

# ECS763U/P Natural Language Processing

Julian Hough

**Week 6: Syntax II:  
Constituency Parsing and  
Dependency Grammars**

(with slides by Mehrnoosh  
Sadrzadeh)

# Generative Grammars

John saw a man with binoculars.

**S**  $\longrightarrow$  **John VP**

**VP**  $\longrightarrow$  **saw a man with binoculars**

**tV**  $\longrightarrow$  **saw**

**NP**  $\longrightarrow$  **a man with binoculars**

Meaning 1

# Generative Grammars

John saw a man with binoculars.

**S**  $\longrightarrow$  **John VP PP**

**VP**  $\longrightarrow$  **saw a man**

**tV**  $\longrightarrow$  **saw**

**NP**  $\longrightarrow$  **a man**

**PP**  $\longrightarrow$  **with binoculars**

Meaning 2

# Ambiguity

- How can we deal with the syntactic (and therefore semantic/meaning) ambiguity of the parse of 'john saw a man with binoculars'?
- We need to assign more than one possible structure!

# Syntactic Parsing

**Parsing:** The task of assigning a syntactic structure to a sentence.

## Applications in NLP

- Grammar checking in word processing tools.
- Semantic analysis, which has applications to
  - Question answering
  - Information extraction

# Grammar in Question

## Answering

**Q: Which books were written by British women authors before 1800?**

To answer to this question we need to, for example:

- what to find answers for, that is, what is the subject of the sentence: books
- which books do we need, all books or special type of books? In other words, what are the modifiers of book: books written by British women authors before 1800.

# Challenges to Parsing:

## Ambiguity

Two kinds:

- **attachment ambiguity:** a constituent can be attached to the parse tree at more than one place.
- **coordination ambiguity:** different sets of phrase can be conjoined by a conjunction, e.g. and/or

Later on: Semantic Ambiguity (meanings of words)

# Challenges to Parsing:

## Example

**“I shot an elephant in my pyjamas.”**

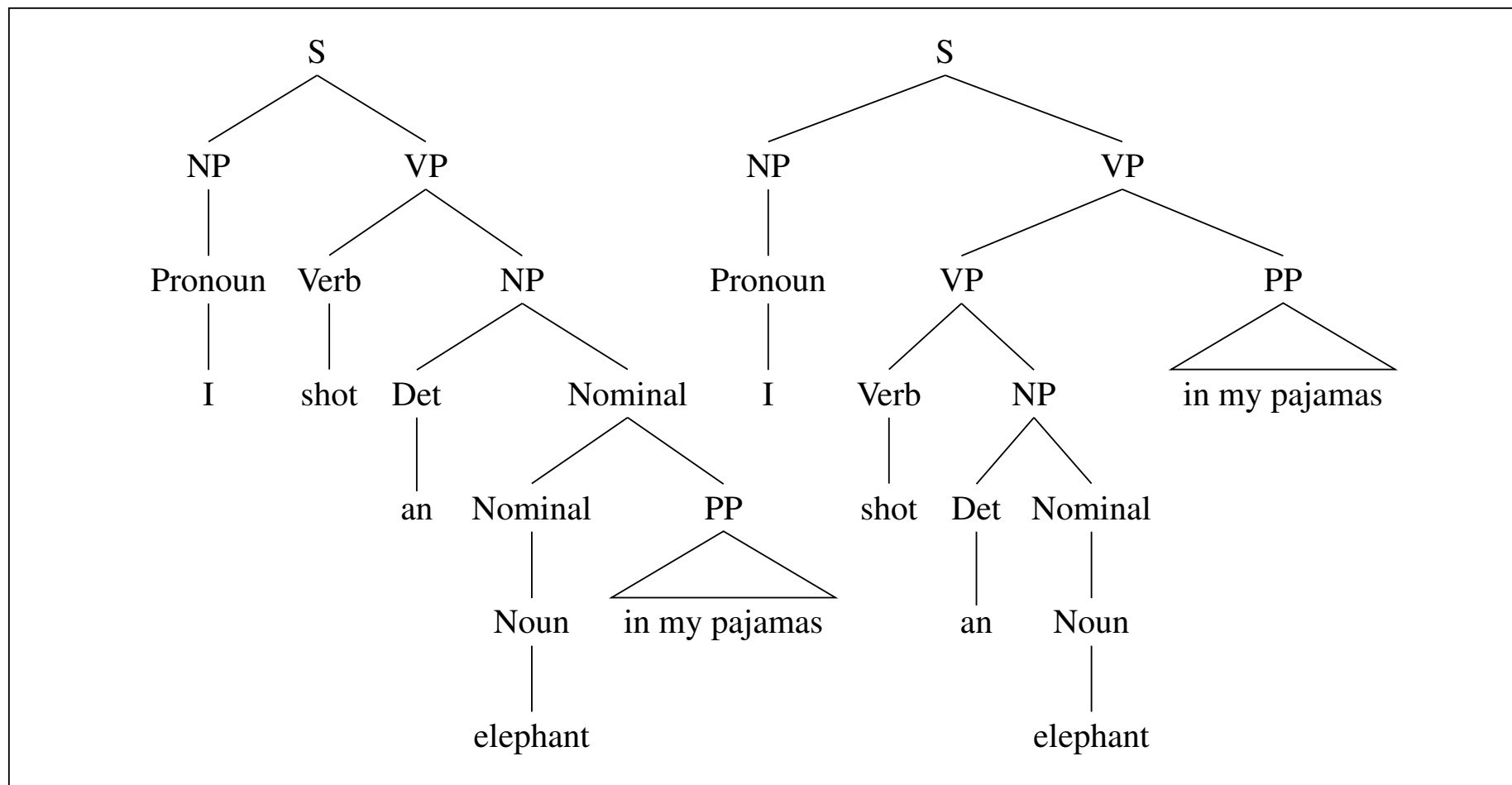
- “in my pyjamas” can be
  - part of a NP headed by “an elephant” (the elephant was in my pyjamas!)

or

- part of a VP headed by “shot”. (I shot the elephant whilst I was wearing my pyjamas)



# Challenges to Parsing: Attachment Ambiguity



**I shot an elephant in my pyjamas.**

An example of **PP-attachment ambiguity**.

# Challenges to Parsing:

## Attachment Ambiguity

“We saw the Eiffel Tower flying to Paris”

- “flying to Paris” is an example of a **VP-attachment** ambiguity, as it can have:

- have Eiffel Tower as subject (the Eiffel Tower flies!)

or:

- modify the verb “saw” (we saw the Eiffel Tower while we were flying to Paris)

**It does not always have to “make sense” semantically (for any type of ambiguity)**

# Challenges to Parsing: Coordination Ambiguity

**“old men and women dance.”**

- 1- (old men) and women dance.
- 2- (old (men and women)) dance.

President Kennedy today pushed aside other White House business to devote all his time and attention to working on the Berlin crisis address he will deliver tomorrow night to the American people over nationwide television and radio.

- ((nationwide tv) and (radio))
- (nationwide (tv and radio))

# Challenges to Parsing: Coordination Ambiguity

- The combination of these two ambiguities and the fact that the options do not have to make sense semantically, gives rise to a large number of options.

President Kennedy today pushed aside other White House business to devote all his time and attention to working on the Berlin crisis address he will deliver tomorrow night to the American people over nationwide television and radio.

# Challenges to Parsing: Syntactic Ambiguity

President Kennedy today pushed aside other White House business to devote all his time and attention to working on the Berlin crisis address he will deliver tomorrow night to the American people over nationwide television and radio.

Prepositional Phrase

- (to the American people) (over nationwide tv and radio)

# Challenges to Parsing: Syntactic Ambiguity

President Kennedy today pushed aside other White House business to devote all his time and attention to working on the Berlin crisis address he will deliver tomorrow night to the American people over nationwide television and radio.

## Prepositional Phrase

- (to the American people) (over nationwide tv and radio)
- (to (the American people over nationwide tv and radio))

# Challenges to Parsing:

## Syntactic Ambiguity

President Kennedy today pushed aside other White House business to devote all his time and attention to working on the Berlin crisis address he will deliver tomorrow night to the American people over nationwide television and radio.

### Prepositional Phrase

- (to the American people) (over nationwide tv and radio)
- (to (the American people over nationwide tv and radio))

**The American people who are over nationwide tv and radio?**

**Again, the parse does not always have to make sense...**

# Challenges to Parsing:

## Syntactic Ambiguity

**Show me the meals on the flight from San Fransisco.**



# Challenges to Parsing:

## Syntactic Ambiguity

**Show me the meals on the flight from San Fransisco.**

when/where should you show me the meals?

Show me the meals

on the flight from San Fransisco.

meals that come from where?

Show me the meals on the flight

from San Fransisco.

# Parsers

Different syntactic structures mean different meanings/semantics!

When developing parsers, we have to have the ambiguity challenge in mind. We want a parser that generates ALL possible parses.

