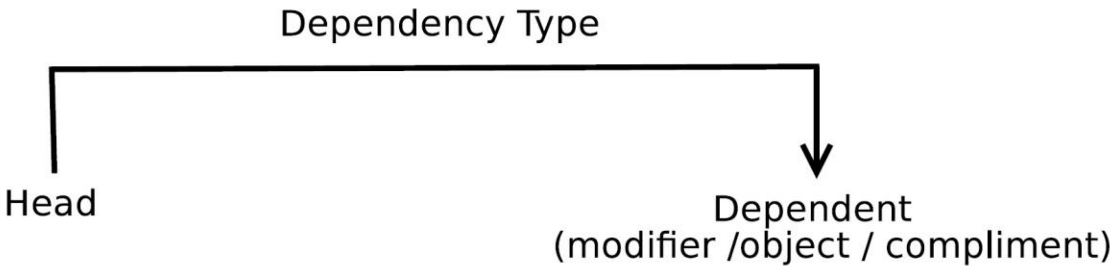


Figure 14.1 Dependency and constituent analyses for *I prefer the morning flight through Denver.*

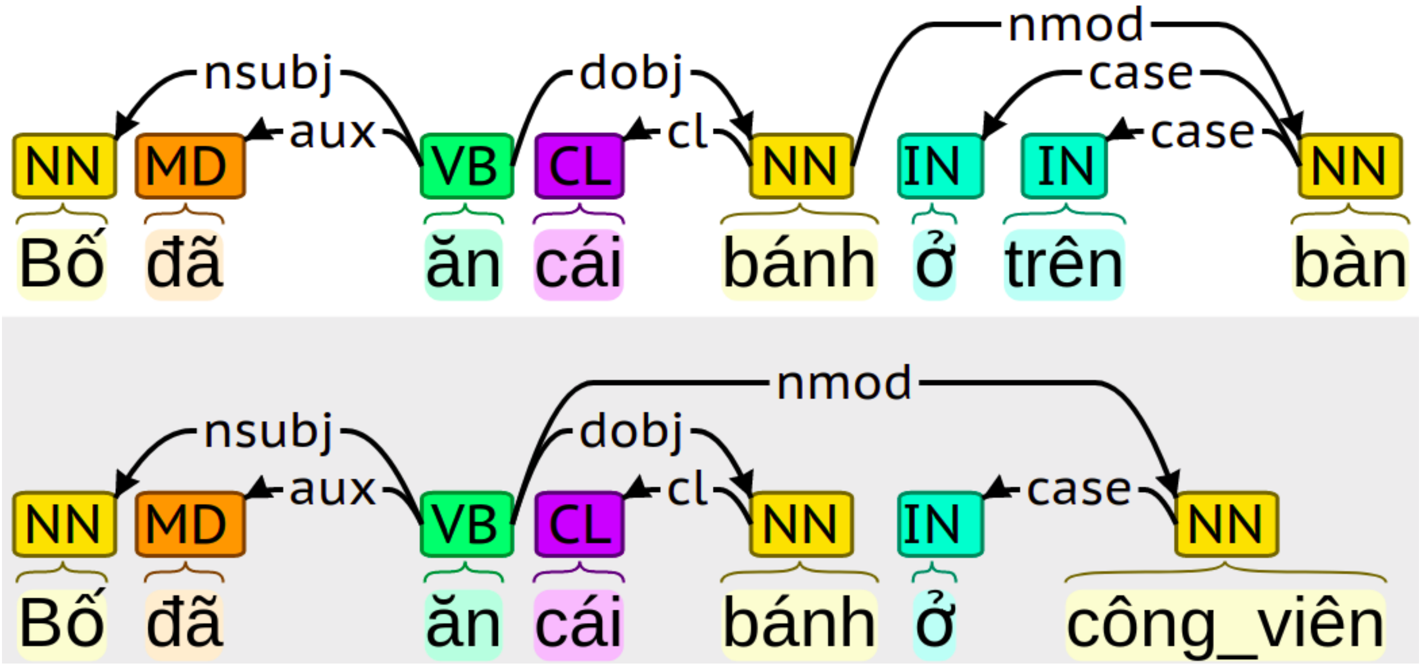
Dan Jurafsky and James Martin. *Speech and Language Processing.* PrenticeHall (3rd draft)

Dependency Grammars

- Syntactic structure = lexical items linked by binary asymmetrical relations called dependencies

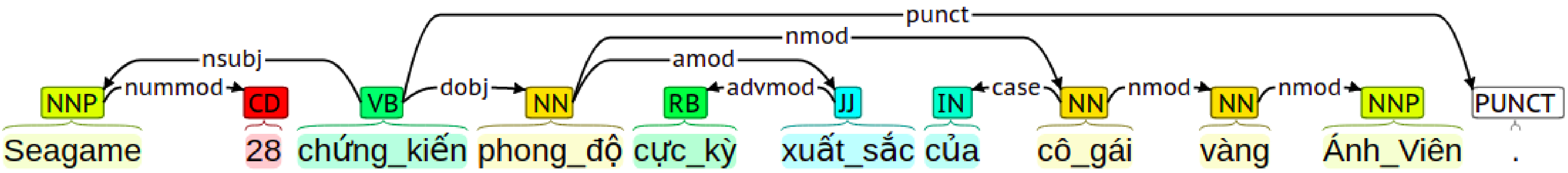


Example Dependency Parse



Some dependency labels

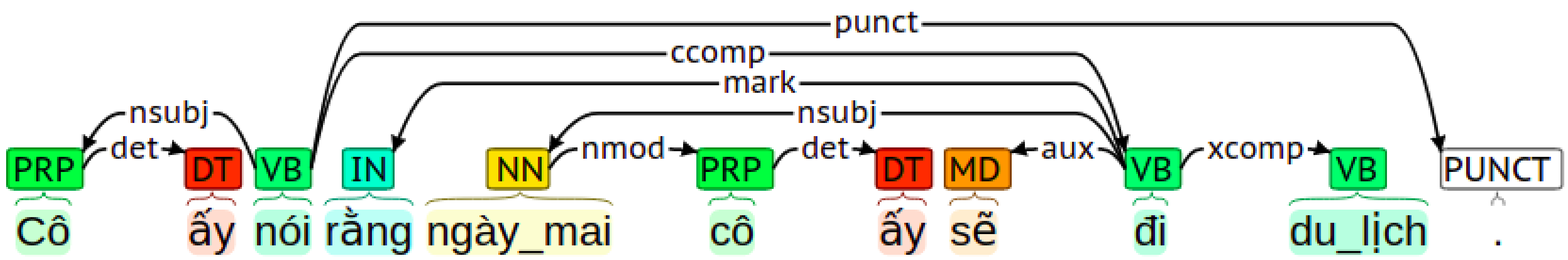
- nsubj (Nominal subject): chủ ngữ, chủ thể
- dobj (Direct object): tân ngữ trực tiếp
- nmod (Nominal modifier): danh từ bổ nghĩa
- amod (Adjectival modifier): tính từ bổ nghĩa
- nummod (Numeric modifier): số từ bổ nghĩa
- case (dependent of the noun they attach to or introduce)



Some dependency labels

- ccomp (Clausal component): Mệnh đề thành phần
- xcomp (Open clausal component): Mệnh đề thành phần mở rộng
- aux (Auxiliary): phụ từ, trợ động từ

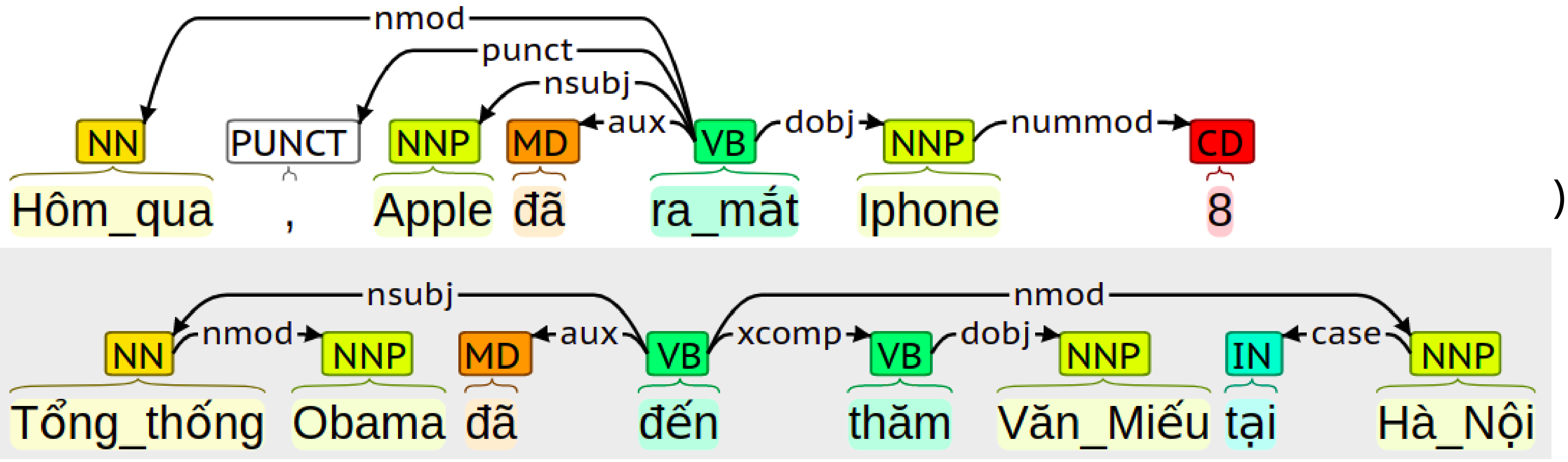
See more: http://universaldependencies.org/u/dep/



