Evaluasi *Parser* dan *Dependency Parsing*

Sumber:

Niranjan Balasubramanian slide,
http://www.phontron.com/slides/nlp-programming-en-11-depend.pdf,
http://www.cs.umd.edu/class/fall2017/cmsc723/slides/slides_12.pdf

Kerangka

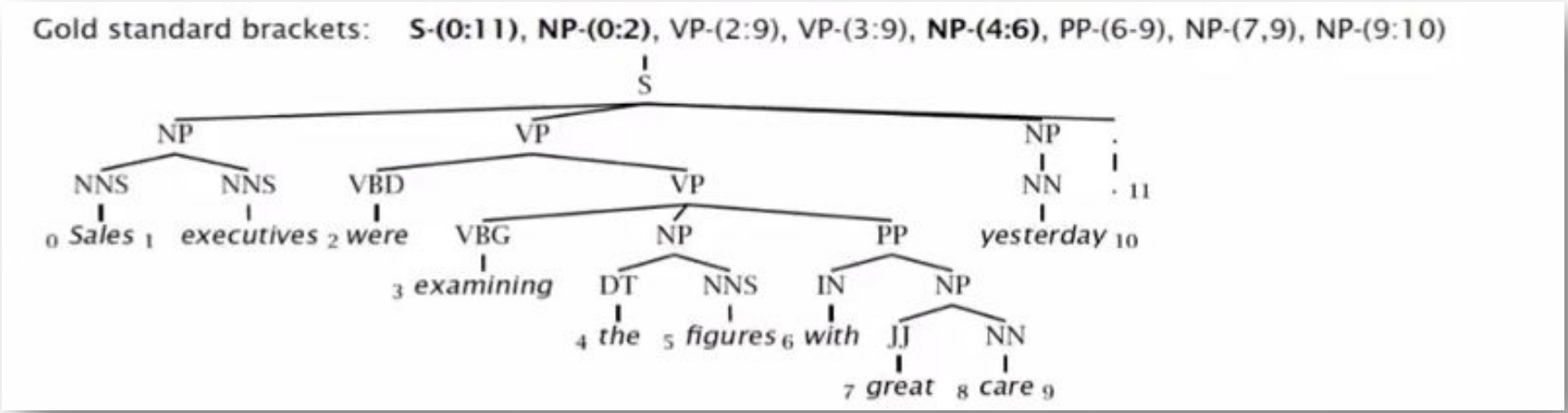
- Evaluasi *constituent-based parser*
- Definisi *dependency parsing*
- Metode untuk membangun *dependency parser*

Evaluasi *constituent-based parser*

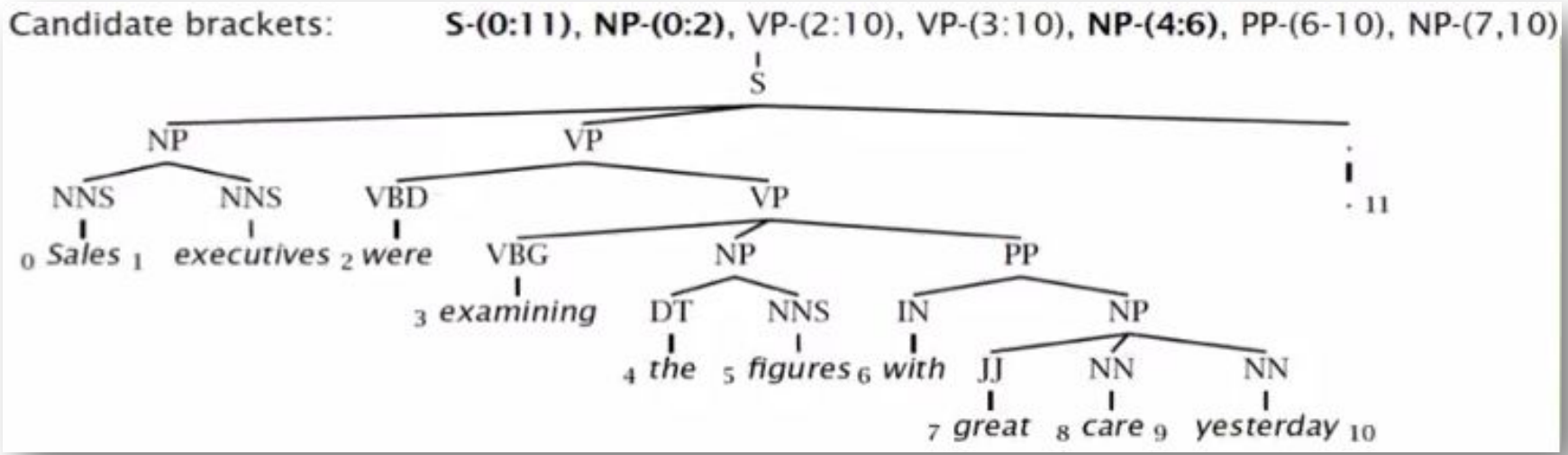
- Use the gold tree (GT) annotations in Penn Tree Bank.
- For each test check if the predicted constituency tree (PT) matches the GT.
- Parsers almost always make mistakes.
  - Allow for partial credit.