# CFG Parsers

- Paradigm: parsing as search
- Searching through the space of possible parse trees to find the correct one: one whose root is S and whose leaves are exactly the words in the input sentence.
- Classic search algorithms:
  - 1- Top-Down
  - 2- Bottom-Up
  - 3- Dynamic Programming

# Top-Down

Given: a grammar and an input sentence

Start: the root of your parse tree: S

Continue: find all trees that can start with S

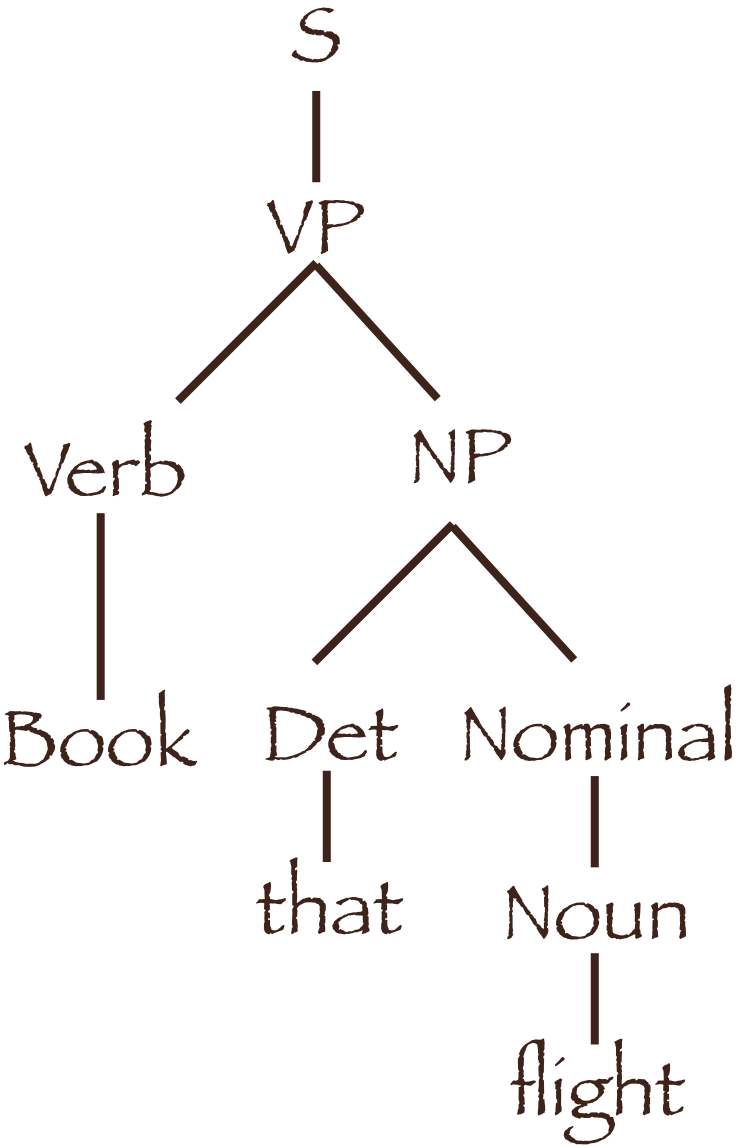
Method: look for rules with S on their left hand side

Repeat for each child

**Stop:** when the children are exactly the input words

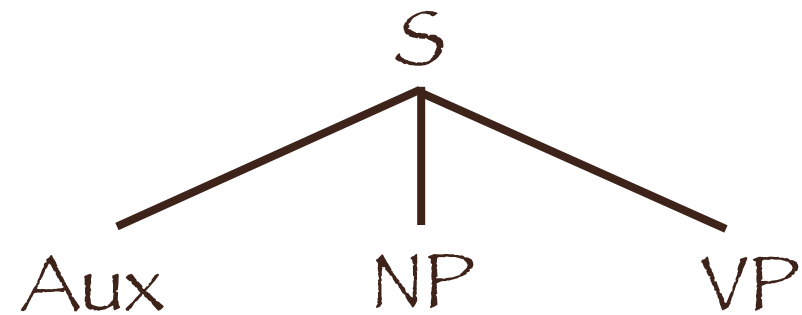
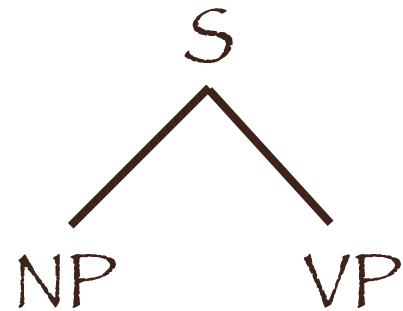
# Example

Grammar	Lexicon
$S \rightarrow NP VP$	$Det \rightarrow that \mid this \mid the \mid a$
$S \rightarrow Aux NP VP$	$Noun \rightarrow book \mid flight \mid meal \mid money$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid prefer$
$NP \rightarrow Pronoun$	$Pronoun \rightarrow I \mid she \mid me$
$NP \rightarrow Proper-Noun$	$Proper-Noun \rightarrow Houston \mid NWA$
$NP \rightarrow Det Nominal$	$Aux \rightarrow does$
$Nominal \rightarrow Noun$	$Preposition \rightarrow from \mid to \mid on \mid near \mid through$
$Nominal \rightarrow Nominal Noun$	
$Nominal \rightarrow Nominal PP$	
$VP \rightarrow Verb$	
$VP \rightarrow Verb NP$	
$VP \rightarrow Verb NP PP$	
$VP \rightarrow Verb PP$	
$VP \rightarrow VP PP$	
$PP \rightarrow Preposition NP$	



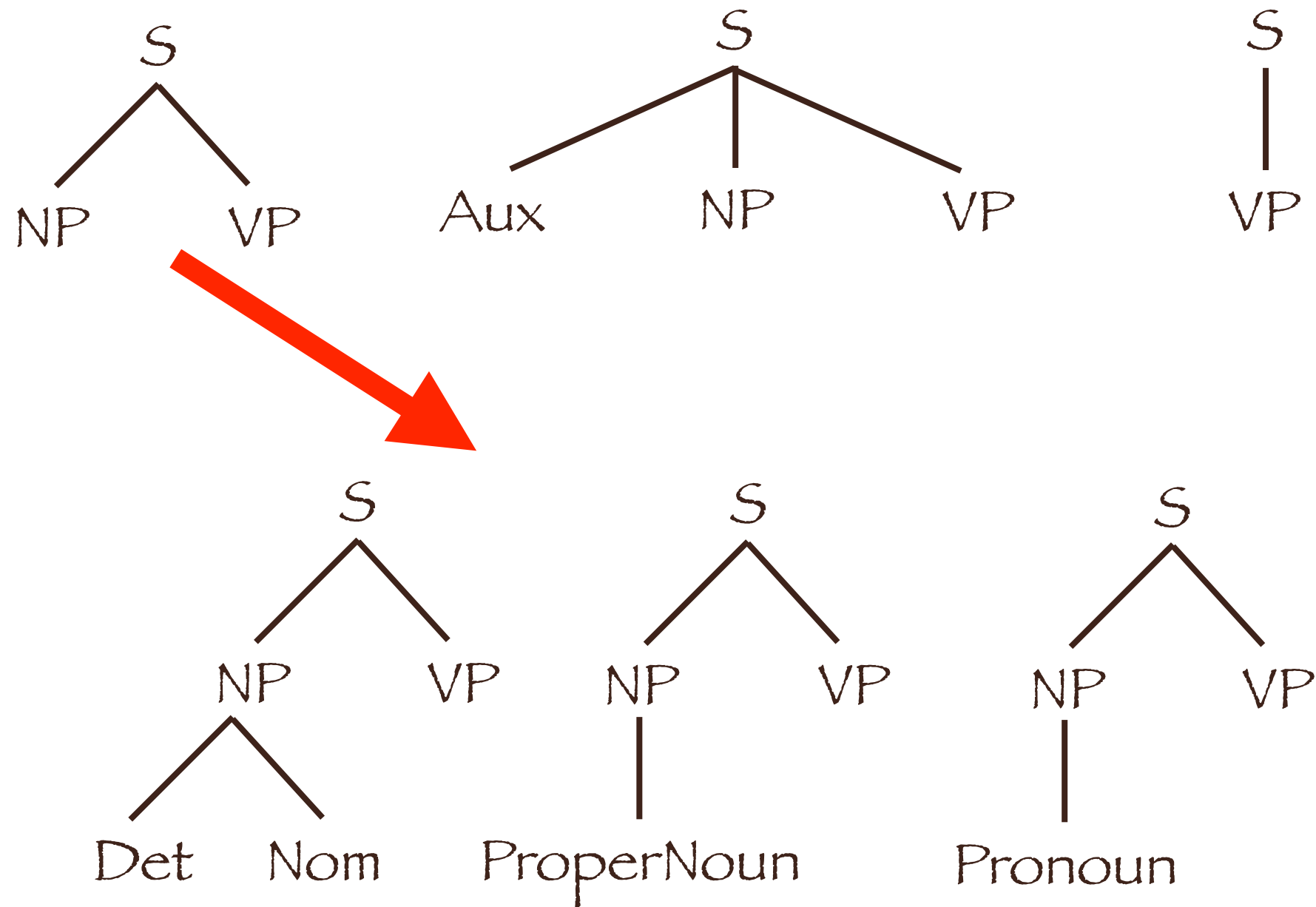
Book that flight.

