4. Syntactic Structure and Dependency Parsing

Key Words

- **Syntatic Structure** : "구문 구조", 문장을 하위 구조로 쪼개어 보는 것
- **Constituency Structure** : "구성 구조", 구문구조 방법 중 하나
- **Dependency Structure** : "의존 구조", 구문구조 방법 중 하나
- **Universial dependencies** : 위의 Constituency Structure 와 Dependency Structure 를 동시에 사용하는 구조
- **Dependency Grammar** : 말그대로 의존 구조의 문법
- **Treebank** : 구문 분석한것을 tree 구조로 저장해놓은 것
- **Parser** : 구문 분석기
  - ⭐ **Dependency Parser** : 의존 구조 분석기
- **Greedy transition-based parsing** : dependency parser의 알고리즘 중 하나

0. Intro

- 왜 NLP에서 구문 분석이 필요할까?
  - 사람들은 복잡한 idea를 소통할 때, 단어들이 큰 단위로 합쳐진 것인 '문장'을 활용
  - 따라서, 언어 모델이 문장을 더 깊게 이해하기 위해서는 구문 분석이 요구됨

1. Syntactic Structure: Constituency and Dependency

Constituency Structure

Context-free grammers(CFGs)

- 단어 간의 종속관계를 무시한 채 문장 내의 단어를 각각의 객체처럼 구분하는 방식
- 예시)

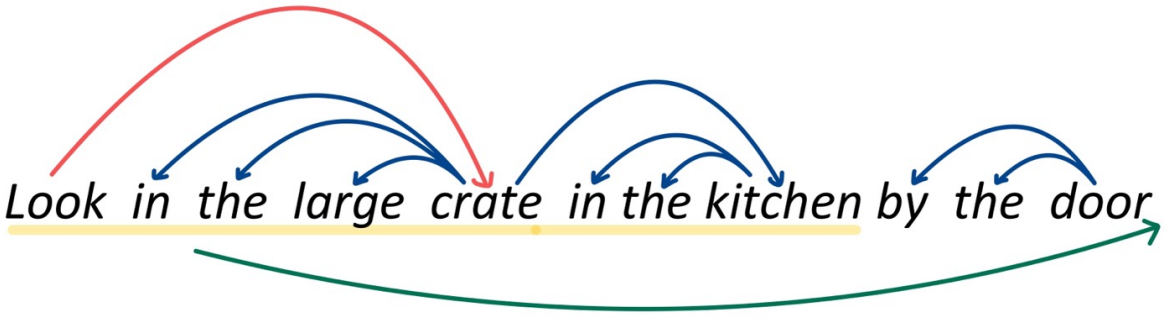
  the cuddly cat by the door

  → {the cuddly cat}, {by the door}

  → {the, cuddly, cat}, {by, the, door}
- 문장을 점점 더 작은 단위로 분할
  - 문법(grammer) + 어휘(lexicon)

Dependency Structure

- 문맥 내에서 단어들이 가지는 관계를 고려한 구조 분석
  - 단어들이 문맥에 따라 parent(= root)와 child의 관계를 가지게 됨



구문 분석이 필요한 이유

- 언어의 모호성

  → 모델이 문장 구조를 잘 이해하여 더 정확한 분석을 수행하기 위해서
- 예시

1. 칼로 남성을 살해함(칼: 살해의 도구)
2. 칼을 가지고 있는 남성을 살해(칼: 남성의 특징)

(외에도 여러가지 예시가 있다..)

PP attachment ambiguities multiply

- PP(Prepositional Phrase) : 전치사 + 명사 구
  - 또 다른 종류의 모호성
- Catalan 수
  - 단어의 수(n)이 증가함에 따라 문장의 ambiguity는 지수적으로(exponentially) 증가한다.

- Catalan numbers: $C_n = (2n)!/[(n+1)!n!]$
- An exponentially growing series, which arises in many tree-like contexts:
  - E.g., the number of possible triangulations of a polygon with $n+2$ sides
    - Turns up in triangulation of probabilistic graphical models (CS228)....

