Dependency Grammars and Parsing

Dependency Grammars

- Turns out that the whole CFG approach does have some shortcomings...
  - It leads to efficiency/tractability problems during parsing
  - It doesn't work very well (or tell us much) with various languages
    - So-called free word order or scrambling languages
    - Languages where the morphology does more of the work
  - And it often turns out that we don't really need much of what the trees contain

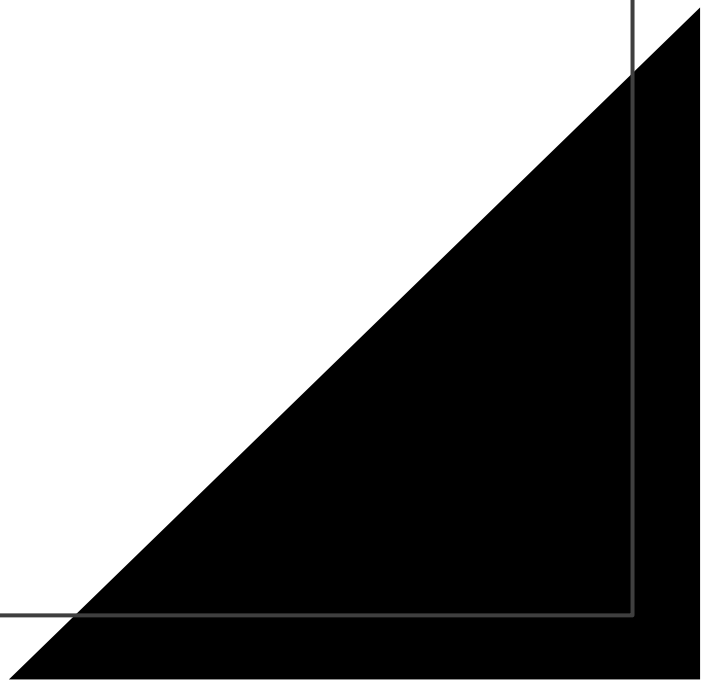
Dependency Grammars

- But it turns out you can get a lot done with just binary relations among the words in an utterance.
- In a **dependency grammar** framework, a parse is a tree where
  - the nodes stand for the words in an utterance
  - The links between the words represent dependency relations between pairs of words.
    - Relations may be typed (labeled), or not.

Comparison

- Dependency structures explicitly represent
  - Head-dependent relations (directed arcs)
  - Functional categories (arc labels)
  - Possibly some structural categories (parts-of-speech)

- Phrase structure explicitly represent
  - Phrases (non-terminal nodes)
  - Structural categories (non-terminal labels)
  - Possibly some functional categories (grammatical functions)



Dependency Relations

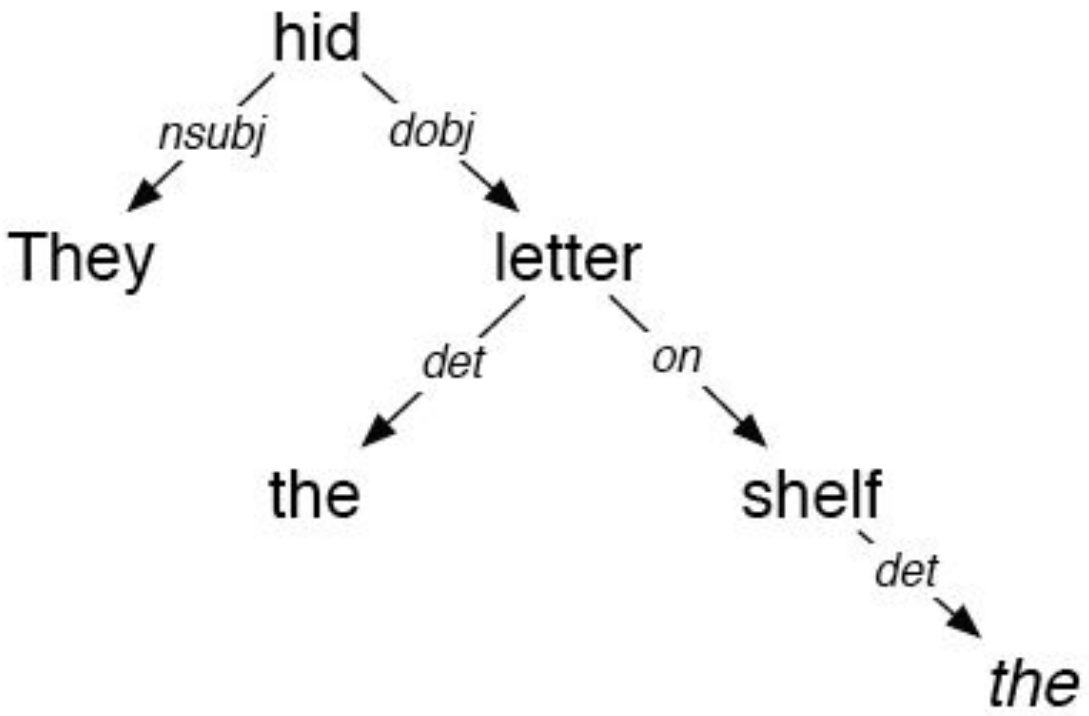
Argument Dependencies	Description
nsubj	nominal subject
csubj	clausal subject
dobj	direct object
iobj	indirect object
pobj	object of preposition
Modifier Dependencies	**Description**
tmod	temporal modifier
appos	appositional modifier
det	determiner
prep	prepositional modifier