# Search Space



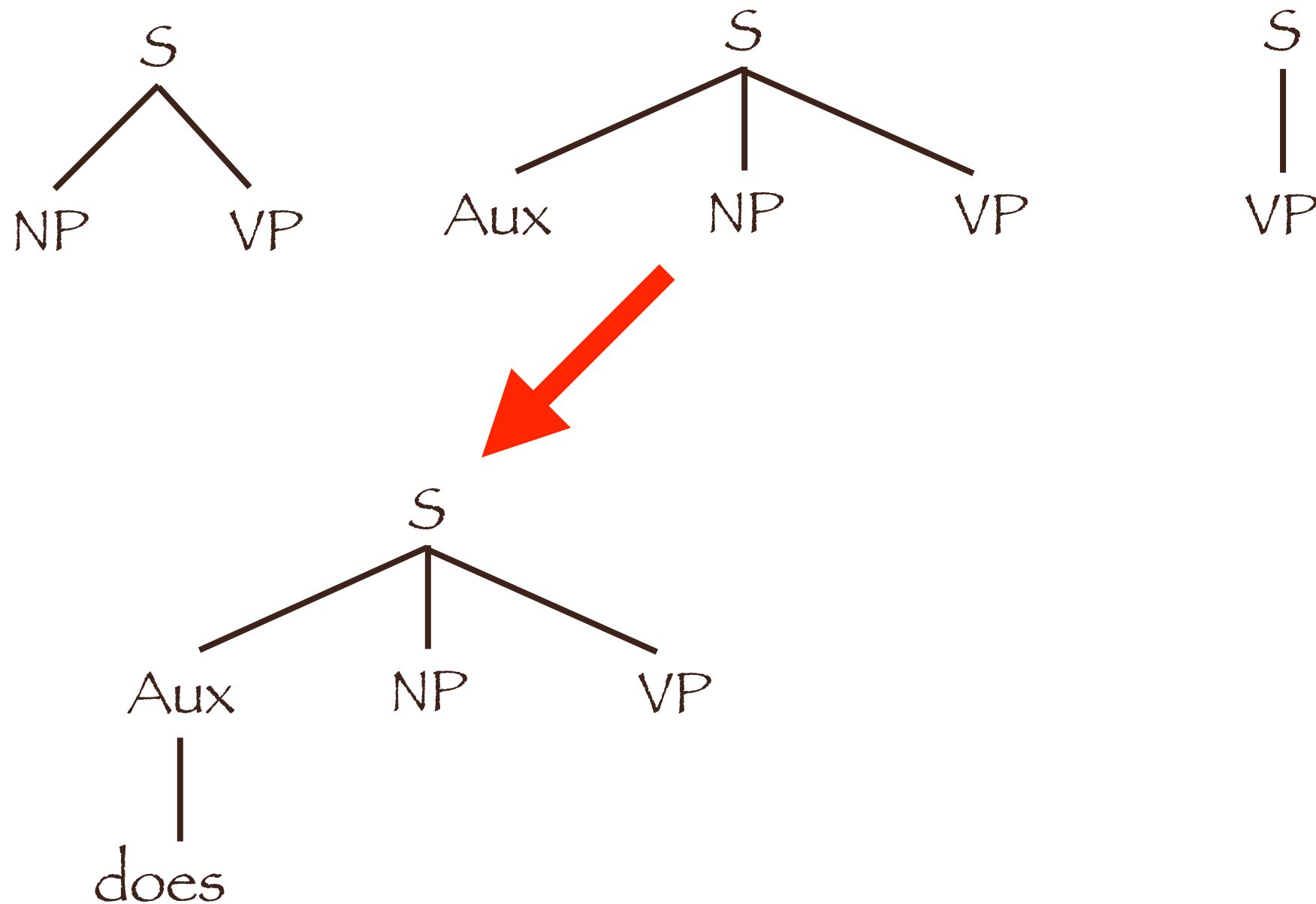
Book that flight.

# Search Space



Book that flight.

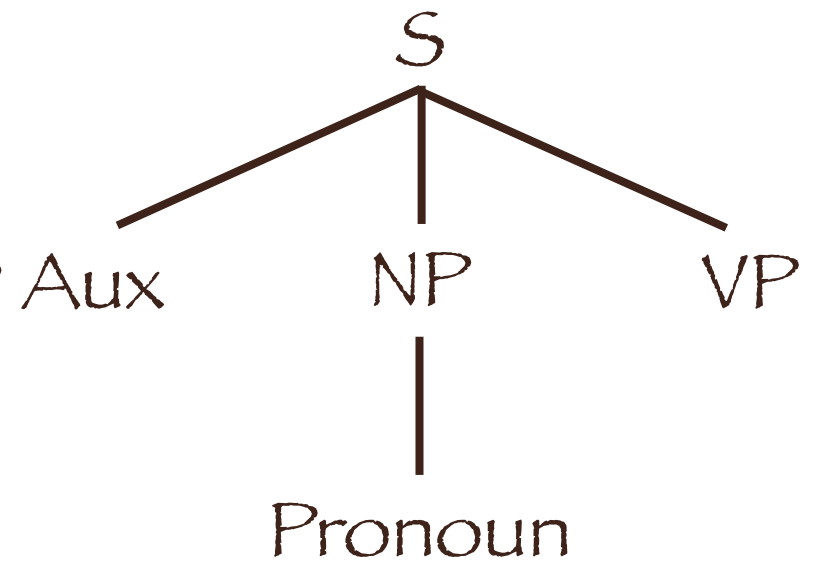
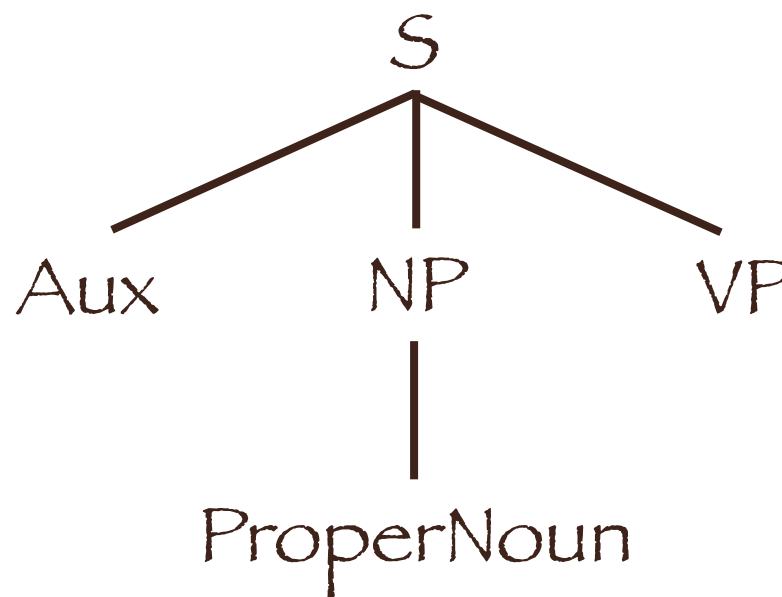
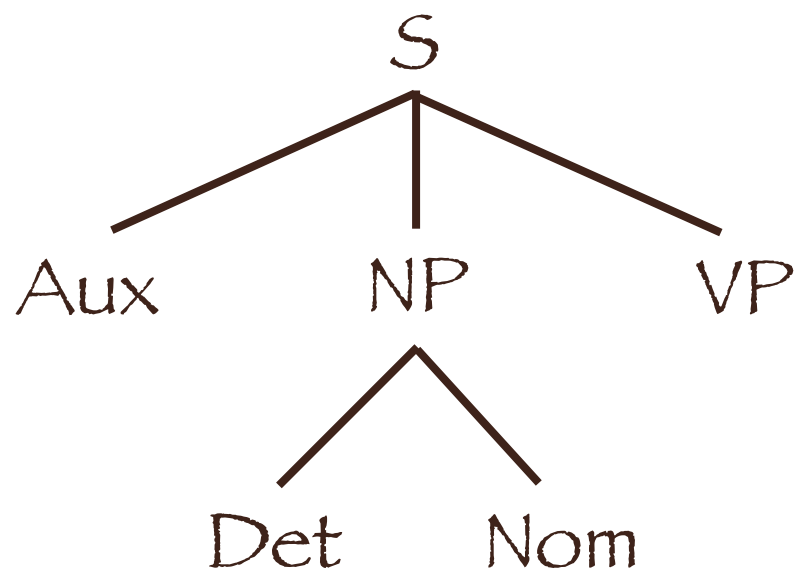
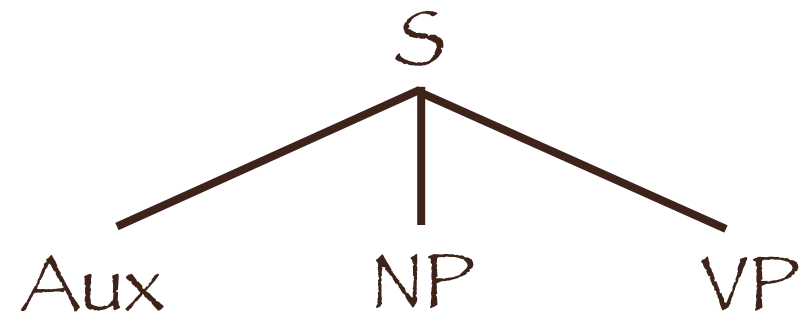
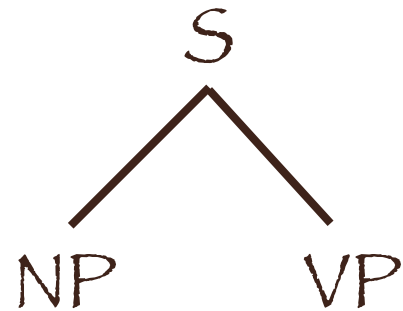
# Search Space



Book that flight.