Applications

Building a knowledge base using relation extraction

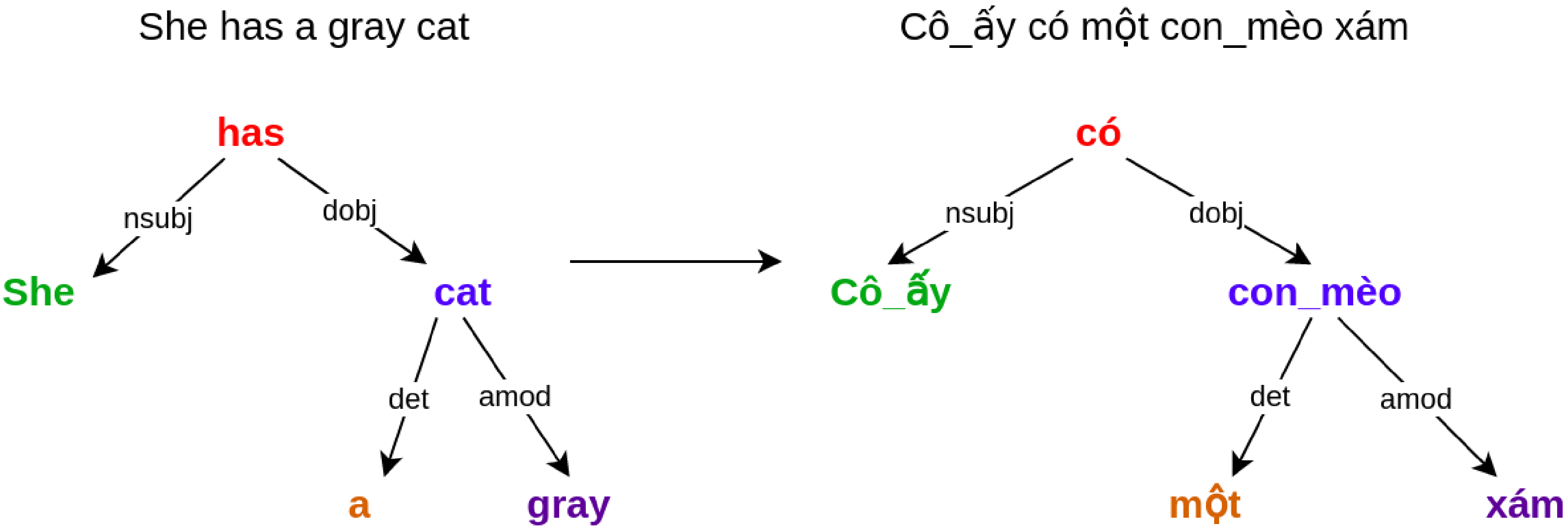
(Ap



)

Applications

Machine Translation

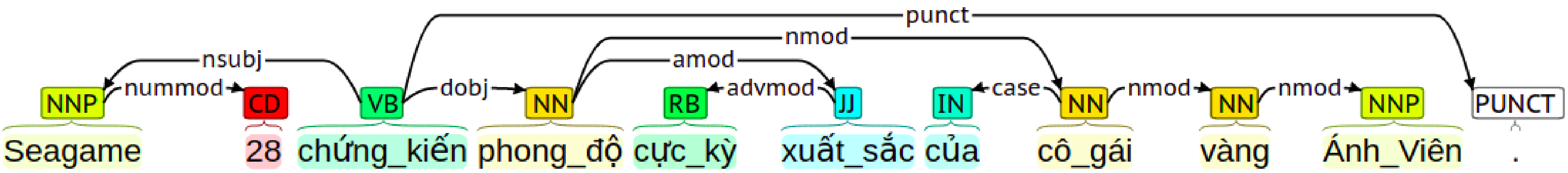


Properties

- General form: a graph $G = (V,A)$
  - V vertices: usually one per word in sentence
  - A arcs (set of ordered pairs of vertices): head-dependent relations between elements in V
- Notational conventions ($i , j \in V$):
  - $i \rightarrow j \quad \equiv \quad (i , j) \in E$
  - $i \rightarrow* j \quad \equiv \quad i = j \vee \exists k : i \rightarrow k, k \rightarrow* j$

Properties

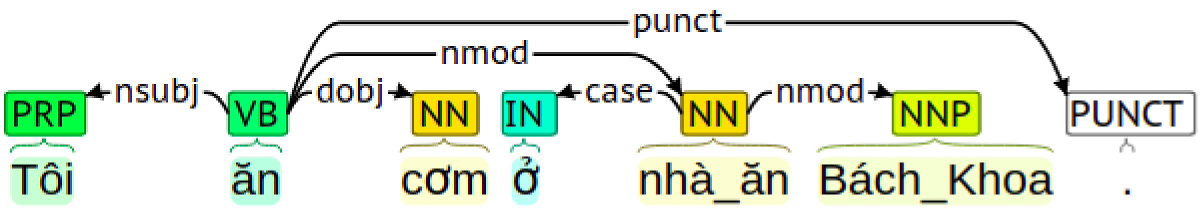
- Weakly Connected
  - For every node *i* there is a node *j* such that $i \to j$ or $j \to i$.
- Acyclic:
  - If $i \to j$ then not $j \to^* i$.
- Single head:
  - If $i \to j$, then not $k \to j$, for any $k \neq i$.



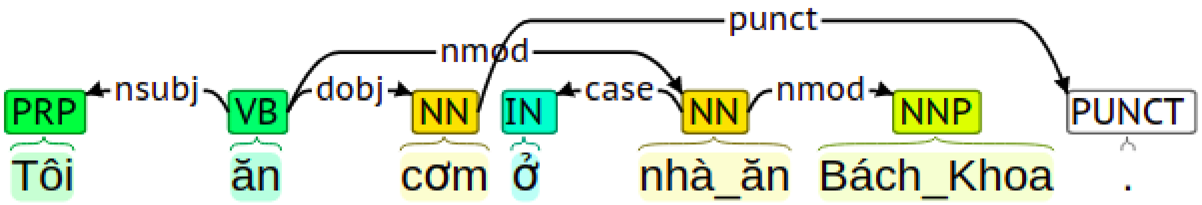
Properties

- Projective: There are no crossing dependencies

Projective



Non-Projective



Content

1. Overview
    - Introduction
    - Applications
    - Properties
2. **Approaches**
    - Transition-based
    - Graph-based
    - Current approaches
3. Some results

Approaches

- **Transition-based**
  - Nivre algorithm
- Graph-based
- Current approaches
  - End to end learning
  - Joint learning

Transition-based

- Main idea is to base on Transitions (SHIFT, REDUCE, LEFT-ARC, RIGHT-ARC)
- When reading a sentence from left to right, the learning model will decide which transition to perform. This sequence of transitions helps to determine the dependency relationship between the words in the sentence.

➢ Need training this model

Transition-based

- Parsing algorithm: Nivre, Covington, ...
- Classifying method: SVM, Neural network, ...

Nivre algorithm

- Given: c = (**Σ**|s, b|**B**, **A**), in which
  - Stack **Σ** stores partially processed tokens
  - Buffer **B** stores unread tokens.
  - Set **A** stores dependent relations being found

- Transition bases on the current configuration to go to the new configuration, also including these 3 members

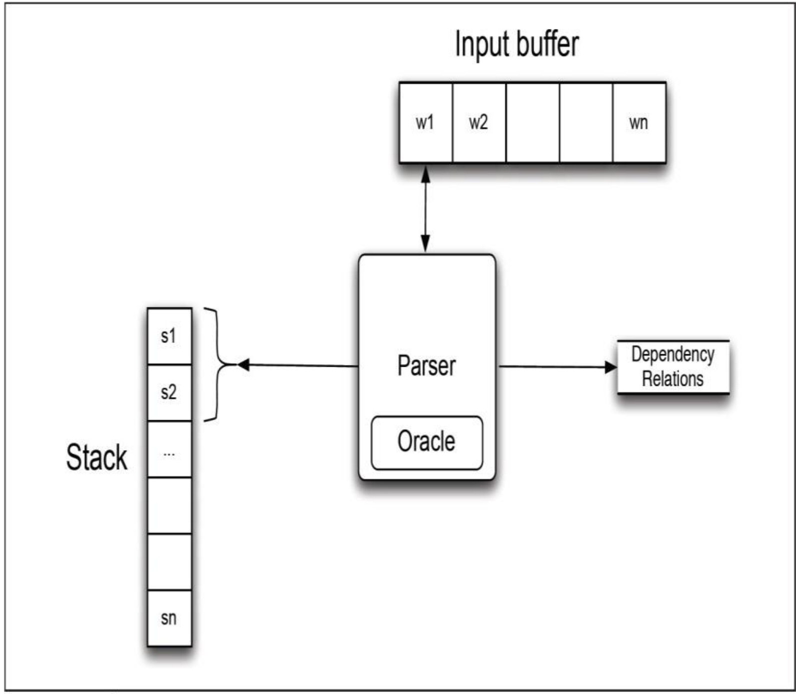


Figure 14.5 Basic transition-based parser. The parser examines the top two elements of the stack and selects an action based on consulting an oracle that examines the current configuration.

Nivre algorithm

- 4 transition:
  - SHIFT [(**Σ**, b|**B**, **A**)] = (**Σ**|b, **B**, **A**)
  - RIGHT_{lb} [(**Σ**|s, b|**B**, **A**)] = (**Σ**|s|b, **B**, **A** ∪ {s, lb, b})
  - LEFT_{lb} [(**Σ**|s, b|**B**, **A**)] = (**Σ**, b|**B**, **A** ∪ {b, lb, s})
  - REDUCE [(**Σ**|s, **B**, **A**)] = (**Σ**, **B**, **A**)
- Description:
  - **SHIFT**: Remove the top word of the buffer and push it onto the stack.
  - **RIGHT**: Insert the top word of the buffer to the stack, add relation (s, lb, b) to A
  - **LEFT**: pop the stack, add relation (b, lb, s) to A
  - **REDUCE**: Pop the stack