Evaluasi *constituent-based parser* (lanj.)

- Break GT and PT into a set of constituents and their token spans.
  - Compute precision and recall of the predicted constituents.



Evaluasi *constituent-based parser* (lanj.)



Metriks Evaluasi

Labeled Precision (LP) = # label constituent yang benar di PT / |PT|
Labeled Recall (LR) = # label constituent yang benar di PT / |GT|
Labeled F1 = 2 LP x LR / (LP + LR)

Performansi Beberapa Parser *constituent-based*

Parser	LP	LR	**F1**
Magerman 95	84.9	84.6	**84.7**
Collins 96	86.3	85.8	**86.0**
Klein & Manning 03	86.9	85.7	**86.3**
Charniak 97	87.4	87.5	**87.4**
Collins 99	88.7	88.6	**88.6**

Performansi Beberapa Parser *constituent-based* (lanj.)

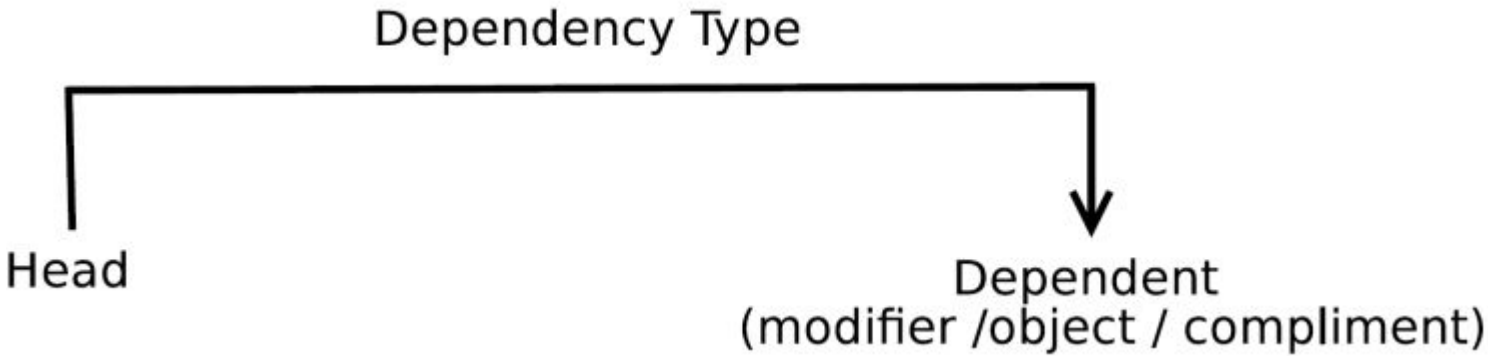
Parser	*F1* ≤ *40 words*	*F1* *all words*
Klein & Manning unlexicalized 2003	86.3	85.7
Matsuzaki et al. simple EM latent states 2005	86.7	86.1
Charniak generative, lexicalized ("maxent inspired") 2000	90.1	89.5
Petrov and Klein NAACL 2007	90.6	90.1
Charniak & Johnson discriminative reranker 2005	92.0	91.4