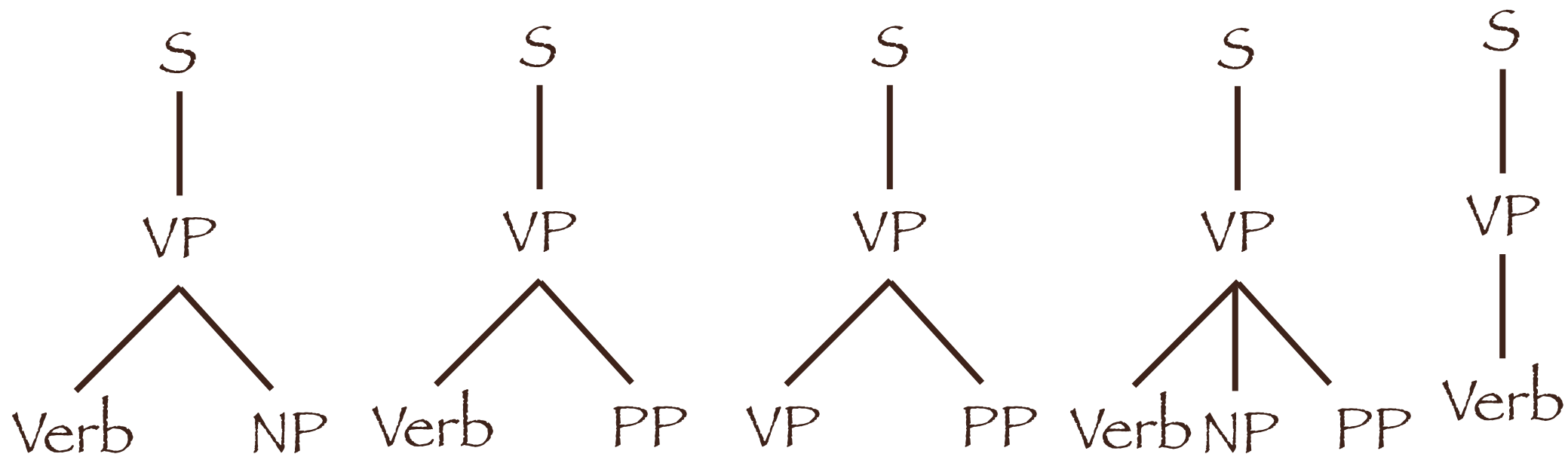
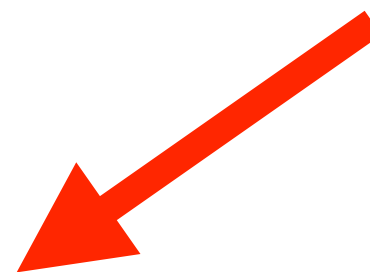
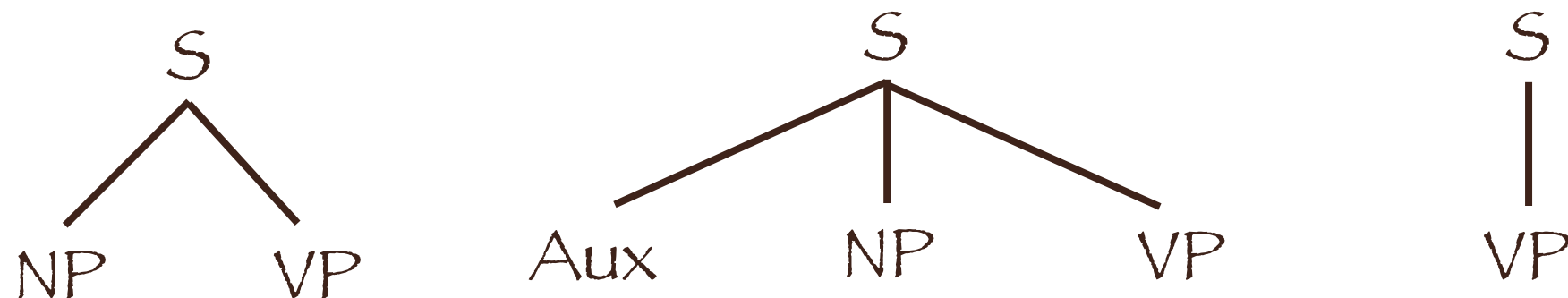


# Search Space



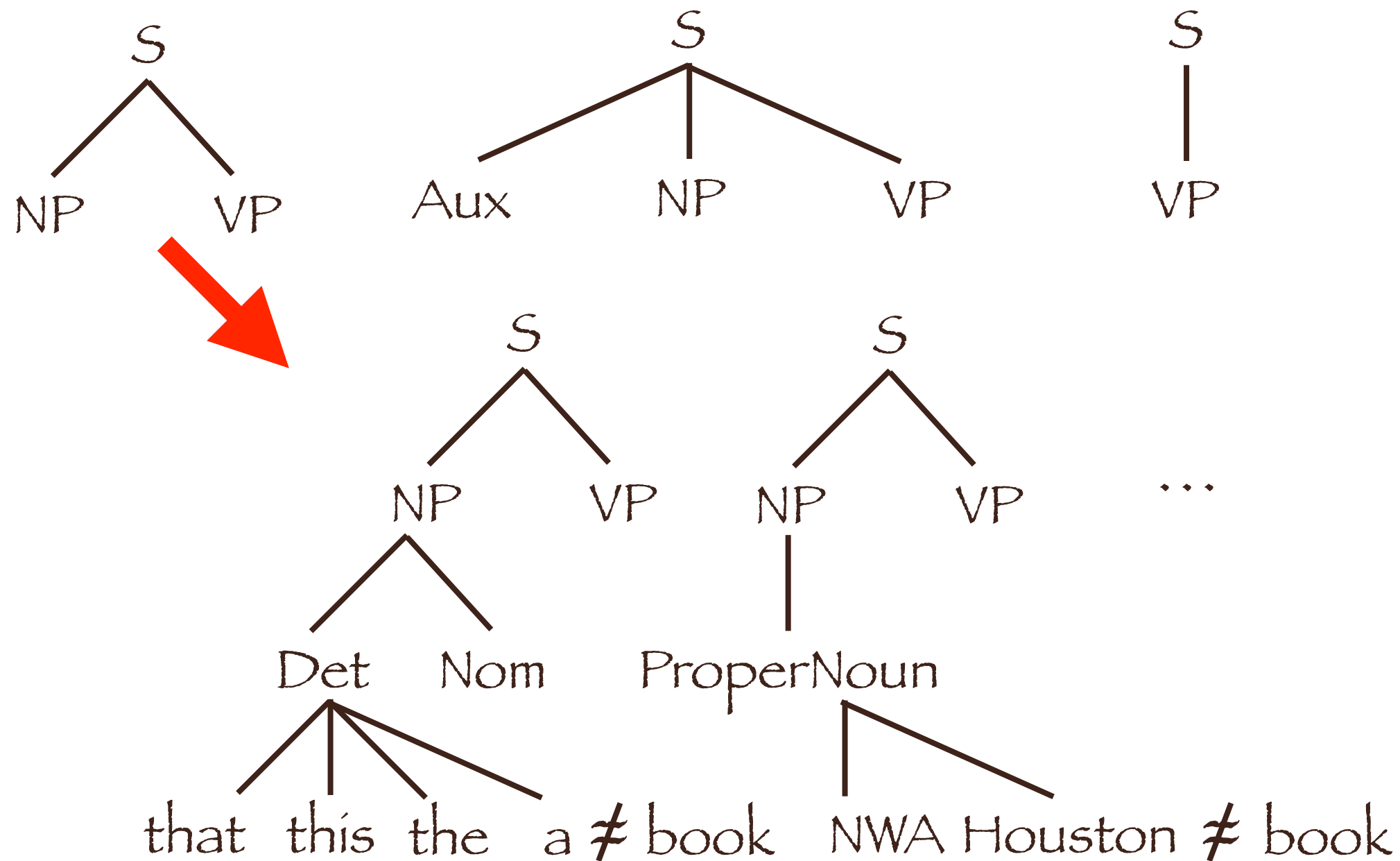
Book that flight.