Example

[root]$_S$ [Economic news had little effect on financial markets .]$_Q$

Example

[root Economic]$_S$ [news had little effect on financial markets .]$_Q$

Shift

Example

nmod

[root]$_S$ Economic [news had little effect on financial

markets .]$_Q$ Left-Arc$_{nmod}$

Example

nmod

[root Economic news]$_S$ [had little effect on financial markets .]$_Q$

Shift

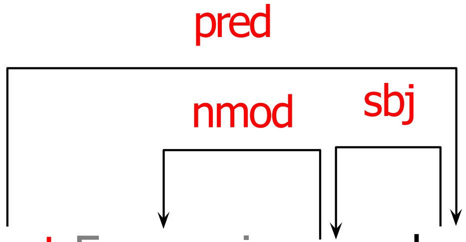
Example

nmod sbj

[root]$_S$ Economic news [had little effect on financial markets .]$_Q$

Left-Arc$_{sbj}$

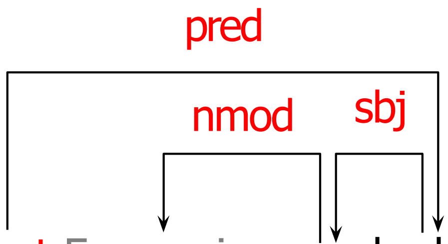
Example



[root Economic news had]$_S$ [little effect on financial markets .]$_Q$

Right-Arc$_{pred}$

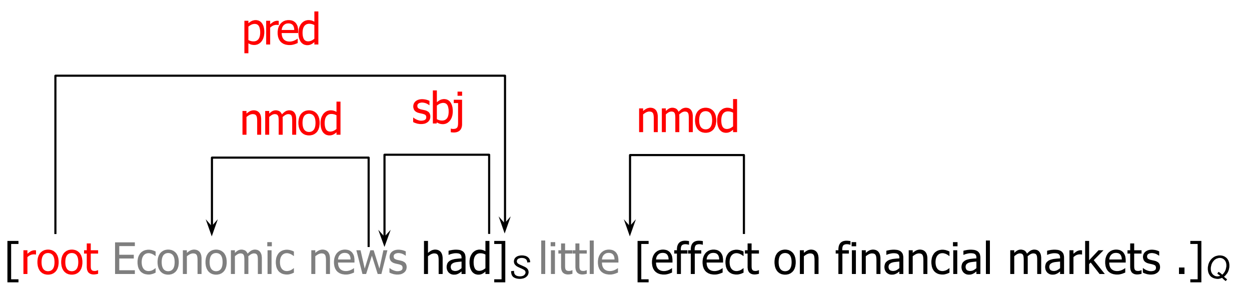
Example



[root Economic news had little]$_S$ [effect on financial markets .]$_Q$

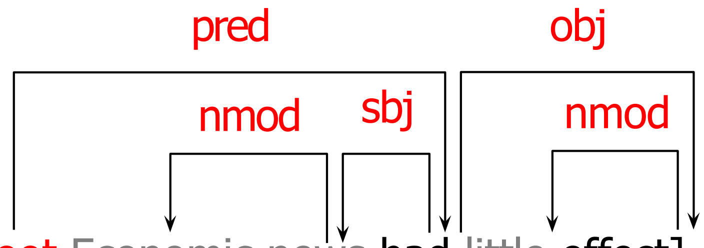
Shift

Example



Left-Arc$_{nmod}$

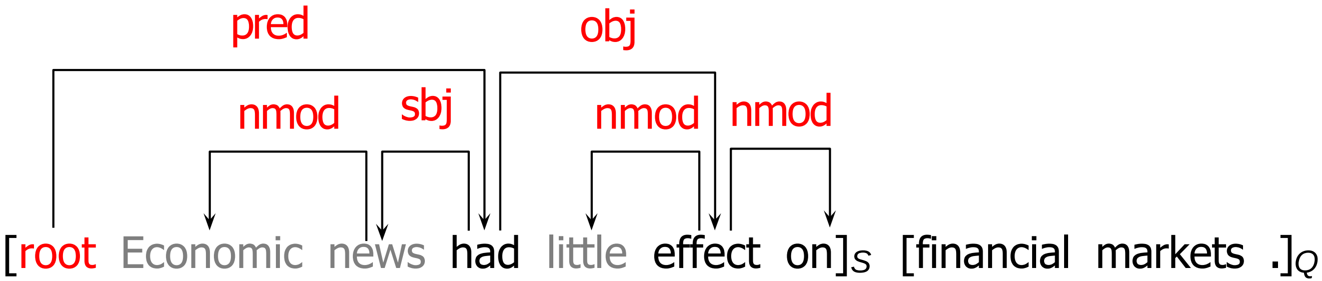
Example



[root Economic news had little effect]$_S$ [on financial markets .]$_Q$

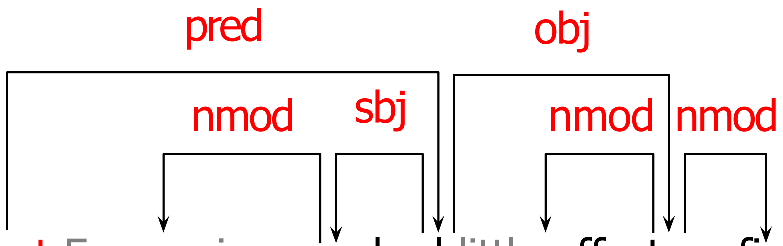
Right-Arc$_{obj}$

Example



Right-Arc$_{nmod}$

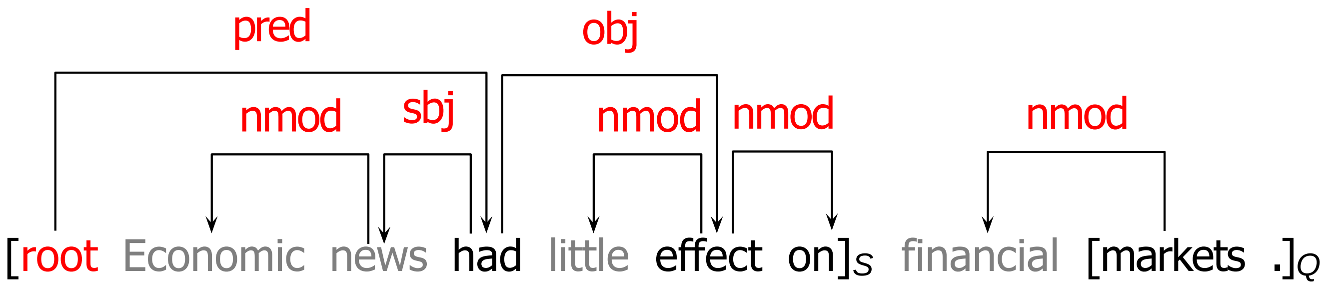
Example



[root Economic news had little effect on financial]$_S$ [markets .]$_Q$

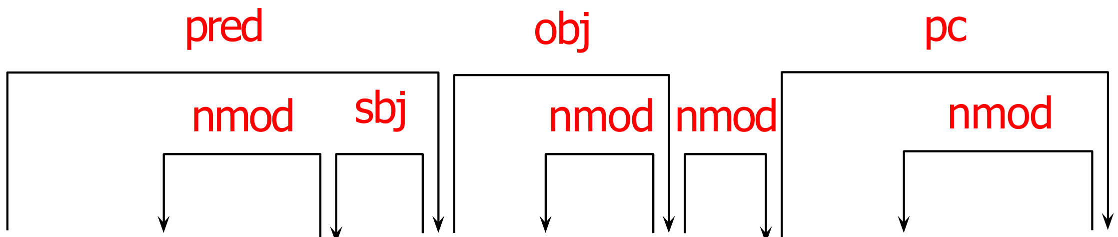