Dependency Relations

- Dependency relations aim at revealing the syntactic structure of a sentence
- The syntactic structure of a sentence can be expressed by:
  - Word Order
  - Prosody
  - Inflection
  - Lexical items

Dependency Relations

- A → B or B ← A
- There is a **dependency relation between A and B**
- In the representations above, we say that:
  - A is a **head, B its dependent**
  - A **governs B**
  - B **is dependent on A**
- We are not only interested in the direction of the relation that exists between A and B



Dependency Parse

They hid the letter on the shelf

Properties of Dependency Relations

Dependency relations must be:

- **antisymmetric:**
  - if A → B, then B → A
  - If A governs B, then B does not govern A
  - E.g. noun-compounds:
    - *student politics ≠ politics student*
    - *politics → student ≠ student → politics*

Properties of Dependency Relations

- **antireflexive:**
  - If A → B, then A ≠ B
  - No item governs itself
    - No wordform is linearly ordered or inflected with respect to itself
    - This property also follows from the antisymmetric property of dependency relations
- **antitransitive:**
  - if A → B, and B → C, then A → C
  - If A governs B, and B governs C, then A does not govern C
  - Dependency relations are always *direct*
  - e.g. a *very smart student:*
    - *student → smart → very*

Properties of Dependency Relations

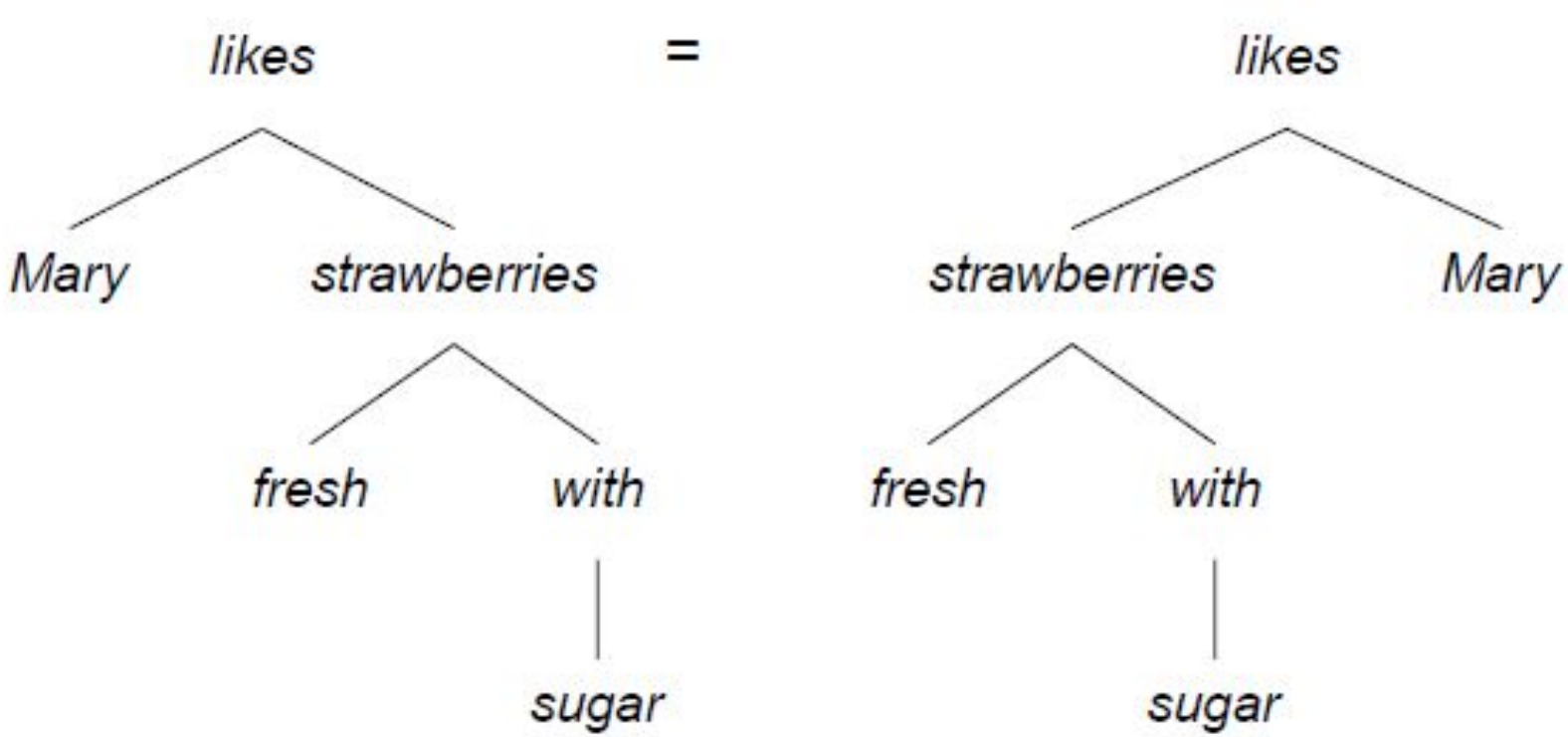
- **labeled:**
  - In *Ram likes Shyam: like → Ram,*
    like → Shyam:
  - *like →* $_{SUBJ}$ *Ram, like →* $_{OBJ}$ *Shyam*
  - In CL, unlabeled dependencies are found as well

Dependency Tree

- A dependency tree is a **connected directed labeled graph**
- Its **nodes or vertices are labeled with reduced word forms** (no inflection)
- Its **branches or arcs are labeled with the names of** syntactic relations
- It must have exactly one node that does not depend on any other node: the **root or top node of the tree**