# Search Space



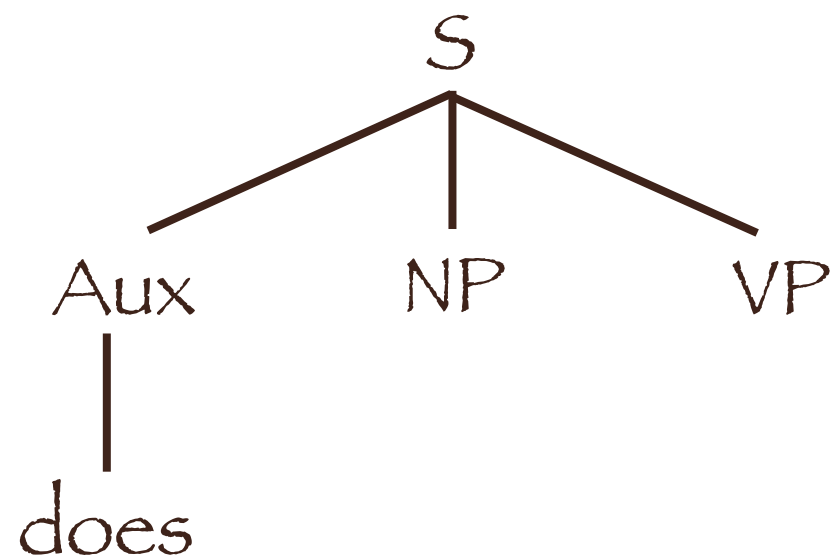
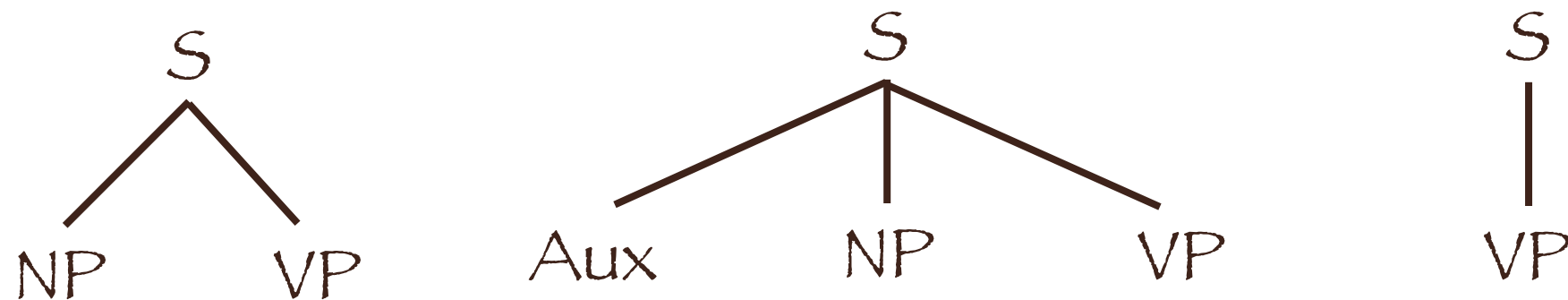
Book that flight.

# Search Space



Book that flight.

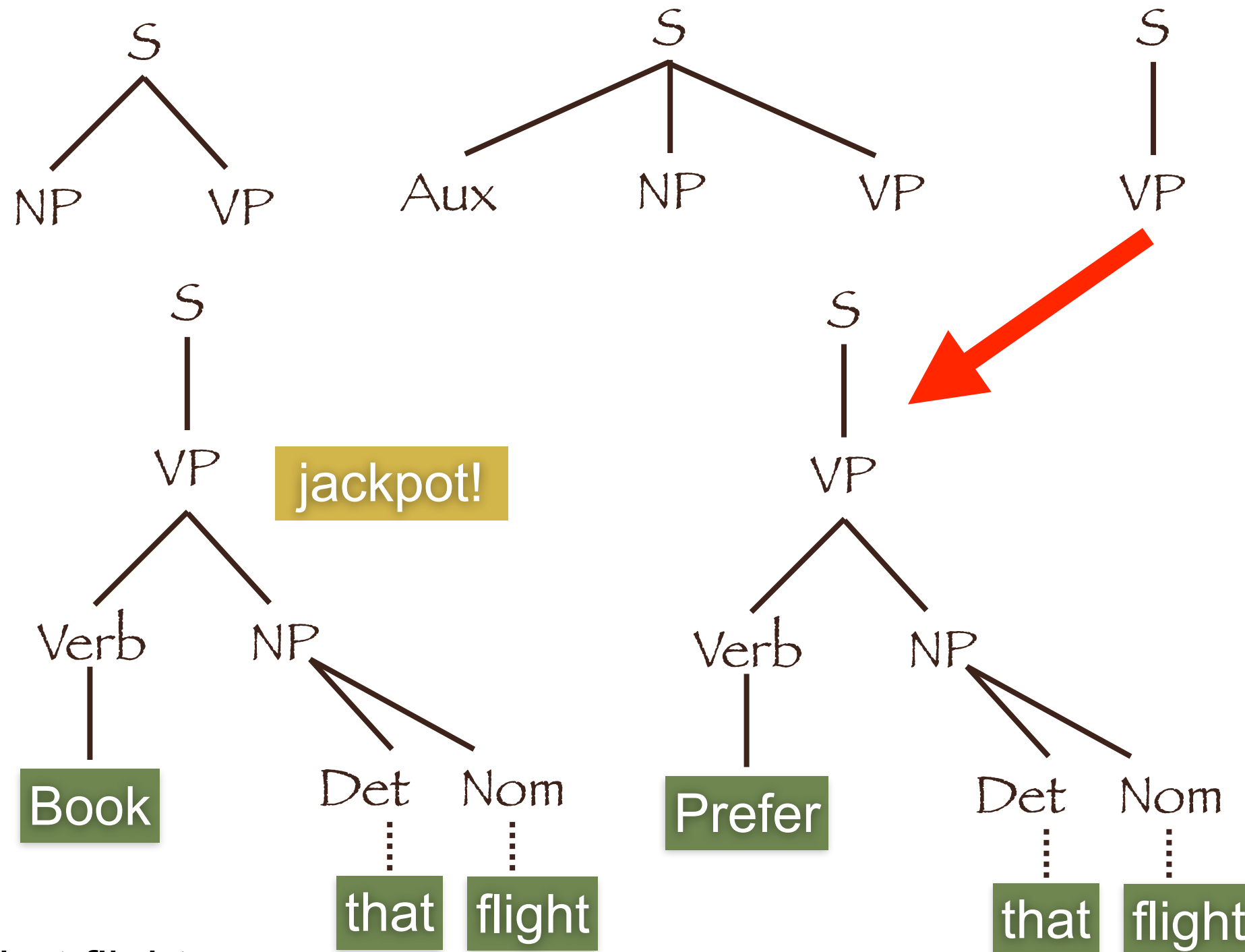
# Search Space



≠ book

Book that flight.

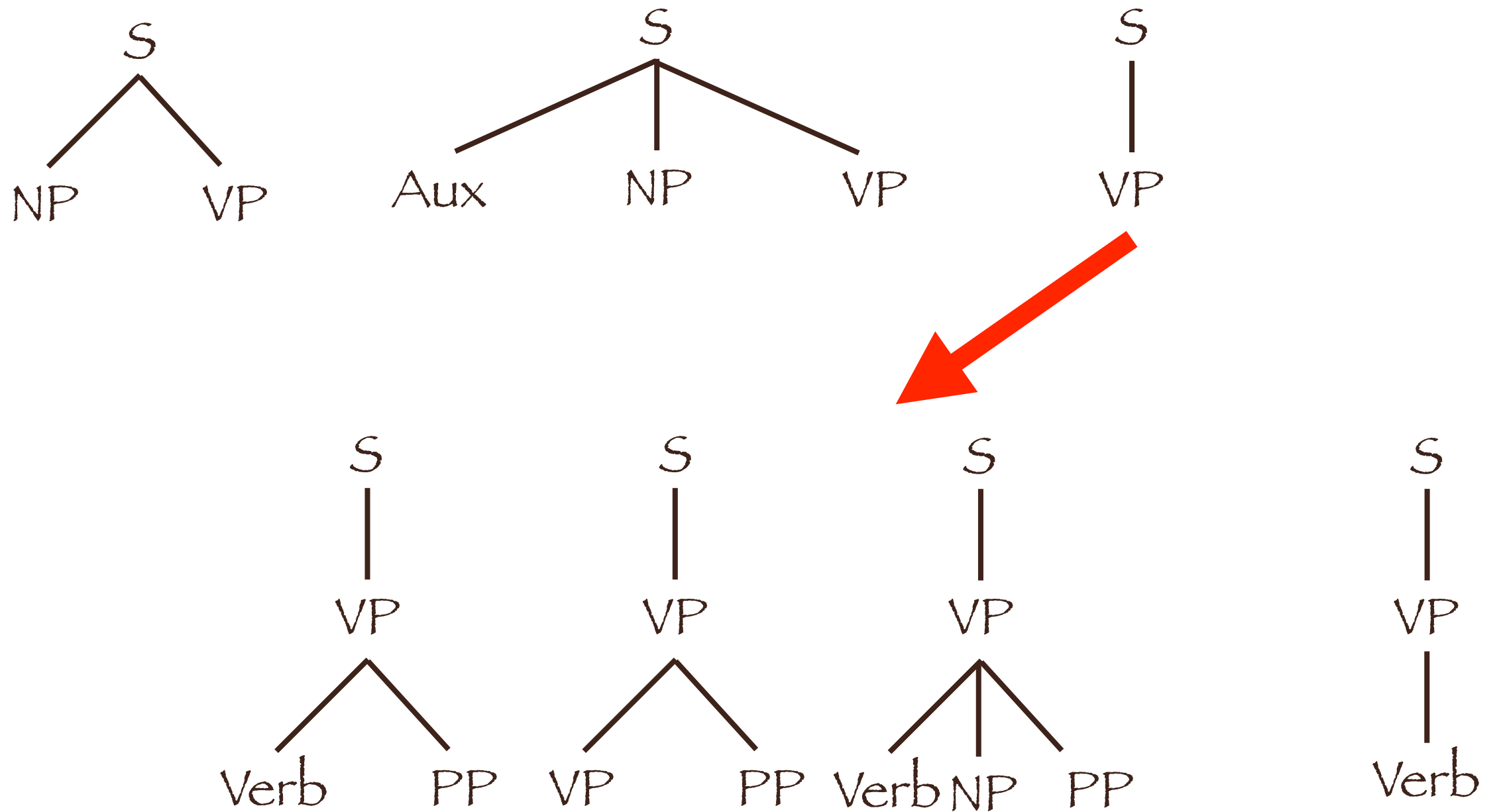
# Search Space



Book that flight.