Shift

Example



Left-Arc$_{nmod}$

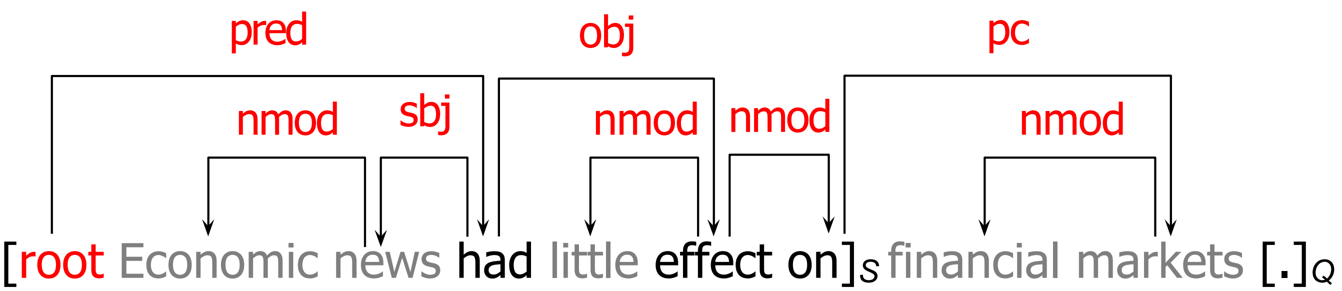
Example



[root Economic news had little effect on financial markets]$_S$ [.]$_Q$

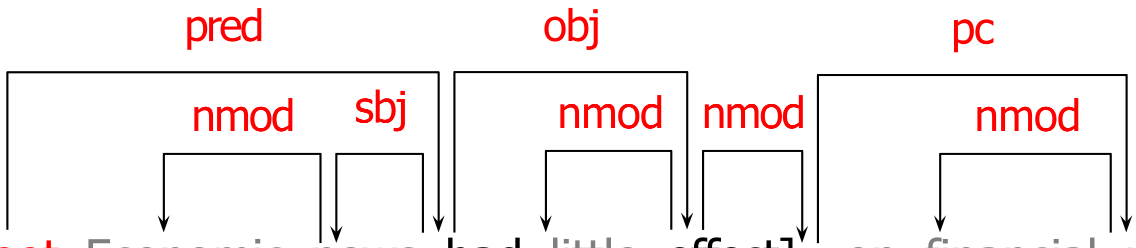
Right-Arc$_{pc}$

Example



Reduce

Example

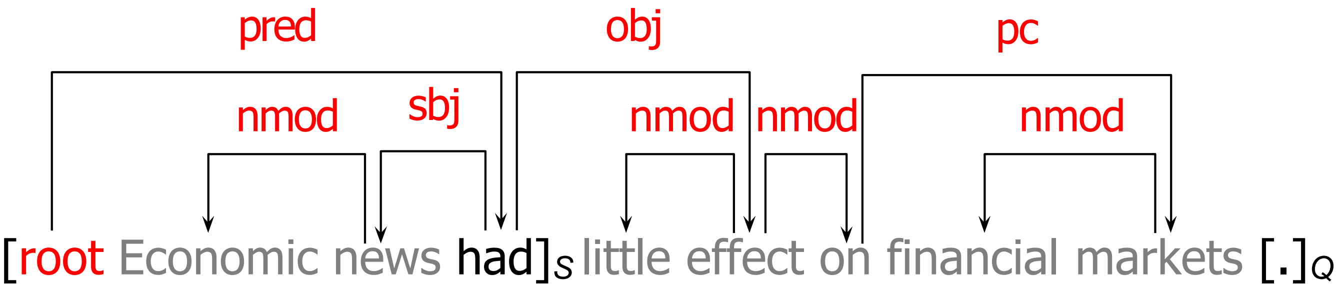


[root Economic news had little effect]$_S$ on financial markets

[.]$_Q$

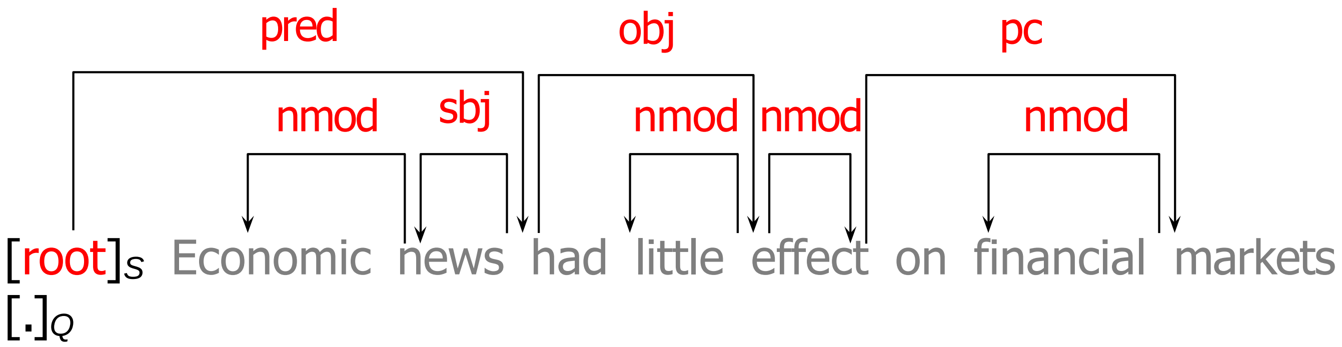
Reduce

Example



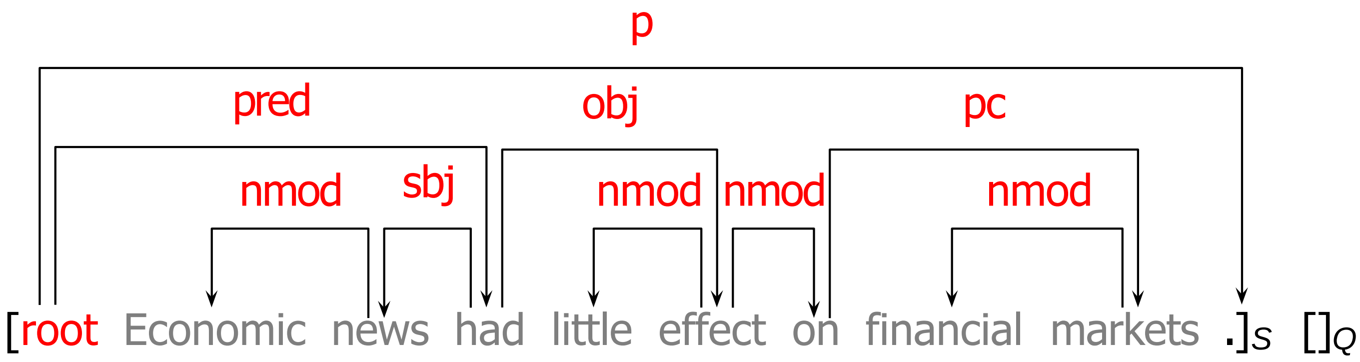
Reduce

Example



Reduce

Example

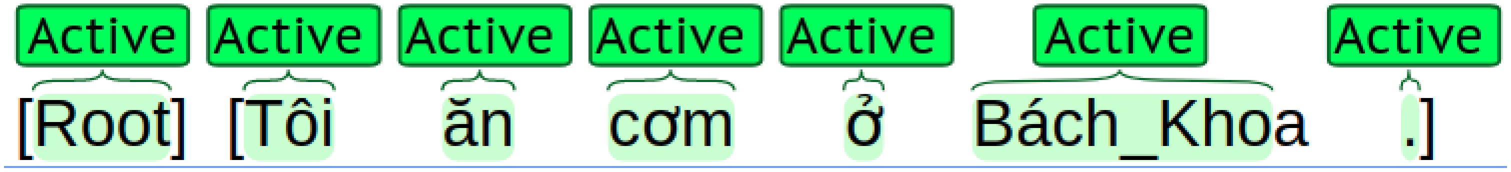


Right-Arc$_p$

Nivre algorithm

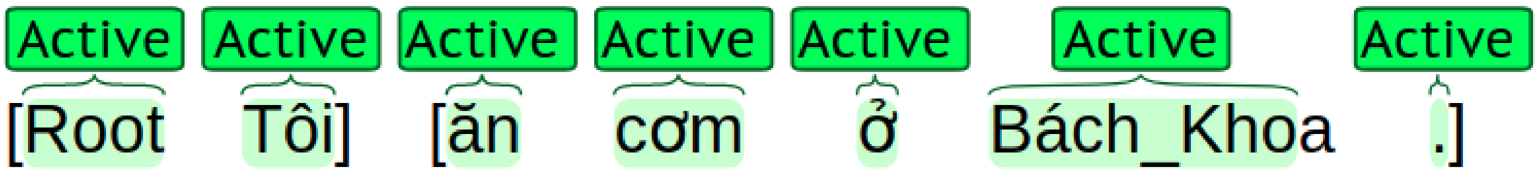
- Input sentence W = $w_1, w_2, \ldots, w_n$. (w_i is the word i^{th} in the sentence)
- Initial configuration: $c_{init} = (\boldsymbol{\Sigma}, \mathbf{B}, \mathbf{A})$
  - $\boldsymbol{\Sigma}$ = {ROOT}
  - **B**: **B** = $w_1, w_2, \ldots, w_n$
  - **A**: {}
- Terminal configuration: $c_{terminal} = (\boldsymbol{\Sigma}, \mathbf{B}, \mathbf{A})$
  - $\boldsymbol{\Sigma}$: {ROOT}
  - **B**: {}
  - **A**: set of dependent relations.

Example



- Input sentence: **Tôi ăn cơm ở Bách_Khoa .**
- Stack: [
- Buffer:]
- A : {}
- Active: the node is being considered
- Deleted: the node is completely visited, remove from Stack