Dependency Parsing

- Dependency grammar
- Definisi formal

Dependency Grammar

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



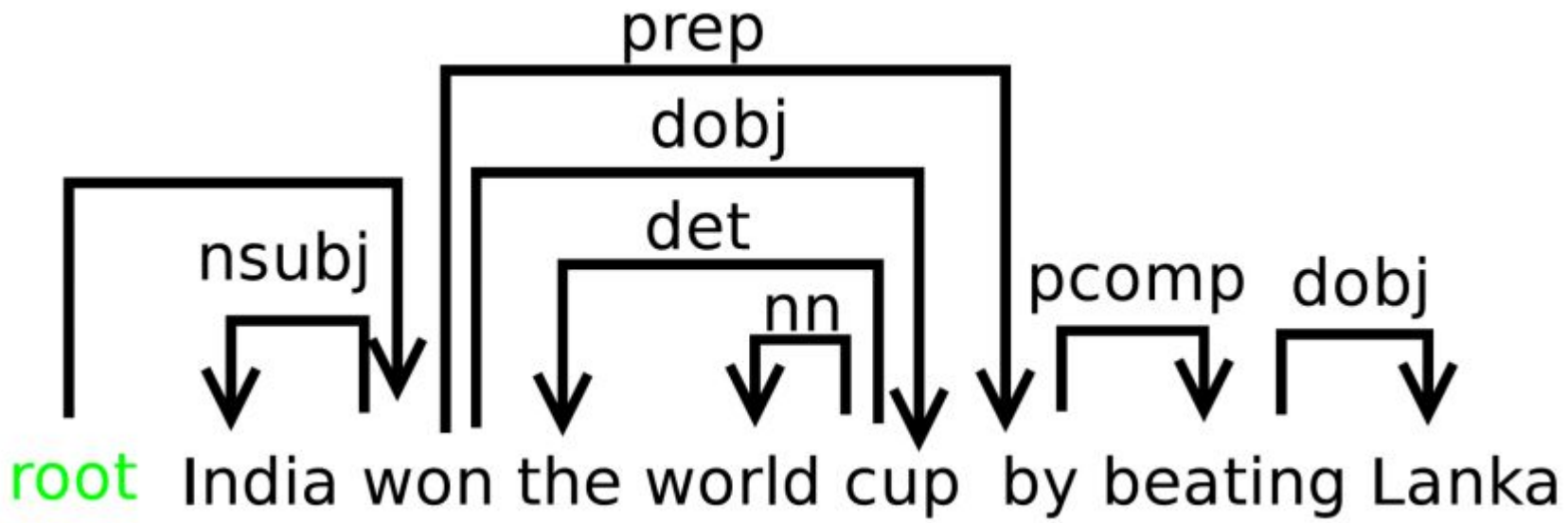
Dependency Relations

Argument Dependencies	**Description**
nsubj	nominal subject
csubj	clausal subject
dobj	direct object
iobj	indirect object
pobj	object of preposition
Modifier Dependencies	**Description**
tmod	temporal modifier
appos	appositional modifier
det	determiner
prep	prepositional modifier

Contoh *dependency relations* dalam kalimat

Relation	Examples with *head* and **dependent**
NSUBJ	**United** *canceled* the flight.
DOBJ	United *diverted* the **flight** to Reno.
	We *booked* her the first **flight** to Miami.
IOBJ	We *booked* **her** the flight to Miami.
NMOD	We took the **morning** *flight*.
AMOD	Book the **cheapest** *flight*.
NUMMOD	Before the storm JetBlue canceled **1000** *flights*.
APPOS	*United*, a **unit** of UAL, matched the fares.
DET	**The** *flight* was canceled.
	Which *flight* was delayed?
CONJ	We *flew* to Denver and **drove** to Steamboat.
CC	We flew to Denver **and** *drove* to Steamboat.
CASE	Book the flight **through** *Houston*.