... but  
anything  
else is  
bad.

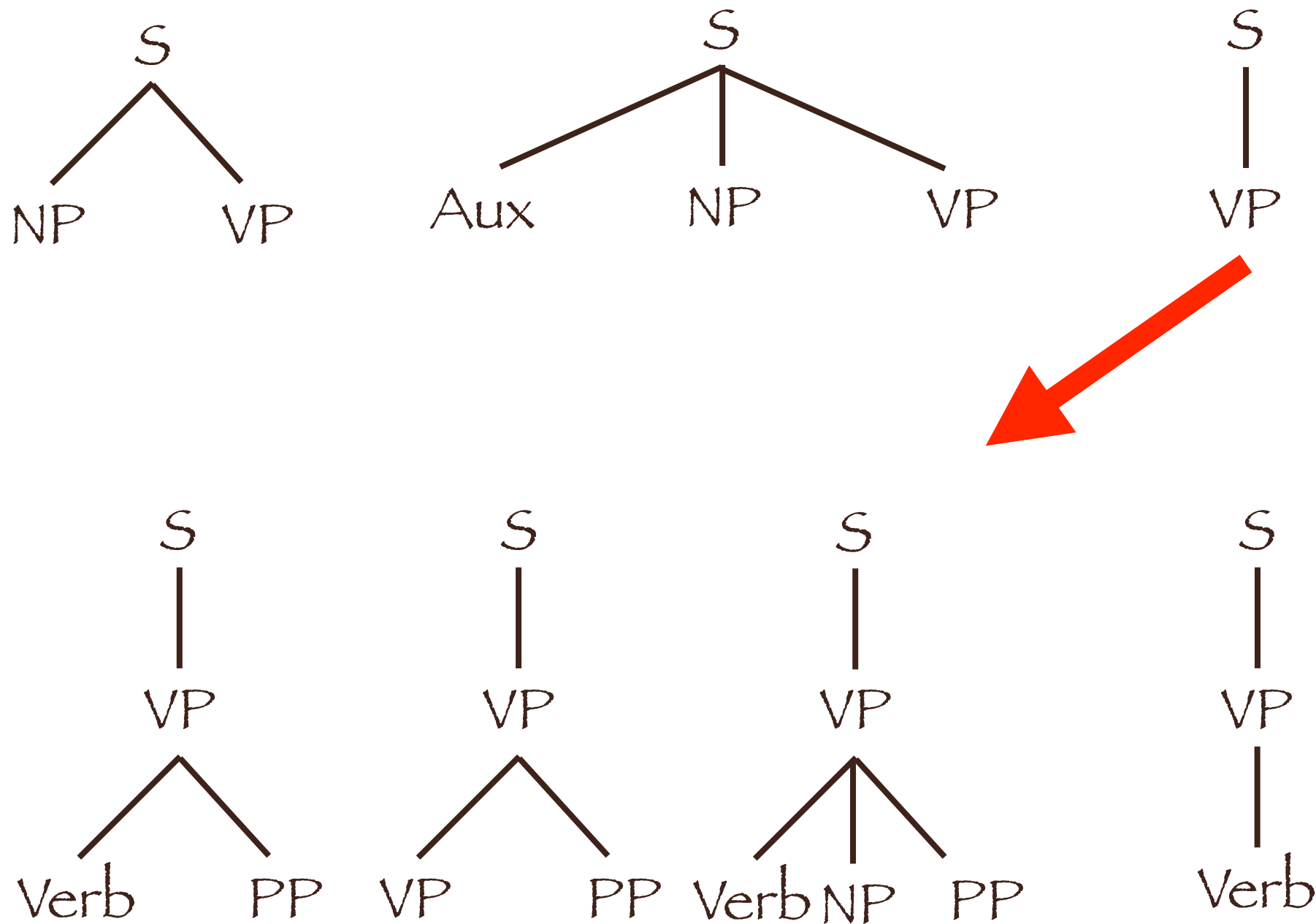
# Search Space



These 3 are bad: there are no PP's in S.

Book that flight.

# Search Space



This is bad: we have more than a verb in *S*.

Book that flight.

# Bottom-Up

Start: words of the sentence we are parsing

Continue: find all trees that can start with words

Method: look for rules with words on their right hand side

Repeat for each child

Stop: a tree with root S



# Search Space

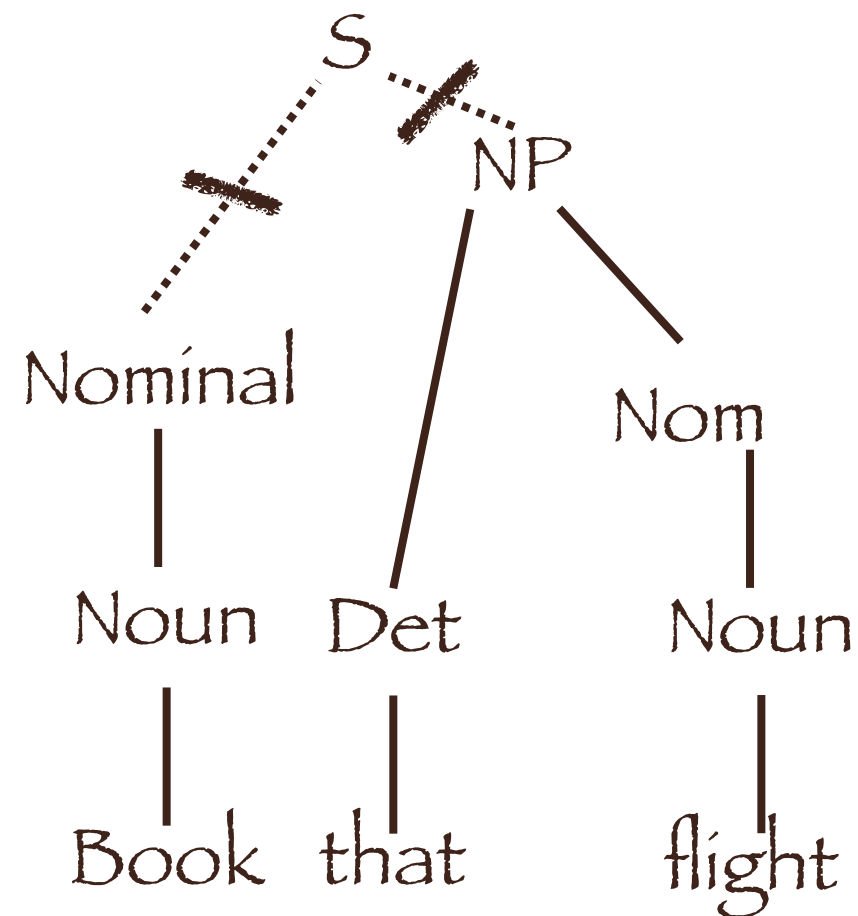
Book that flight.

Noun	Det	Noun
Book	that	flight

Verb	Det	Noun
Book	that	flight

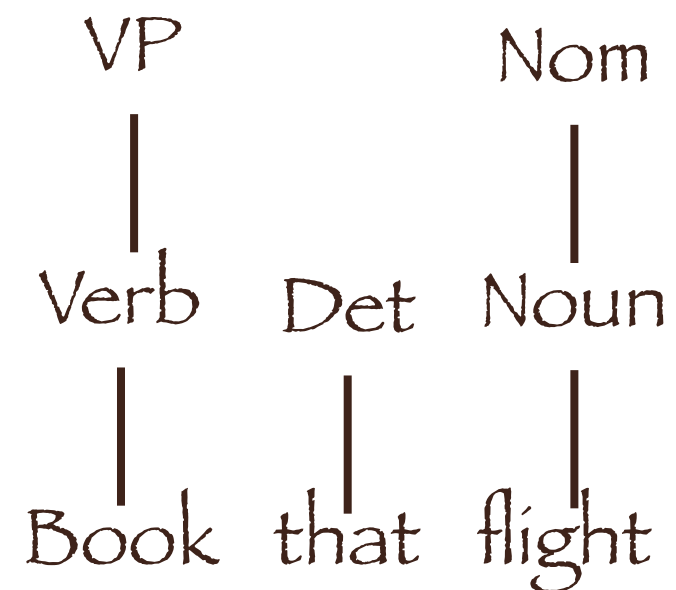
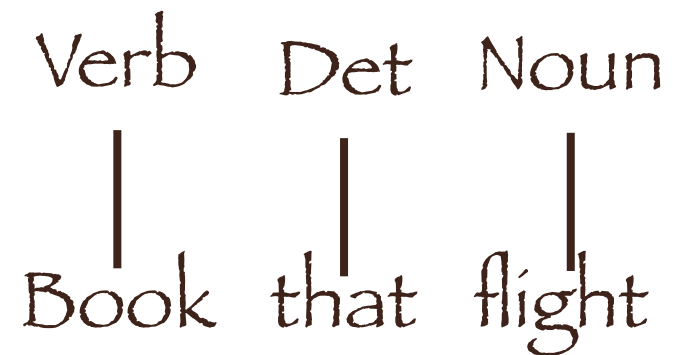
# Search Space

Book that flight.