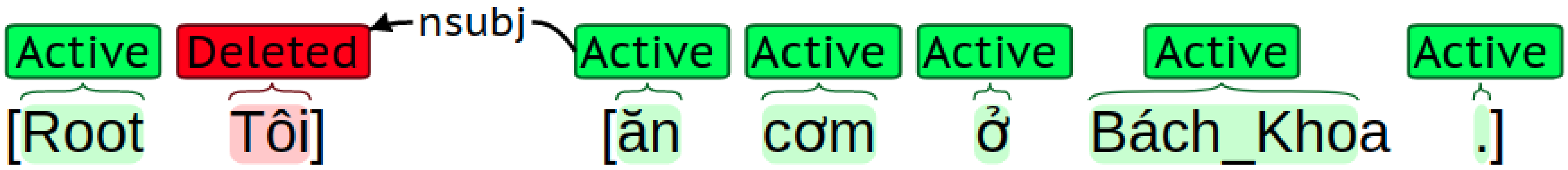
Example



SHIFT: move 'Tôi' from Buffer to Stack

A = {}

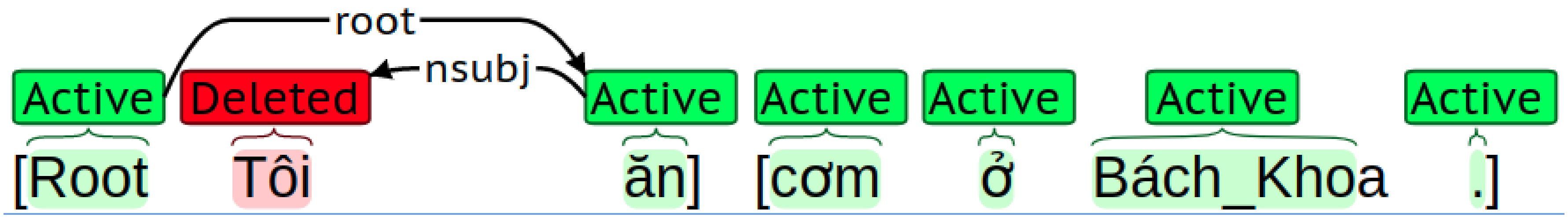
Example



LEFT$_{nsubj}$: Delete 'Tôi' from Stack, add (ăn, nsubj, Tôi) into A

A= {(ăn, nsubj, Tôi)}

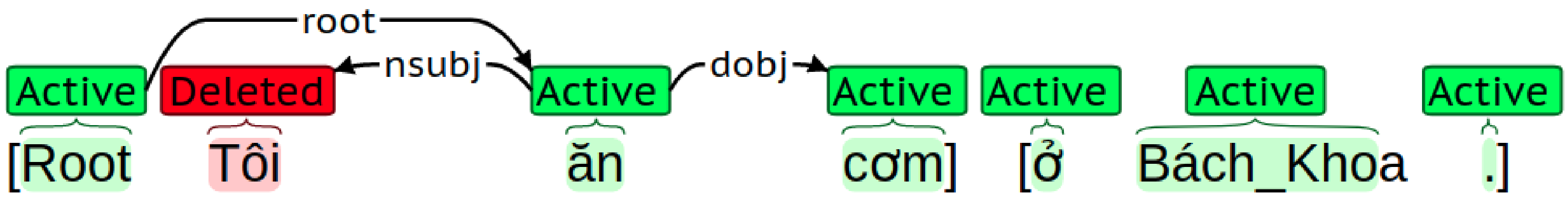
Example



RIGHT$_{root}$: Add 'ăn' from bufer to stack, add (Root, root, ăn) to A

A= {(ăn, nsubj, Tôi), (Root, root, ăn)}

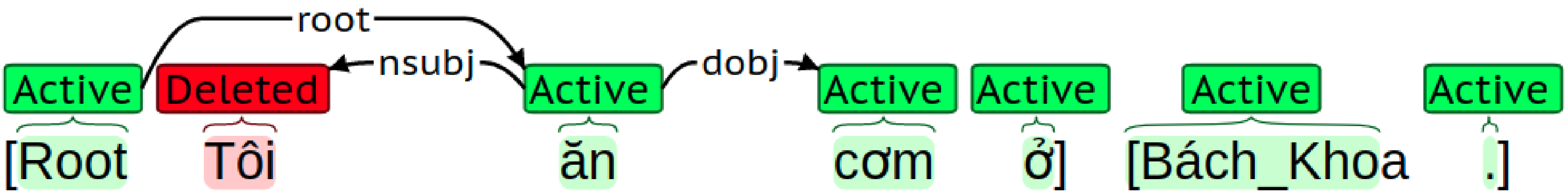
Example



RIGHT$_{dobj}$: Add 'cơm' from buffer to stack, add (ăn, dobj, cơm) to A

A= {(ăn, nsubj, Tôi), (Root, root, ăn), (ăn, dobj, cơm) }

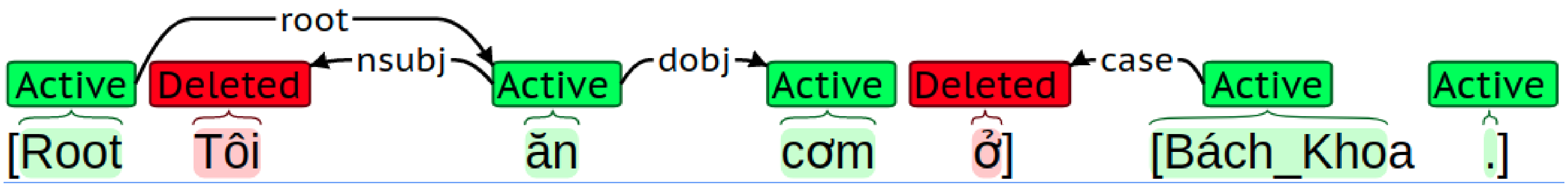
Example



SHIFT: move 'ở' from buffer to stack

A= {(ăn, nsubj, Tôi), (Root, root, ăn), (ăn, dobj, cơm) }

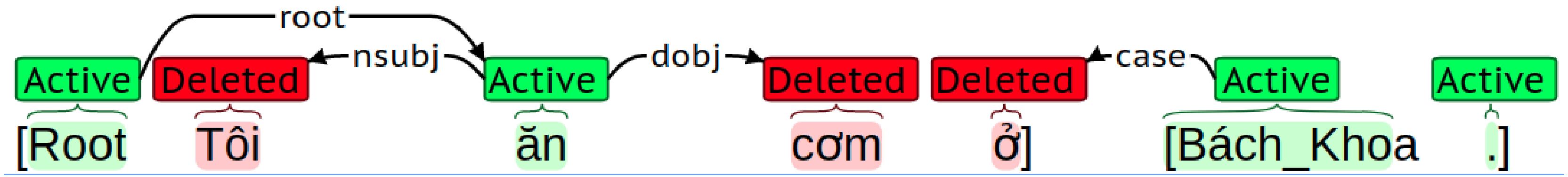
Example



LEFT$_{case}$: Remove 'ở' from Stack, add (Bách_Khoa, case, ở) to A

A= {(ăn, nsubj, Tôi), (Root, root, ăn), (ăn, dobj, cơm), (Bách_Khoa, case, ở) }

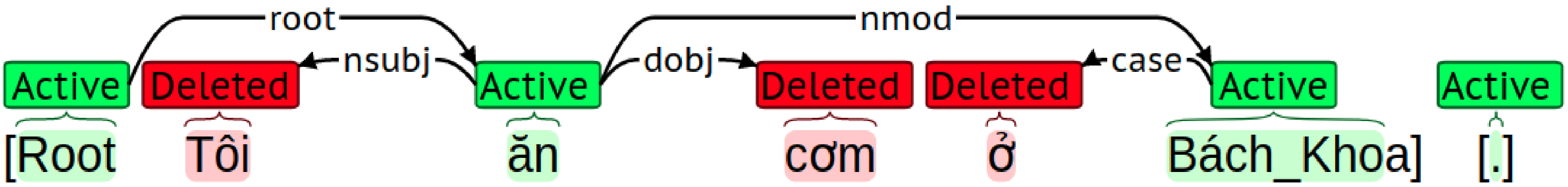
Example



REDUCE: REmove 'cơm' from Stack

A= {(ăn, nsubj, Tôi), (Root, root, ăn), (ăn, dobj, cơm), (Bách_Khoa, case, ở) }

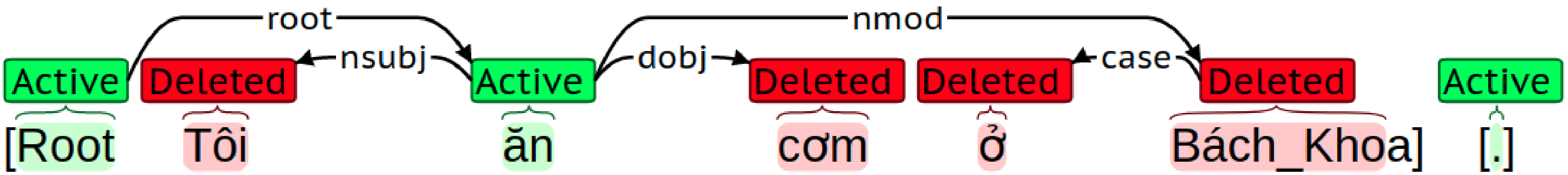
Example



$\text{RIGHT}_{\text{nmod}}$: Add 'Bách_Khoa' from buffer to stack, add (ăn, nmod, Bách_Khoa) to A

A= {(ăn, nsubj, Tôi), (Root, root, ăn), (ăn, dobj, cơm), (Bách_Khoa, case, ở), (ăn, nmod, Bách_Khoa) }

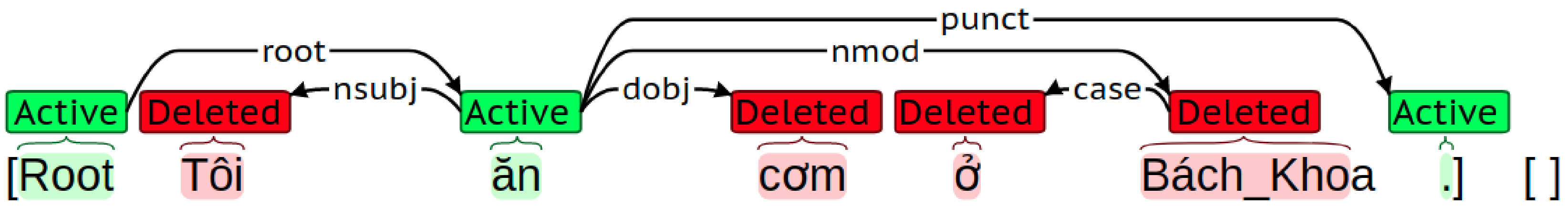
Example



REDUCE: Remove 'Bách_Khoa' from Stack

A= {(ăn, nsubj, Tôi), (Root, root, ăn), (ăn, dobj, cơm), (Bách_Khoa, case, ở), (ăn, nmod, Bách_Khoa) }

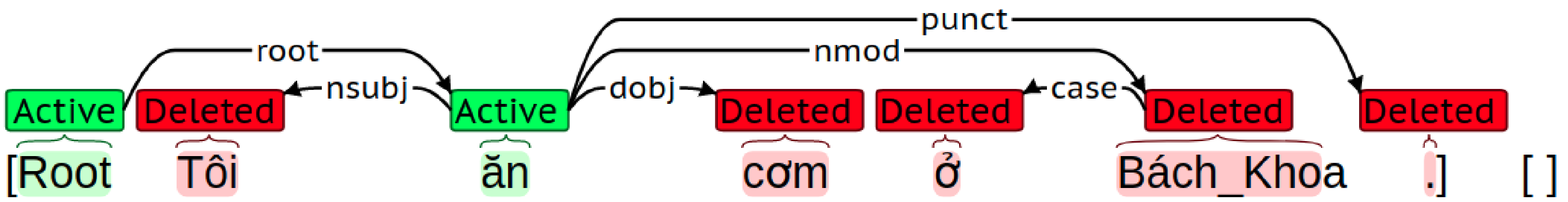
Example



RIGHT$_{\text{punct}}$: Add '.' from buffer to stack, add (ăn, punct, .) to A

A= {(ăn, nsubj, Tôi), (Root, root, ăn), (ăn, dobj, cơm), (Bách_Khoa, case, ở), (ăn, nmod, Bách_Khoa), (ăn, punct, .) }

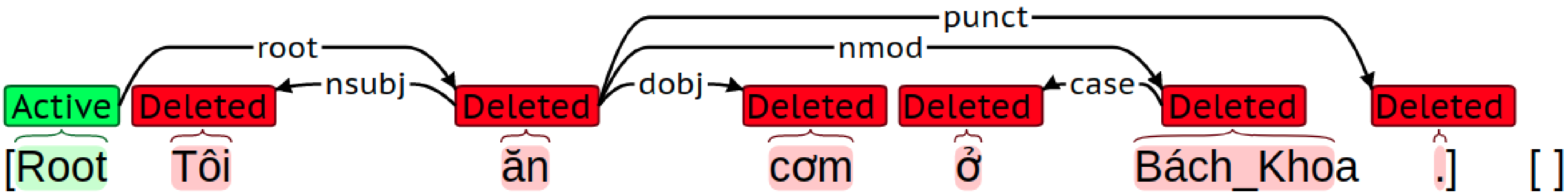
Example



REDUCE: Remove '.' from Stack

A= {(ăn, nsubj, Tôi), (Root, root, ăn), (ăn, dobj, cơm), (Bách_Khoa, case, ở), (ăn, nmod, Bách_Khoa), (ăn, punct, .) }

Example

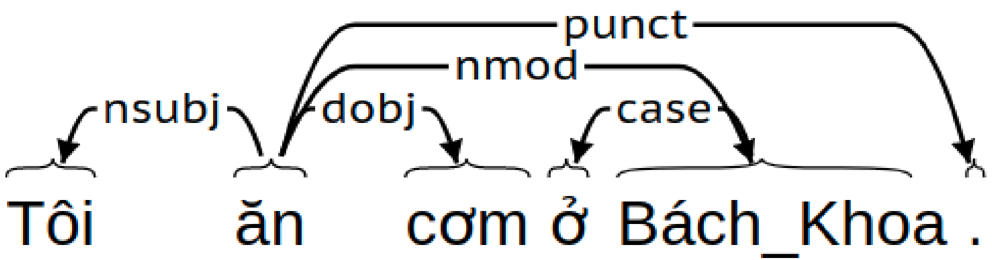


REDUCE: Remove 'ăn' from Stack

Now is the final configuration, Stack = {Root}, Buffer ={}. Return A

Example

Final tree



Approaches

- Transition-based
  - Nivre algorithm
- **Graph-based**
- Current approaches
  - End to end learning
  - Joint learning

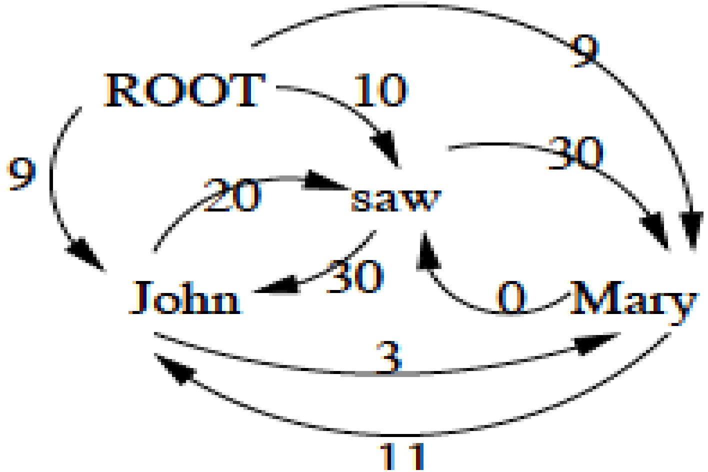
Graph-based Dependency Parsing

- Goal: Find the highest scoring dependency tree T for sentence S
  - If S is unambiguous, T is the correct parse.
  - If S is ambiguous, T is the highest scoring parse.
- Where do scores come from?
  - Weights on dependency edges by machine learning
  - Learned from large dependency treebank
- Where are the grammar rules?
  - Data-driven processing