Contoh *dependency parse* sebuah kalimat



Definisi Formal *dependency*

Most 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

Restricting to trees provide computational advantages

- Single designated ROOT node that has no incoming arcs
- Except for ROOT, each vertex has exactly one incoming arc
- Unique path from ROOT to each vertex in V

Aturan dalam *dependency*

- Each word has a single head
- Dependency structure is connected
- There is a single root node from which there is a unique path to each word

Projectivity

Arc from head to dependent is projective

- If there is a path from head to every word between head and dependent

Dependency tree is projective

- If all arcs are projective
- Or equivalently, if it can be drawn with no crossing edges

Projective trees make computation easier

- But most theoretical frameworks do not assume projectivity
- Need to capture long-distance dependencies, free word order

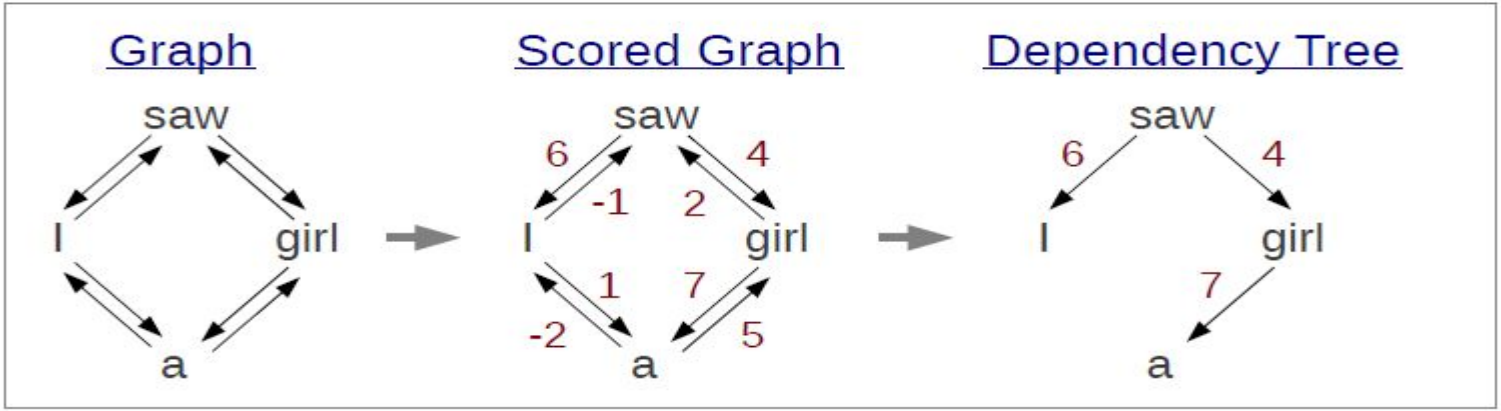
Pendekatan dalam membangun *dependency parser*

- Goal: learn a good predictor of dependency graphs
- Input: sentence
- Output: dependency graph/tree G = (V,A)

approaches: spanning tree, transition-based parsing/shift-reduce, ...

Maximum Spanning Tree

- Each dependency is an edge in a directed graph
- Assign each edge a score (with machine learning)
- Keep the tree with the highest score



(Chu-Liu-Edmonds Algorithm)

Transition-based/shift-reduce dependency parser

Builds on shift-reduce parsing [Aho & Ullman, 1927]

Configuration:

- Stack
- Input buffer of words
- Set of dependency relations

Goal of parsing: find a final configuration where all words accounted for relations form dependency tree

Transition Operators

- Transitions: produce a new configuration given current configuration

- Parsing is the task of
  - Finding a sequence of transitions
  - That leads from start state to desired goal state