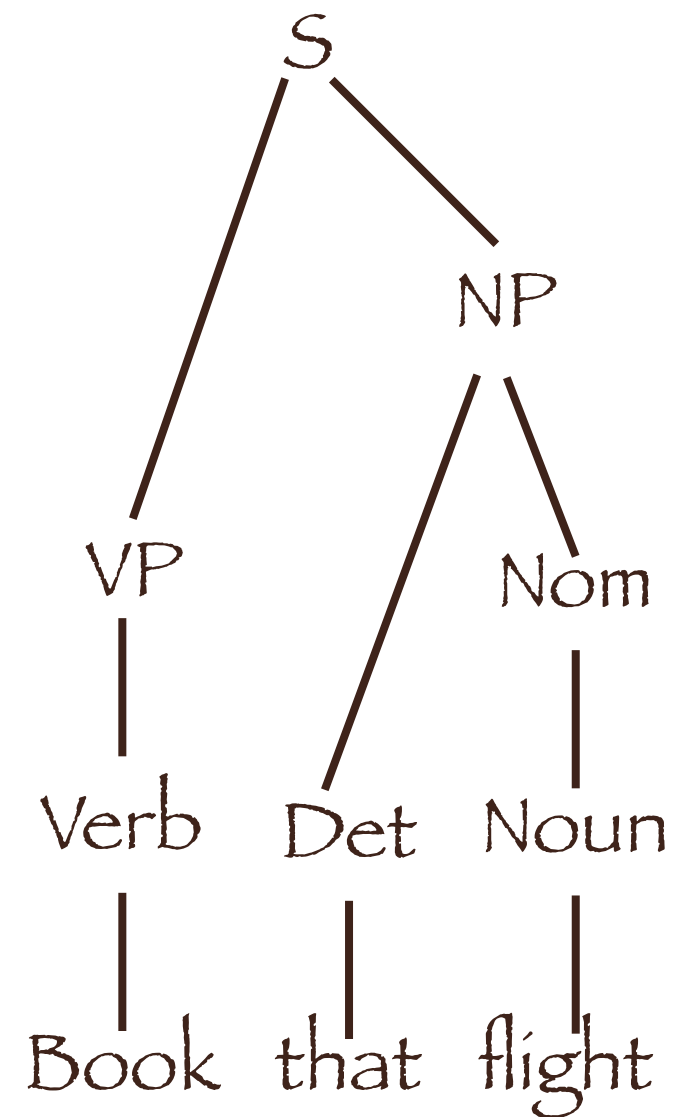
# Search Space

Book that flight.



# Search Space

Book that flight.



# Which Method?

- Each has their own advantages and disadvantages
- The Top-Down will never waste time with trees that cannot result in S.
- The Bottom-Up will not waste time with trees that cannot end in the words of input.
- Which one is worse? In our example, Bottom-Up seemed to be better. But this could be accidental.

# Parsing Algorithm: CKY

- CKY (or CYK) algorithm was introduced by different people. This version is due to:
  - Cocke
  - Younger
  - Kasami
- It is also called “chart parsing”.
- It stores constituents and subtrees as it finds them, so they do not need to be reconstructed again.

# Chomsky Normal Form




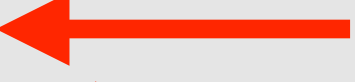

- To use CKY, one has to work with grammars that are in **Chomsky Normal Form**. This is when each rule expands to either of the following forms:
  - only two non-terminals
  - one single single terminal

$$A \rightarrow B C \quad A \rightarrow a$$

$$A, B, C, \in N \quad a \in \Sigma$$

- If we work with these grammars have we lost expressive power? Nothing is lost!
  - Theorem: *A CFG and its ChNF accept the same language.*

# Example of a Grammar and its ChNF

$\mathcal{L}_1$ Grammar	$\mathcal{L}_1$ in CNF
$S \rightarrow NP VP$	$S \rightarrow NP VP$
$S \rightarrow Aux NP VP$	$S \rightarrow X1 VP$ 
	$X1 \rightarrow Aux NP$ 
$S \rightarrow VP$	$S \rightarrow book \mid include \mid prefer$
	$S \rightarrow Verb NP$
	$S \rightarrow X2 PP$ 
	$S \rightarrow Verb PP$
	$S \rightarrow VP PP$
$NP \rightarrow Pronoun$	$NP \rightarrow I \mid she \mid me$
$NP \rightarrow Proper-Noun$	$NP \rightarrow TWA \mid Houston$
$NP \rightarrow Det Nominal$	$NP \rightarrow Det Nominal$
$Nominal \rightarrow Noun$	$Nominal \rightarrow book \mid flight \mid meal \mid money$
$Nominal \rightarrow Nominal Noun$	$Nominal \rightarrow Nominal Noun$
$Nominal \rightarrow Nominal PP$	$Nominal \rightarrow Nominal PP$
$VP \rightarrow Verb$	$VP \rightarrow book \mid include \mid prefer$
$VP \rightarrow Verb NP$	$VP \rightarrow Verb NP$
$VP \rightarrow Verb NP PP$	$VP \rightarrow X2 PP$ 
	$X2 \rightarrow Verb NP$ 
$VP \rightarrow Verb PP$	$VP \rightarrow Verb PP$
$VP \rightarrow VP PP$	$VP \rightarrow VP PP$
$PP \rightarrow Preposition NP$	$PP \rightarrow Preposition NP$



# Chomsky Normal Form

- Why ChNF?
  - The parse trees are now all binary!
  - So they can be denoted by 2 dimensional matrices.
    - More specifically, the upper triangle of an  $(n+1) \times (n+1)$  matrix.
    - Where  $n$  is the number of words in the input sentence.

# Parsing Algorithm: CKY

The cell representing the entire input is [0,n].

How do we fill in the matrix?

1- Index your input sentence by inserting a number before and after each word:

0 Book 1 the 2 flight 3 through 4 Houston 5

<i>Book</i>	<i>the</i>	<i>flight</i>	<i>through</i>	<i>Houston</i>
S, VP, Verb Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	S,VP,X2 [0,5]
	Det [1,2]	NP [1,3]	[1,4]	NP [1,5]
		Nominal, Noun [2,3]	[2,4]	Nominal [2,5]
			Prep [3,4]	PP [3,5]
				NP, Proper- Noun [4,5]

# Parsing Algorithm: CKY

The main diagonal has the constituencies of the words of the sentence:

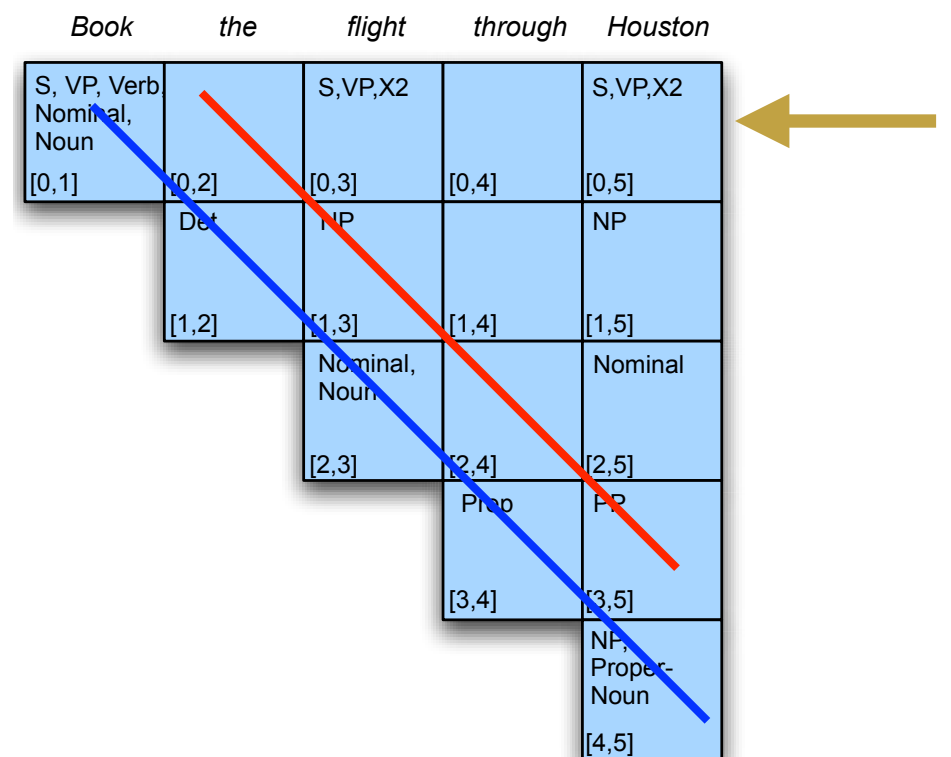
**[0,1]Book,**  
**[1,2]the,**  
**[2,3]flight,**  
**[3,4]through,**  
**[4,5]Houston.**