Graph-based Dependency Parsing

- Map dependency parsing to maximum spanning tree
- Idea:
  - Build initial graph: fully connected
    - Nodes: words in sentence to parse
    - Edges: Directed edges between all words
    - + Edges from ROOT to all words
  - Identify maximum spanning tree
    - Tree s.t. all nodes are connected
    - Select such tree with highest weight
    - Arc-factored model: Weights depend on end nodes & link
      - Weight of tree is sum of participating arcs

Initial Tree

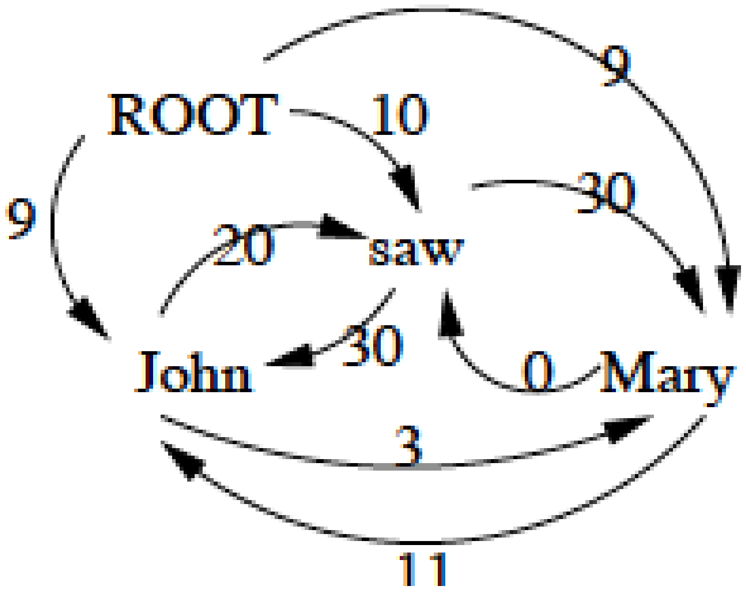


- Sentence: John saw Mary (McDonald et al, 2005)
  - All words connected; ROOT only has outgoing arcs
- Goal: Remove arcs to create a tree covering all words
  - Resulting tree is dependency parse

Maximum Spanning Tree

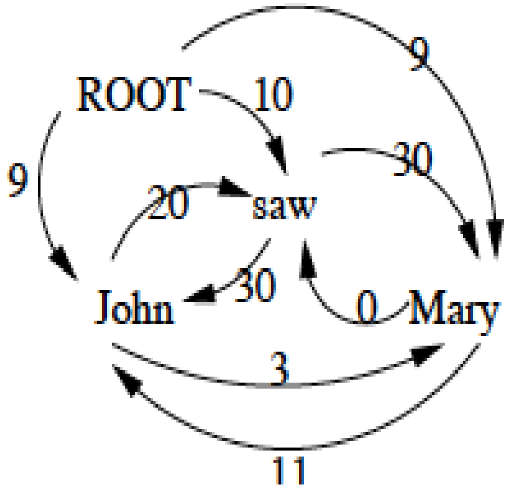
- McDonald et al, 2005 use variant of Chu-Liu-Edmonds algorithm for MST (CLE)
- Sketch of algorithm:
  - For each node, greedily select incoming arc with max w
  - If the resulting set of arcs forms a tree, this is the MST.
  - If not, there must be a cycle.
    - "Contract" the cycle: Treat it as a single vertex
    - Recalculate weights into/out of the new vertex
    - Recursively do MST algorithm on resulting graph
- Running time: naïve: $O(n^3)$; Tarjan: $O(n^2)$
  - Applicable to non-projective graphs

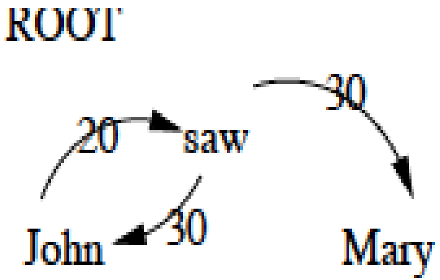
Initial Tree



CLE: Step 1

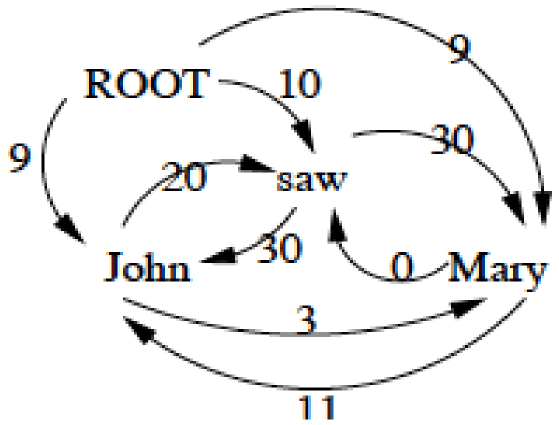
- Find maximum incoming arcs
  - Is the result a tree?
    - No
  - Is there a cycle?
    - Yes, John/saw

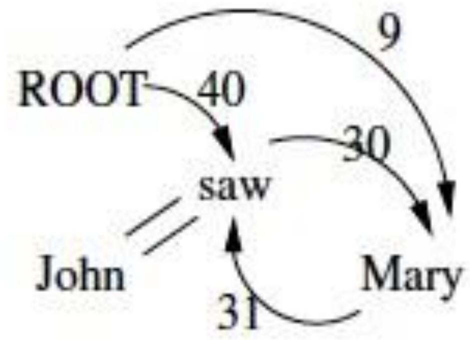




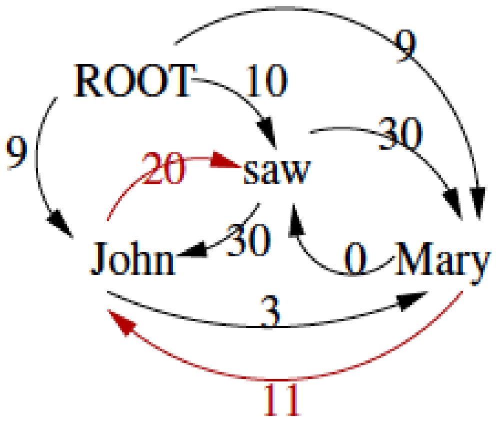
CLE: Step 2

- Since there's a cycle:
  - Contract cycle & reweight
  - John+saw as single vertex
  - Calculate weights in & out as:
    - Maximum based on internal arcs
    - and original nodes
- Recurse

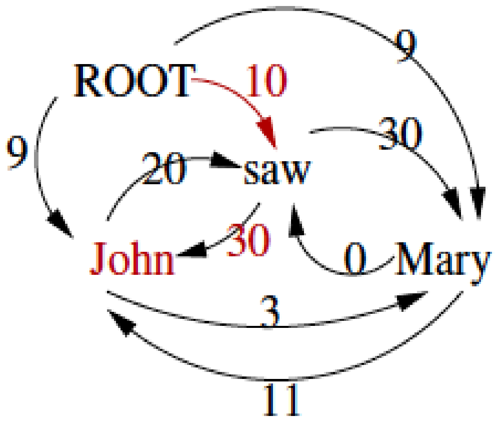




Calculating Graph



s(Mary, C) $11+20 = 31$

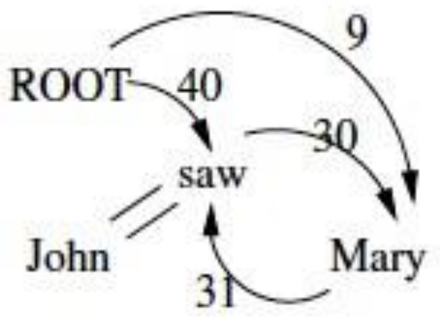


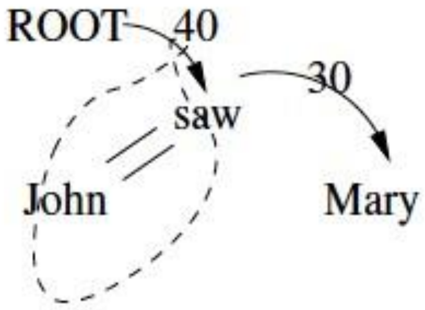
s(ROOT, C) $10+30 = 40$

CLE: Recursive Step

- In new graph, find graph of
  - Max weight incoming arc for each word

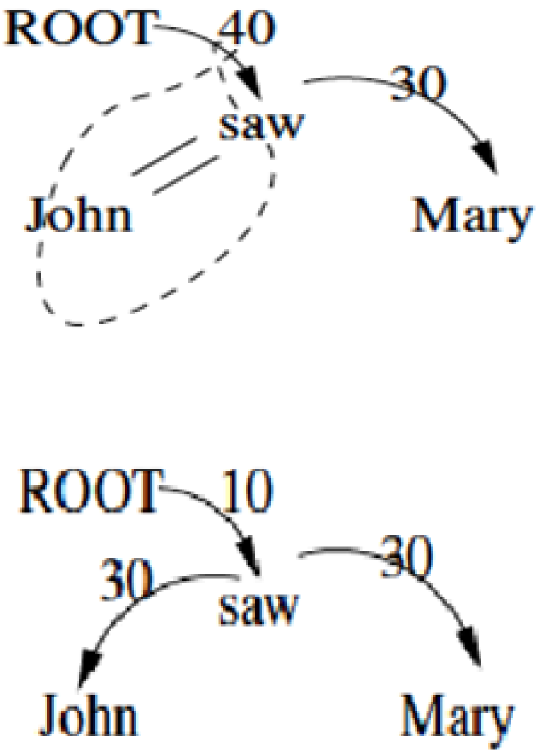
- Is it a tree? Yes!
  - MST, but must recover internal arcs → parse





CLE: Recovering Graph

- Found maximum spanning tree
  - Need to ‘pop’ collapsed nodes
- Expand “ROOT → John+saw” = 40
- MST and complete dependency parse



Learning Weights

- Weights for arc-factored model learned from corpus
  - Weights learned for tuple (w_i, w_j, l)
- McDonald et al, 2005 employed discriminative ML
  - Perceptron algorithm or large margin variant
- Operates on vector of local features

Features for Learning Weights

- Simple categorical features for (w_i, L, w_j) including:
  - Identity of w_i (or char 5-gram prefix), POS of w_i
  - Identity of w_j (or char 5-gram prefix), POS of w_j
  - Label of L, direction of L
  - Sequence of POS tags b/t w_i, w_j
  - Number of words b/t w_i, w_j
  - POS tag of w_{i-1}, POS tag of w_{i+1}
  - POS tag of w_{j-1}, POS tag of w_{j+1}
- Features conjoined with direction of attachment and distance b/t words