Coordination scope ambiguity

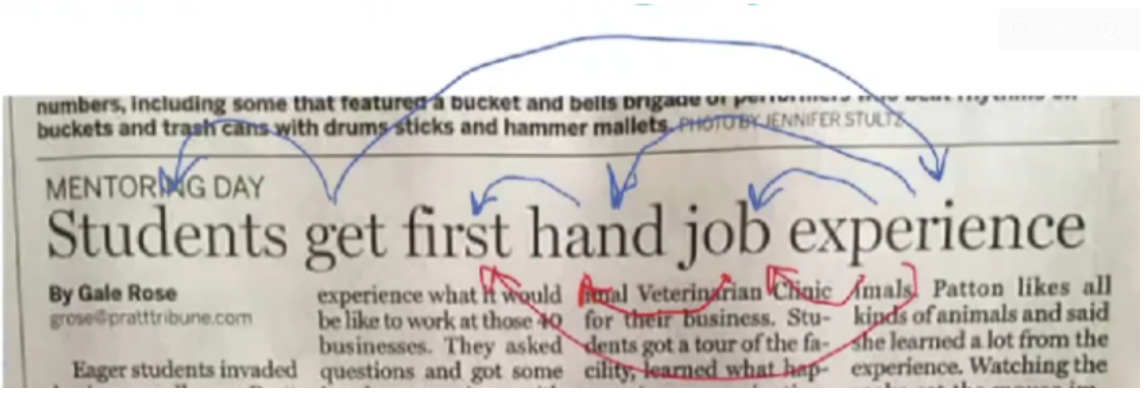
- 문장을 구분해서 읽는 “위치”에 따라 문장의 의미 해석이 달라질 수 있다.
  - 무엇이 무엇을 꾸미는지 명확하지 않은 경우
- 예시)

  Shuttle veteran and longtime NASA executive Fred Gregory appointed to board.

  1. 사람이 2명(shuttle veteran & longtime NASA executive)
  2. 1명이 여러 업무 담당(Fred Gregory라는 사람이 두 가지 일을 다 함)

Adjectival/Adverbial Modifier Ambiguity

- 수식의 범위가 모호한 경우


numbers, including some that featured a bucket and bells brigade of
buckets and trash cans with drums sticks and hammer mallets. PHOTO BY JENNIFER STULTZ

MENTORING DAY

Students get first hand job experience

By Gale Rose
grose@prattribune.com

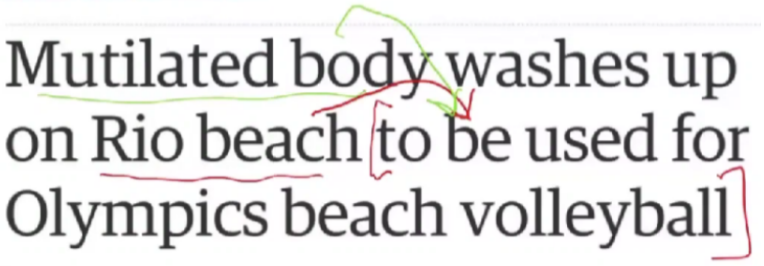
Eager students invaded

experience what it would be like to work at those 10 businesses. They asked questions and got some

for their business. Students got a tour of the facility, learned what hap-

Patton likes all kinds of animals and said she learned a lot from the experience. Watching the

Verb Pharse(VP) attachment ambiguity

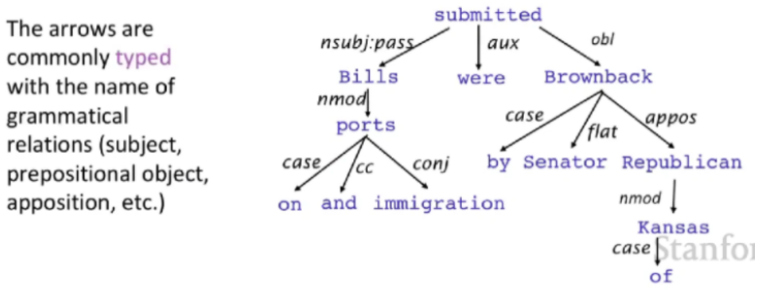


⇒ 인간의 언어 구조는 복잡하고 모호하다.