Dependency Trees

- Linear Order is not represented in dependency trees



Dependency Parsing

- The dependency approach has a number of advantages over full phrase-structure parsing.
  - Deals well with free word order languages where the constituent structure is quite fluid
  - Parsing is much faster than CFG-based parsers
  - Dependency structure often captures the syntactic relations needed by later applications
    - CFG-based approaches often extract this same information from trees anyway.

Dependency Grammar/Parsing

- A sentence is parsed by relating each word to other words in the sentence which depend on it.
- The idea of dependency structure goes back a long way
  - To Pāṇini's grammar (c. 5th century BCE)
- Constituency is a new-fangled invention
  - 20th century invention
- Modern work often linked to work of L. Tesniere (1959)
  - Dominant approach in "East" (Eastern bloc/East Asia)
- Among the earliest kinds of parsers in NLP, even in US:
  - David Hays, one of the founders of computational linguistics, built early (first?) dependency parser (Hays 1962)

Dependency structure

- Words are linked from head (regent) to dependent
- Warning! Some people do the arrows one way; some the other way (Tesniere has them point from head to dependent…).
- Usually add a fake ROOT so every word is a dependent

Relation between CFG to dependency parse

- A dependency grammar has a notion of a head
- Officially, CFGs don't
- But modern linguistic theory and all modern statistical parsers (Charniak, Collins, Stanford, …) do, via hand-written phrasal "head rules":
  - The head of a Noun Phrase is a noun/number/adj/…
  - The head of a Verb Phrase is a verb/modal/….
- The head rules can be used to extract a dependency parse from a CFG parse (follow the heads).
- A phrase structure tree can be from a dependency tree, but dependents are flat (no VP!)