Dependency Parsing

- Dependency grammars:
  - Compactly represent pred-arg structure
  - Lexicalized, localized
  - Natural handling of flexible word order
- Dependency parsing:
  - Conversion to phrase structure trees
  - Graph-based parsing (MST), efficient non-proj $O(n^2)$
  - Transition-based parser
    - MALTparser: very efficient $O(n)$
      - Optimizes local decisions based on many rich features

Approaches

- Transition-based
  - Nivre algorithm
- Graph-based
- **Current approaches**
  - End to end learning
  - Joint learning

End-to-end Learning

- Training data: CoNLL Format.
- Labelled information:
    - id
    - word
    - POS tag
    - Head’s id
    - Dependency labels

```
1	Nhưng	_	CC	CC	_	8	cc	_	_
2	có_vẻ	_	RB	RB	_	8	advmod	_	_
3	như	_	IN	IN	_	8	mark	_	_
4	rất	_	RB	RB	_	5	advmod	_	_
5	nhiều	_	JJ	JJ	_	6	amod	_	_
6	người	_	NN	NN	_	8	nsubj	_	_
7	chưa	_	RB	RB	_	8	neg	_	_
8	biết	_	VB	VB	_	0	ROOT	_	_
9	về	_	IN	IN	_	10	case	_	_
10	nấm	_	NN	NN	_	8	nmod	_	_
11	Agaricus	_	NNP	NNP	_	10	nmod	_	_
12	cũng	_	IN	IN	_	13	case	_	_
13	công_dụng	_	NN	NN	_	10	nmod	_	_
14	vượt_trội	_	JJ	JJ	_	13	amod	_	_
15	từ	_	IN	IN	_	16	case	_	_
16	nó	_	PRP	PRP	_	13	nmod	_	_
17	.	_	PUNCT	PUNCT	_	8	punct	_	_

1	Nhằm	_	TO	TO	_	2	mark	_	_
2	hưởng_ứng	_	VB	VB	_	0	ROOT	_	_
3	chương_trình	_	NN	NN	_	2	dobj	_	_
4	“	_	PUNCT	PUNCT	_	5	punct	_	_
5	Hành_trình	_	NN	NN	_	3	nmod	_	_
6	đỏ	_	JJ	JJ	_	5	amod	_	_
7	”	_	PUNCT	PUNCT	_	5	punct	_	_
```

End-to-end Learning

Manually choosing features:

- Need experts
- #feature template is large due to the feature combination

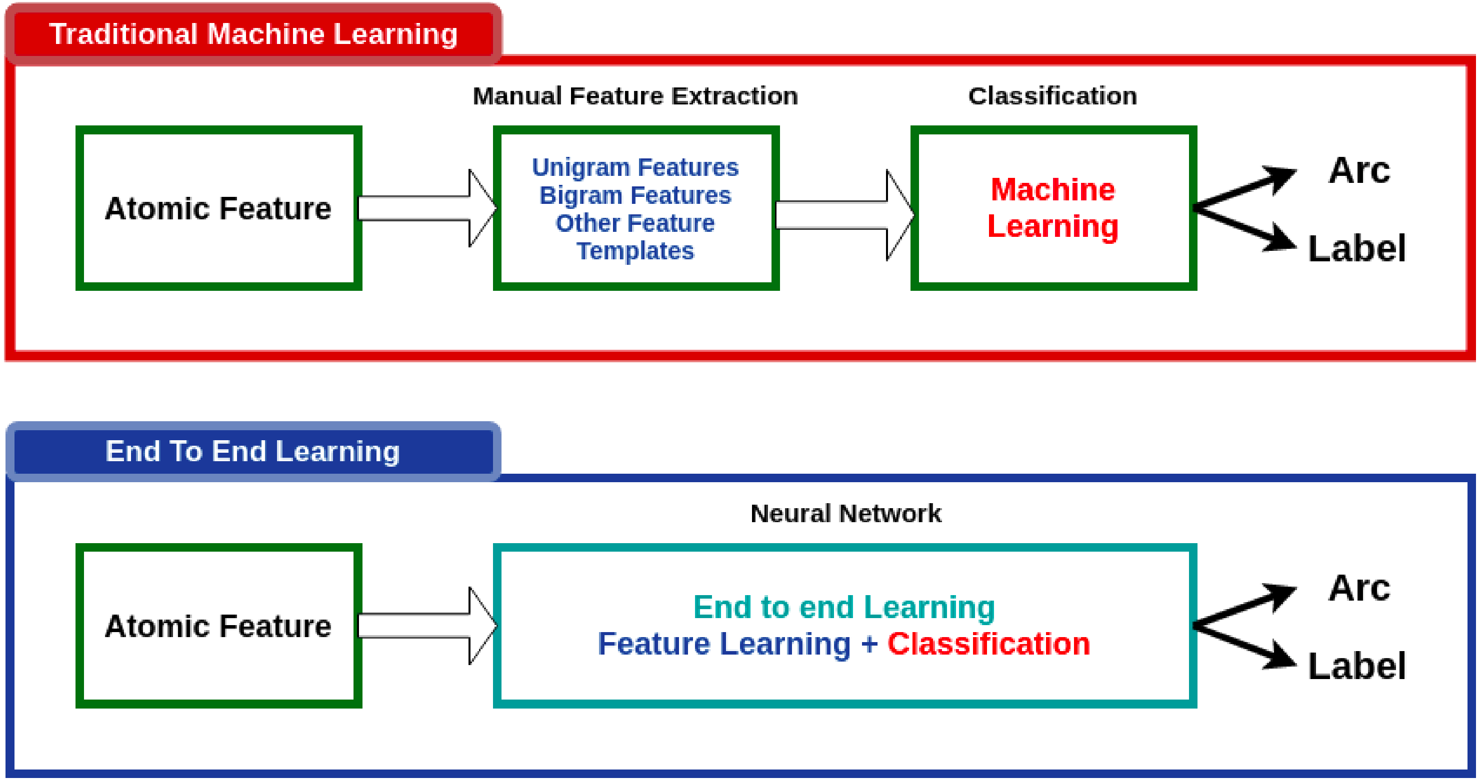
=> Maybe the highest cost for solving this task.

Basic Big-ram Features
p-word, p-pos, c-word, c-pos
p-pos, c-word, c-pos
p-word, c-word, c-pos
p-word, p-pos, c-pos
p-word, p-pos, c-word
p-word, c-word
p-pos, c-pos

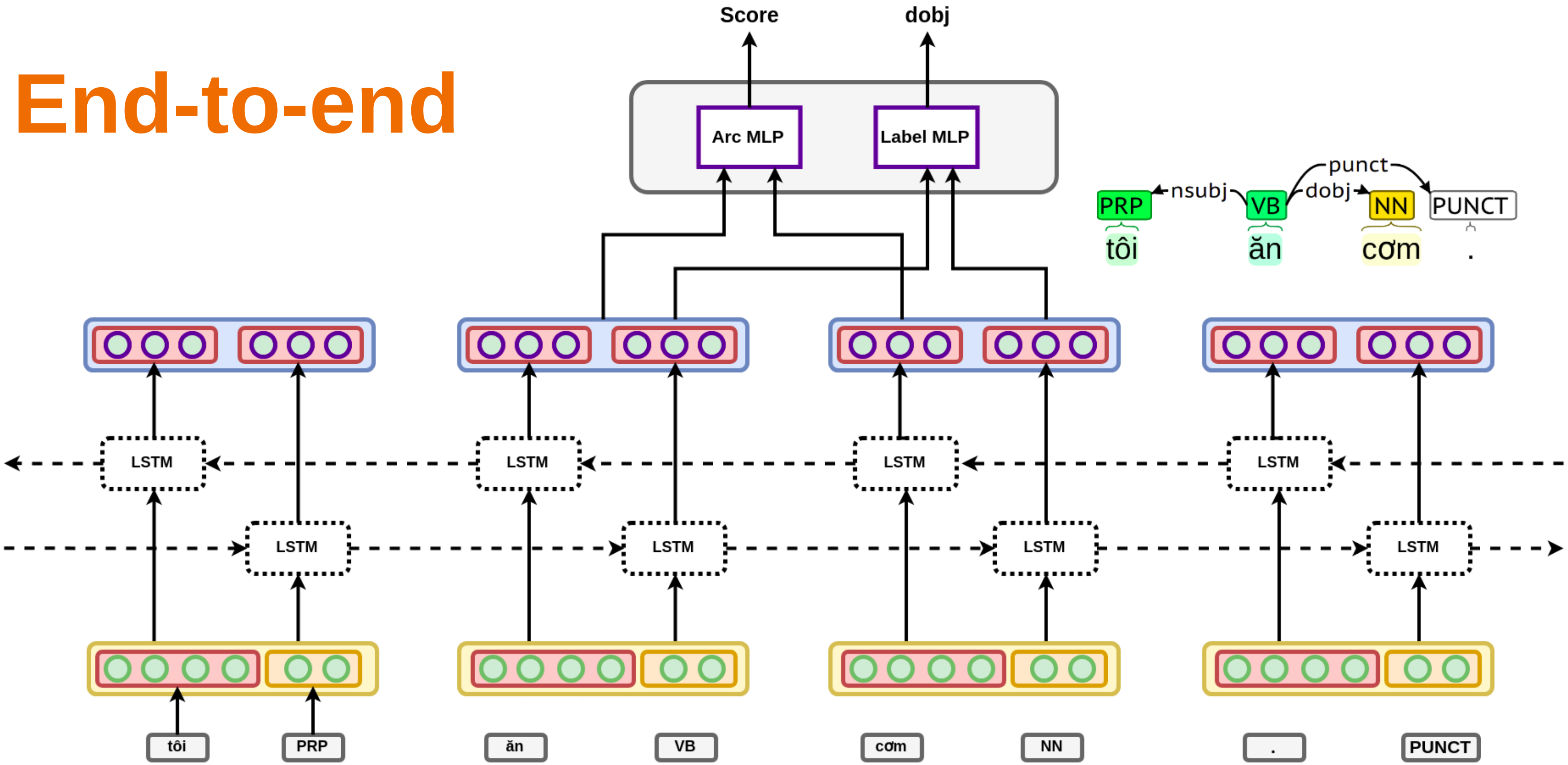
End-to-end Learning

- End to end learning for solving this task:
- Idea: training in parallel 2 modules: feature extractor and classifier
- Don’t need to choose features manually