2. Dependency Grammer and Treebanks

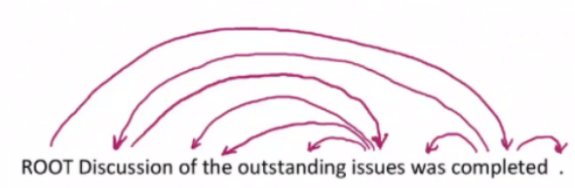
Dependency path

- dependency structure를 트리 구조로 표현 가능



문장 내의 종속 관계를 그래프로 표현 가능

Dependency Grammar and Dependency Structure



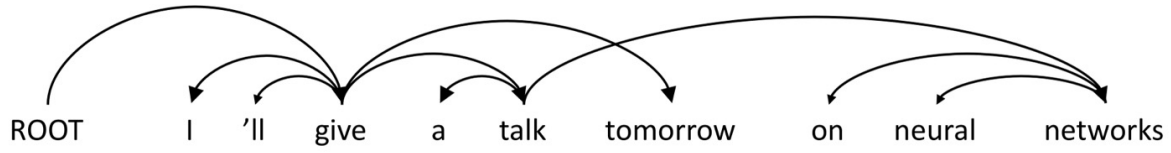
- 화살표의 tail은 상위 구조, head(= arrow)는 하위 구조를 의미

The rise of annotated data

- 많은 사람들이 parser를 만드려는 시도를 함
- 인력으로 treebank를 만드는 것은 비용이 많이 듦

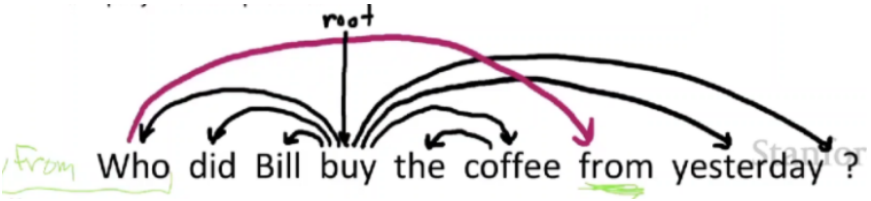
- treebank는 재사용성이 높고, NLP system가 제대로 parse 했는지 evaluate 할 수 있는 좋은 도구임

Dependency conditioning Preferences



Projectivity(투사성)

- 도치된 형태가 아닌 정상적인 형태의 문장이여야 한다..?



3. Methods of Dependency Parsing

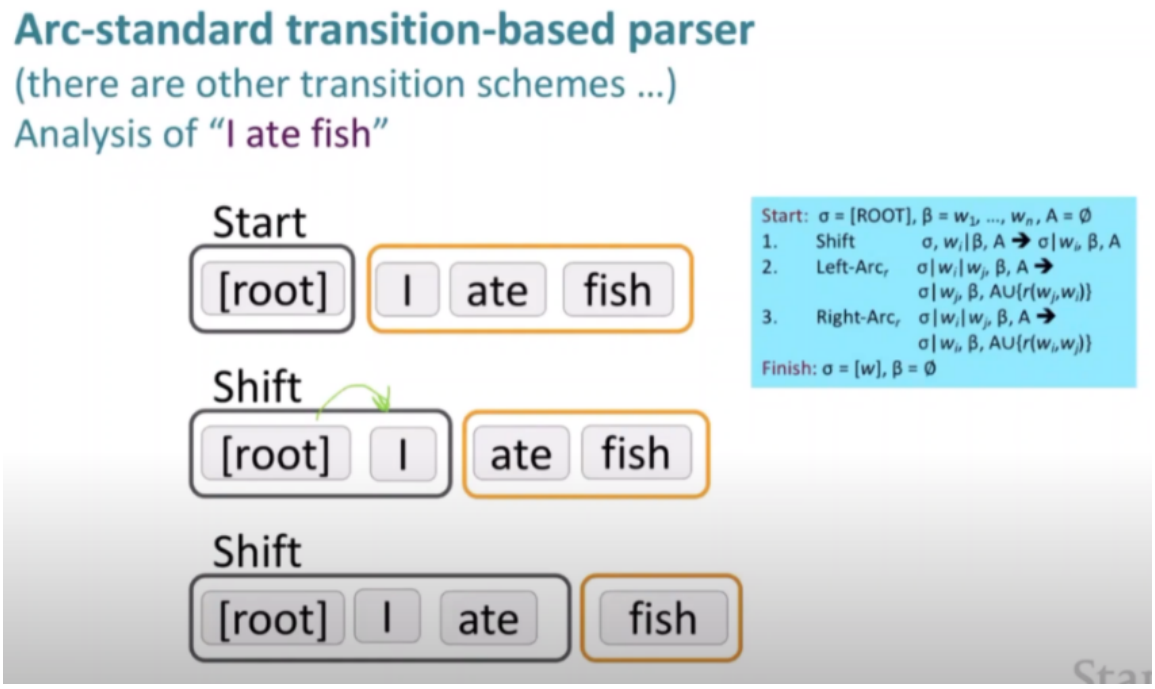
- **Transition-based** dependency parsing