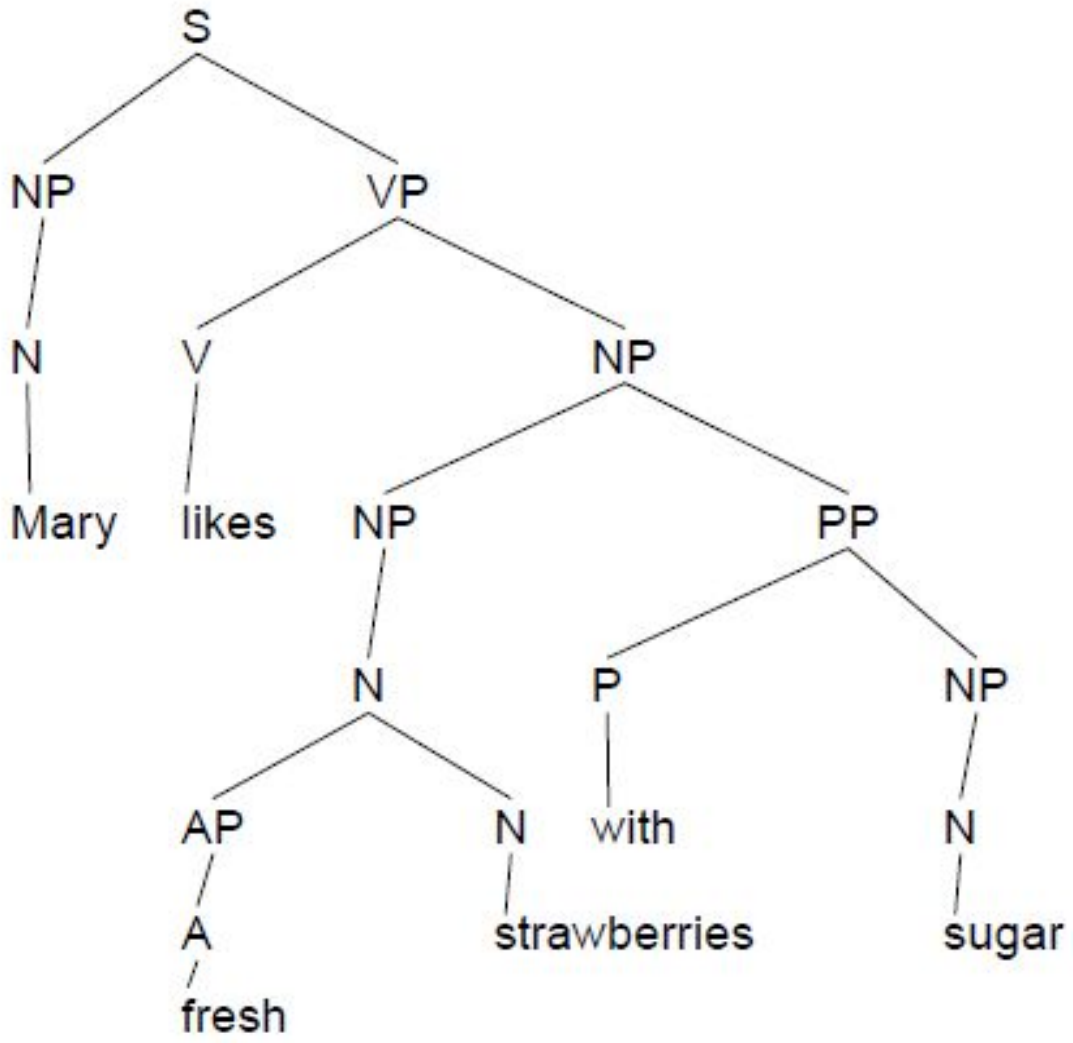
Converting a PS Tree to dependancies

1. Start at the root of the tree
2. Identify lexical head of the phrase
3. Percolate the lexical head up to its maximal projection
4. Remove redundant nodes from the tree
5. Repeat steps 2-4 for all maximal projections in the tree