End-to-end Learning



End-to-end



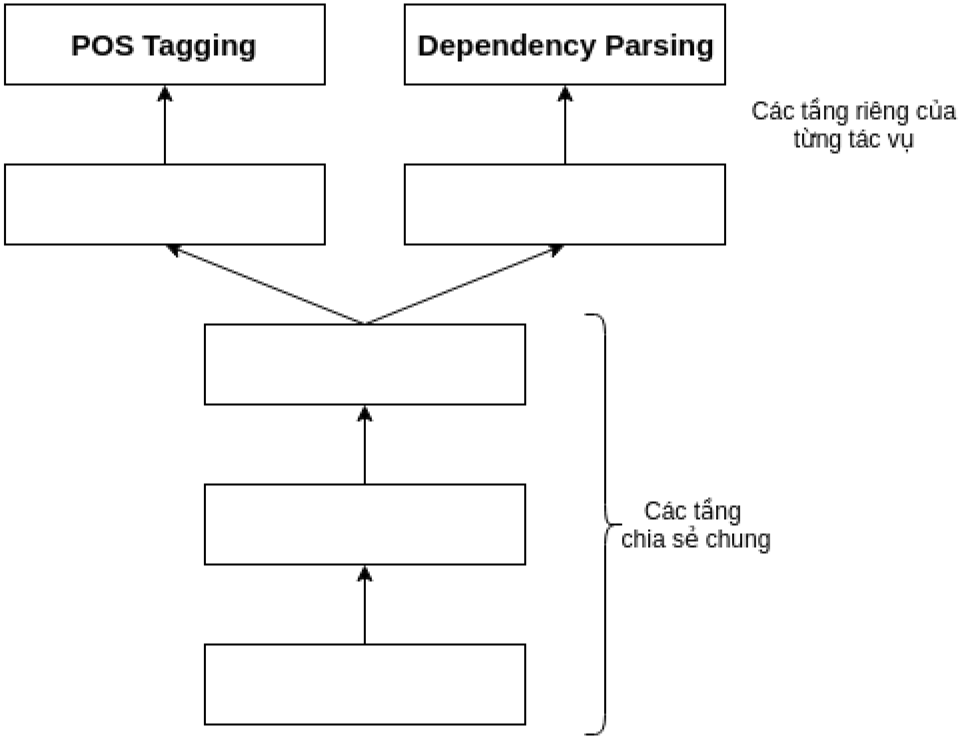
Score
dobj
Arc MLP
Label MLP
LSTM
LSTM
LSTM
LSTM
LSTM
LSTM
LSTM
LSTM
tôi
PRP
ăn
VB
cơm
NN
.
PUNCT
punct
nsubj
dobj
PRP
VB
NN
PUNCT
tôi
ăn
cơm
.

Joint Learning

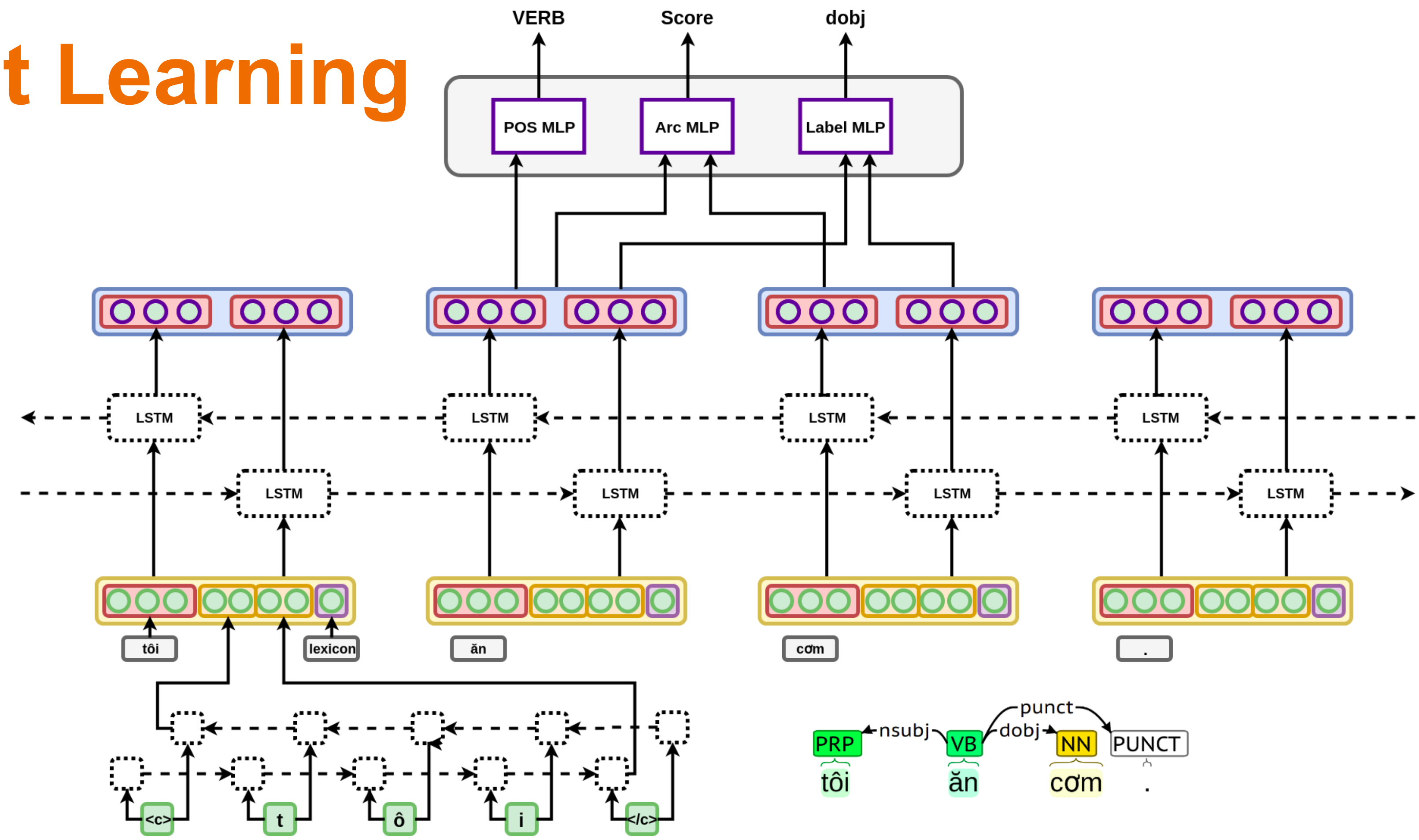
- Learning in parallel multi-tasks :
    - The learning tasks need to be related
    - Joint learning has many advantages: the shared parts contain information of several tasks, reducing model's overfitting
- 2 joint learning tasks in dependency parsing: POS Tagging + Dependency Parsing.

Joint Learning

- In the figure:
  - 2 tasks: POS tagging and Dependency Parsing share input neural layers.
  - The output of the share input layers is used as the input for each task.
- Recent research use BiLSTMs as input neural layers



Joint Learning



Joint Learning

- An RNN is used to generate word embedding
- BiLSTM generates input representation for MLP networks of POS Tagging and Dependency Parsing tasks (from vector containing information of characters, words, POS tags)

Joint Learning

2 joint learning tasks:

- POS tagging
- Computing edge weights (dependent relations connecting word pairs)
- Determining dependent labels between each word pairs.

Content

1. Overview
    - Introduction
    - Applications
    - Properties
2. Approaches
    - Transition-based
    - Graph-based
    - Current approaches
3. **Some results**

Some results

- POS Tagging
- Dependency Parsing
- Dataset
- Experimental Results

POS Tagging

- CRFSuite
- jPTDP: tool for joint learning, using Neural Network, joint learns POS Tagging and Dependency Parsing.

Dependency Parsing.

- Malt Parser (Transition based):
  - Dependency parser: **Nivre**
  - Learning method: **SVM**
- Yara Parser (Transition based):
  - Dependency parser: **Nivre**
  - Learning method: **Neural Network**
  - Improvement: **Error Exploration, Beam Search**
- BiLSTM Transition-based:
  - Dependency parser: **Nivre**
  - Learning method: **Neural Network**
  - End to end learning

Dependency Parsing.

- BiLSTM Graph-based:
  - Dependency parser: **Eisner**
  - Learning method: **Neural Network**
  - End to end learning
- jPTDP (Graph-based):
  - Dependency parser: **Eisner**
  - Learning method: **Neural Network**
  - End to end learning
  - Joint Learning POS Tagging + Dependency Parsing

Dataset

- Dataset: BK Treebank.
  - 6908 sentences in CoNLL-U Format
  - 4505 sentences for training, 1134 sentences for development, 1269 sentences for testing
- Evaluating measures:
  - POS Tagging: Accuracy.
  - Dependency Parsing: UAS and LAS
    - UAS: Unlabeled Attachment Score
    - LAS: Labeled Attachment Score

Results

Methods	UAS	LAS
Malt Parser	84.4 %	81.4 %
Yara Parser	86.3 %	83.4 %
BiLSTM Transition	86.4 %	82.9 %
BiLSTM Graph	**87 %**	**84.2%**

The input text has been assigned with POS tags.

Results

Method	POS Accuracy	UAS	LAS
CRF + Malt Parser	90.66 %	76.7 %	70.2 %
CRF + Yara Parser	90.66 %	79.1 %	72.6 %
CRF + BiLSTM Transition	90.66 %	78.9 %	72.2 %
CRF + BiLSTM Graph	90.66 %	79.7 %	**73 %**
jPTDP	89.16 %	**80.4 %**	**73 %**

The input text has not been assigned with POS tags.

Result

Method	POS Accuracy	UAS	LAS
jPTDP	89.16 %	80.4 %	73 %
jPTDP + Lexicon	**91.50 %**	**82.13 %**	**75.67 %**
jPTDP + Lexicon (Not Character Embed)	91.05%	81.46 %	75.23 %

The input text has not been assigned with POS tags.

References

[1] Kiem-Hieu Nguyen. BKTreebank: Building a Vietnamese Dependency Treebank
[2] Yue Zhang and Joakim Nivre. Training Deterministic Parsers with Non-Deterministic Oracles
[3] Eliyahu Kiperwasser and Yoav Goldberg. Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations
[4] Dat Quoc Nguyen, Mark Dras and Mark Johnson. A Novel Neural Network Model for Joint POS Tagging and Graph-based Dependency Parsing

Tài liệu tham khảo.

[5] Yuan Zhang and David Weiss. Stack-propagation: Improved Representation Learning for Syntax
[6] Barbara Plank, Anders Søgaard, Yoav Goldberg. Multilingual Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Models and Auxiliary Loss
[7] Ryan McDonald et al. Online Large-Margin Training of Dependency Parsers
[8] Yoav Goldberg and Joakim Nivre. Training Deterministic Parsers with Non-Deterministic Oracles