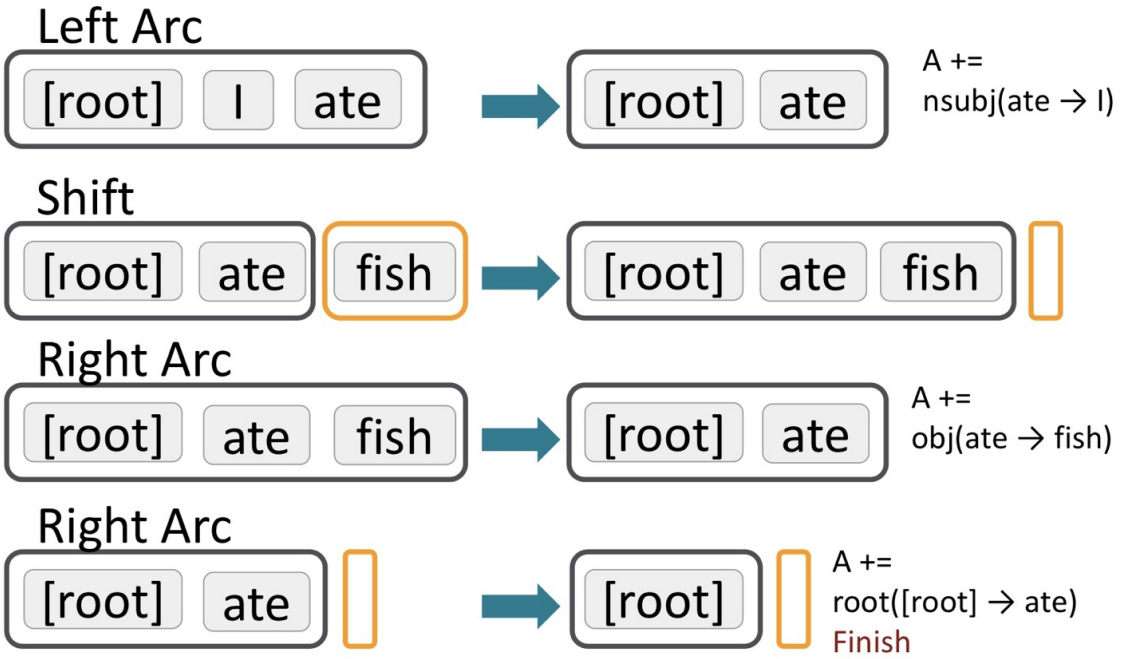
Dependency Parser 생성 방법

1. dynamic programming(1996)
   - 경우의수 때려 박아서 만들어 볼 수는 있었지만, time cost가 비쌌다.
2. 그래프 알고리즘의 사용
   - 문장으로 minimum spanning tree를 생성
   - 그리고 dependencies를 ML classifier 을 사용해서 점수메김
   - GNN-parser(2017) : 성공적!!
3. Dependency의 제약을 느슨하게 만들어도 봄
4. **transition step을 reduce한다.**
   - MaltParser(2008) : greedy transition-based parsing 에서 사용하는 ML classifier 알고리즘

Greedy transition-based parsing(Nivre 2003)

- 심플한 greedy discriminative dependency parser
- parser의 구성
    - a stack σ, written with top to the right
        - which starts with the ROOT symbol
    - a buffer β, written with top to the left
        - which starts with the input sentence
    - a set of dependency arcs A
        - which starts off empty
    - a set of actions
- 예시





- [root]가 stack에 유일하게 남으면 종료
- root까지 양옆을 단어 순으로 reduce 시켜주는 알고리즘
  - 해당 transition machine으로 문장을 parsing up 할 수 있음

> ❓ 언제 Left Arc를 사용하고 언제 Right Arc를 사용하고 언제 Shift를 사용한다는 것일까?

MaltParser

- 알고리즘에서 left arc를 할것인지, right arc를 할것인지 고르는 classifier
- 80, 90년대에는 모든 path를 일일히 explore 했지만, 그렇기에 당연히 복잡도는 아주아주 높았음
- 2000년도 초반에는 ML로 SVM등의 classifier을 훈련하여 다음 action을 선택하도록 하였음

⇒ 복잡도가 exponential에서 linear로 내려왔음

Conventional Feature Representation

(여기서부턴 5강에 정리)