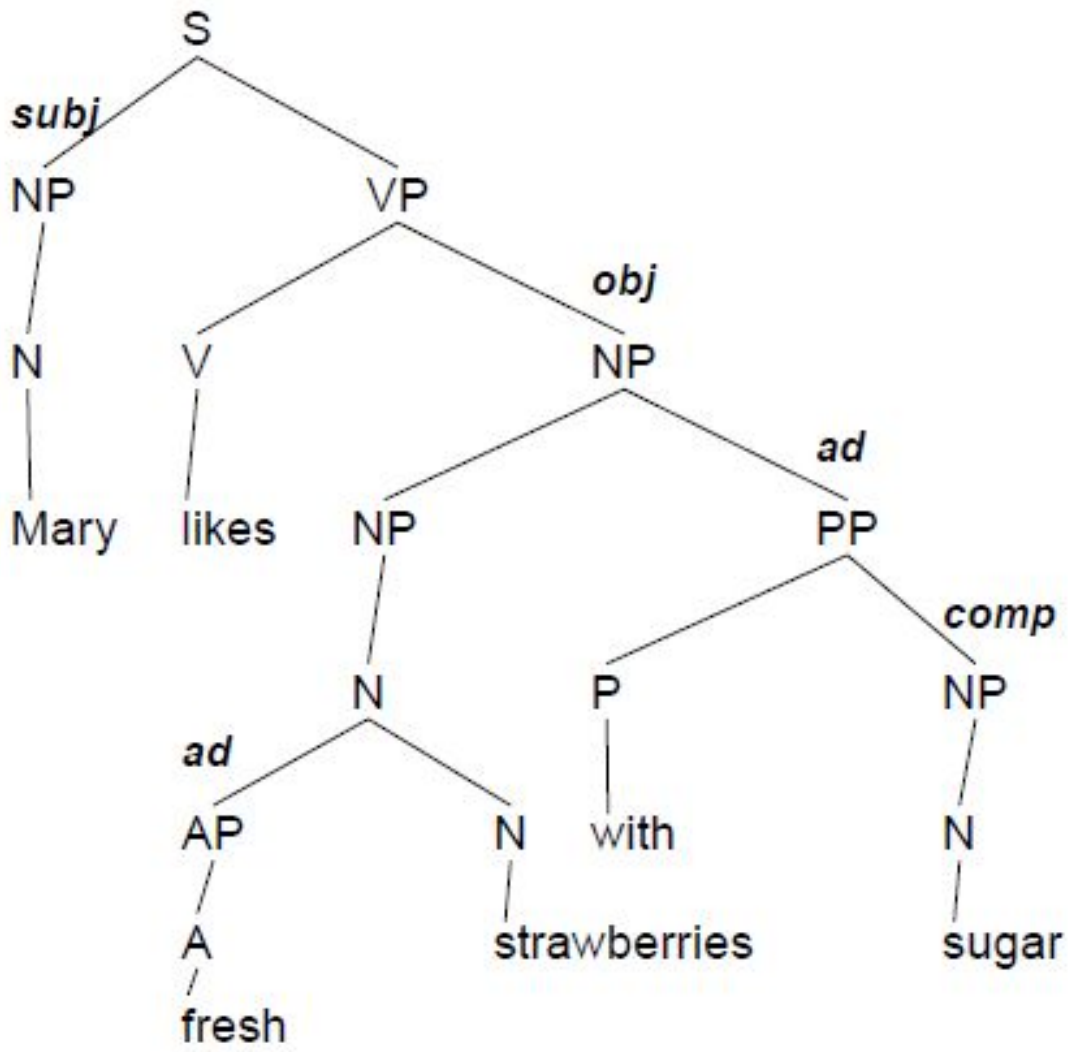
Dependency Relations

- Heads are easily identifiable in phrase structures
- So we can easily identify heads and their dependents
- But what about their labels?
- They can be defined with respect to the tree:
- Recall:
  - subject-of [NP, S]
  - object-of [NP, VP]
  - etc.
- Naturally, this means that we need to integrate labels before removing redundant nodes from the tree