- Start state
  - Stack initialized with ROOT node
  - Input buffer initialized with words in sentence
  - Dependency relation set = empty

- End state
  - Stack and word lists are empty
  - Set of dependency relations = final parse

Arc Standard Transition System

- LEFT-ARC:
  - create head-dependent rel. between word at top of stack and 2nd word (under top)
  - remove 2nd word from stack
- RIGHT-ARC:
  - Create head-dependent rel. between word on 2nd word on stack and word on top
  - Remove word at top of stack
- SHIFT
  - Remove word at head of input buffer
  - Push it on the stack

Arc Standard Transition System

Preconditions

- ROOT cannot have incoming arcs
- LEFT-ARC cannot be applied when ROOT is the 2nd element in stack
- LEFT-ARC and RIGHT-ARC require 2 elements in stack to be applied

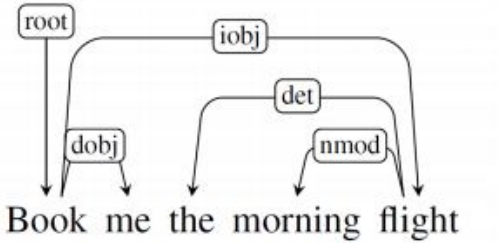
Transition-based dependency parser

- Assume an oracle
- Parsing complexity: Linear in sentence length!
- Greedy algorithm, unlike Viterbi for POS tagging

```
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
```

Ilustrasi *transition-based parsing*



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	

Bagaimana memperoleh *oracle*?

Multiclass classification problem

- Input: current parsing state (e.g., current and previous configurations)
- Output: one transition among all possible transitions
- Q: size of output space?

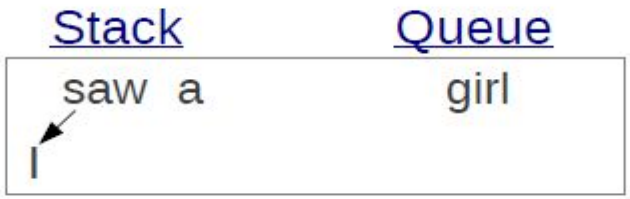
Supervised classifiers can be used, e.g., perceptron

Diskusi:

- What are good features for this task?
- Where do we get training examples?

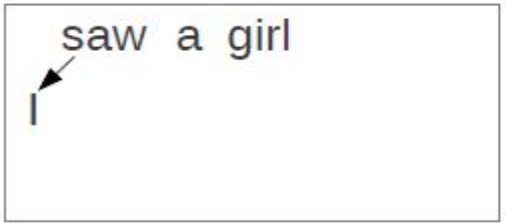
Classification for action

- Given a state:

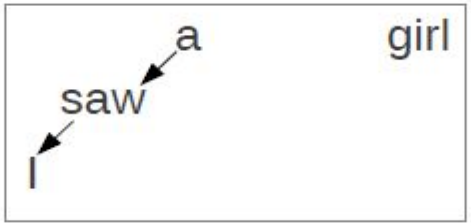


- Which action do we choose?

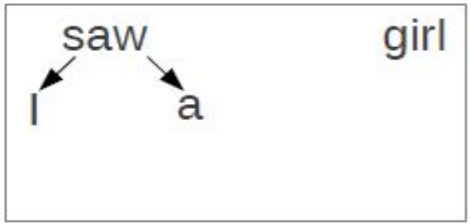
shift ?



r left ?



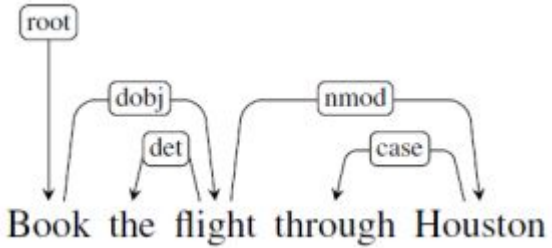
r right ?



- Correct actions → correct tree

Membangun Data Latih

Treebank dataset:



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