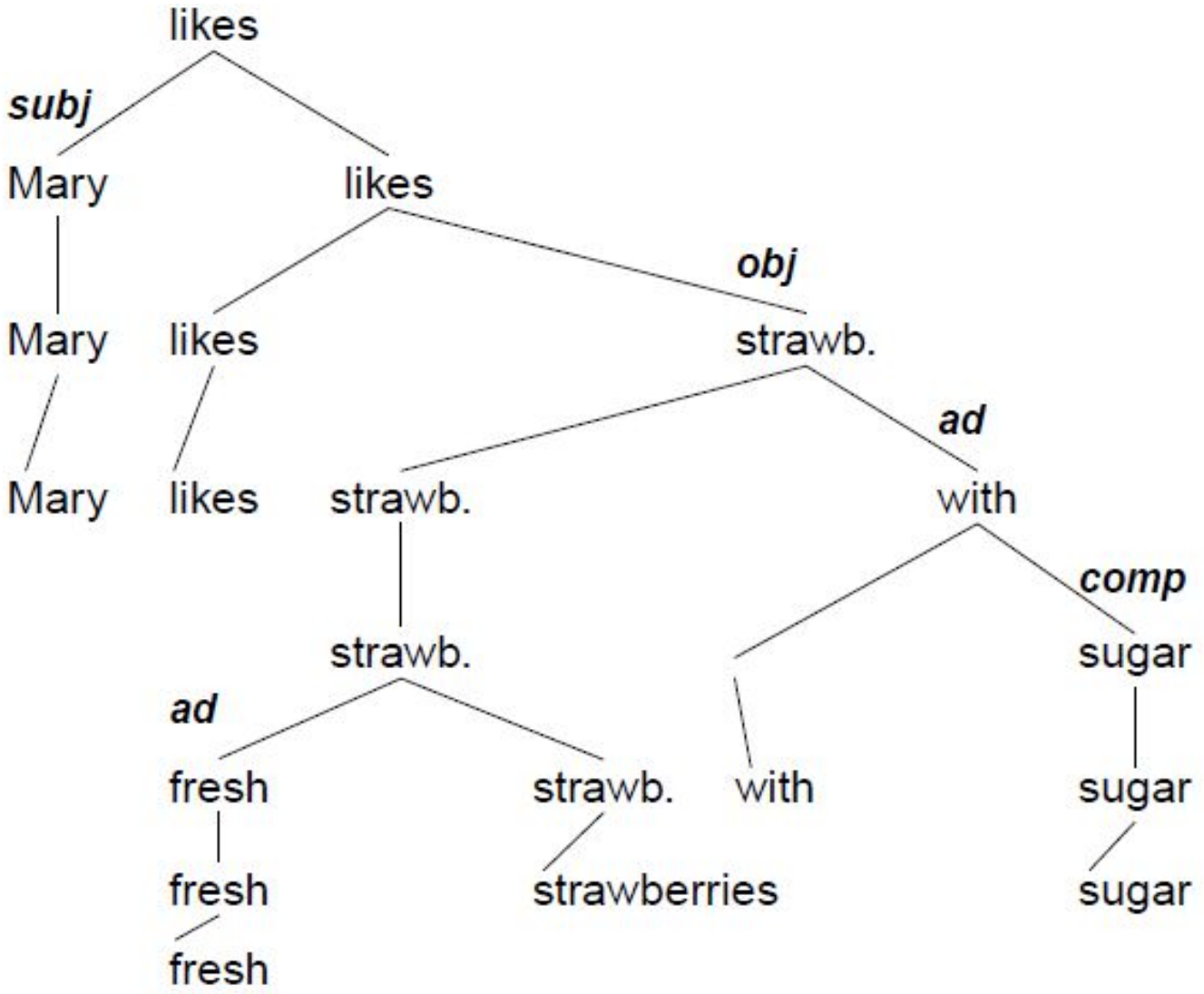
Deriving a Dependency Tree from a PS Tree



Deriving a Dependency Tree from a PS Tree



Deriving a Dependency Tree from a PS Tree



Deriving a Dependency Tree from a PS Tree