<i>Book</i>	<i>the</i>	<i>flight</i>	<i>through</i>	<i>Houston</i>
S, VP, Verb Nominal, Noun [0,1]	[0,2] Det	S,VP,X2 [0,3] NP	[0,4] Nominal, Noun	S,VP,X2 [0,5] Nominal
	[1,2]	[1,3] Nominal, Noun	[1,4] Prep	[1,5] PP
		[2,3]	[2,4] NP, Proper- Noun	[2,5]
			[3,4]	[3,5]
				[4,5]

# Parsing Algorithm: CKY

The second diagonals cover constituencies of combinations of two words:

**[0,2]Book the,**  
**[1,3]the flight,**  
**[2,4]flight through,**  
**[3,5] through Houston.**

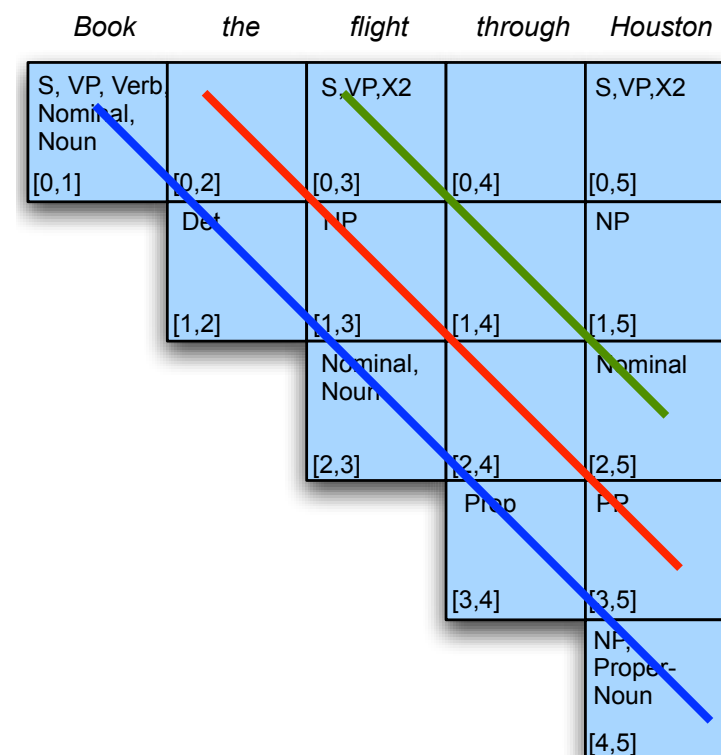


# Parsing Algorithm: CKY

The cell representing the entire input is  $[0,n]$ .

The third diagonals cover constituencies of combinations of three words:

**$[0,3]$ Book the flight,**  
 **$[1,4]$ the flight through,**  
 **$[2,5]$ flight through Houston,**



# Parsing Algorithm: CKY

and so on ...

The cell representing the entire input is  $[0,n]$ .

The last diagonal has the constituency of the full sentence:

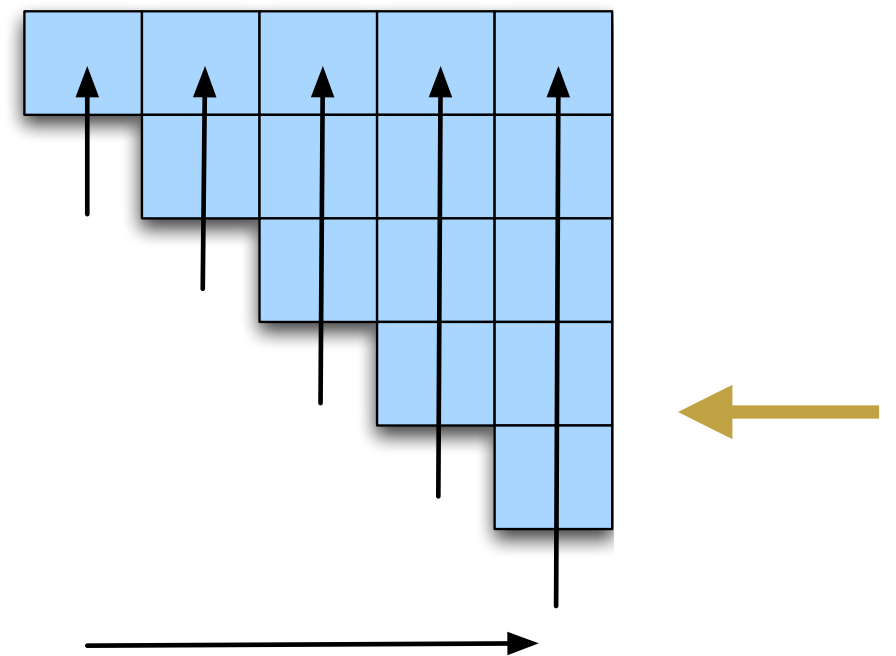
**[0,5] Book the flight through Houston**

<i>Book</i>	<i>the</i>	<i>flight</i>	<i>through</i>	<i>Houston</i>
S, VP, Verb Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	S,VP,X2 [0,5]
	Det [1,2]	NP [1,3]	[1,4]	NP [1,5]
		Nominal, Noun [2,3]	[2,4]	Nominal [2,5]
			Prep [3,4]	PP [3,5]
				NP, Proper- Noun [4,5]

# Parsing Algorithm: CKY

The matrix is filled in a bottom-up fashion:

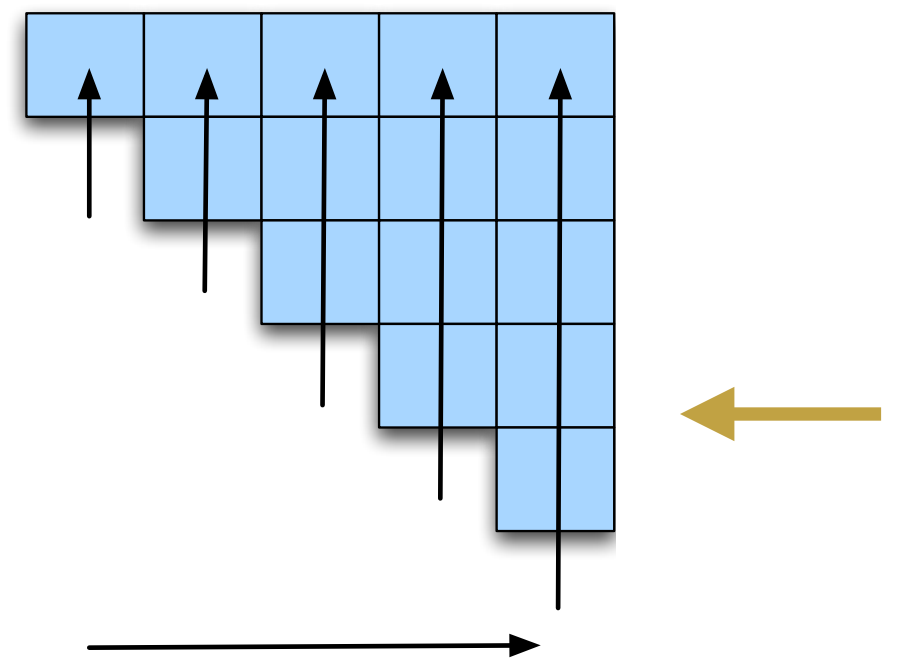
- fill the matrix a column at a time from the left.
- fill each column from bottom to top.



# Parsing Algorithm: CKY

The matrix is filled in a bottom-up fashion:

- fill the matrix a column at a time from the left.
- fill each column from bottom to top.



Why?

1- At each cell, we have all the info we need.

2- This is how incremental, word-by-word parsing (which is what humans are supposed to do) works.



# Parsing Algorithm: CKY

How do we fill in the matrix?

0 Book 1 the 2 flight 3 through 4 Houston 5

Each cell (i,j) of matrix is filled with a set of non terminals, representing all constituents from position i to j.

The cell representing the entire input is [0,n].

Book	the	flight	through	Houston
S, VP, Verb Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	S,VP,X2 [0,5]
	Det [1,2]	NP [1,3]	[1,4]	NP [1,5]
		Nominal, Noun [2,3]	[2,4]	Nominal [2,5]
			Prep [3,4]	PP [3,5]
				NP, Proper- Noun [4,5]

# Parsing Algorithm: CKY

How do we fill in the matrix?

0 Book 1 the 2 flight 3 through 4 Houston 5

Each cell (i,j) of matrix is filled with a set of non terminals, representing all constituents from position i to j.

Consider the cell (0,1). Fill it with the non terminals that represent the constituency of the word from position 0 to 1, i.e. “Book”.

The cell representing the entire input is [0,n].

Book	the	flight	through	Houston
S, VP, Verb Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	S,VP,X2 [0,5]
	Det [1,2]	NP [1,3]	[1,4]	NP [1,5]
		Nominal, Noun [2,3]	[2,4]	Nominal [2,5]
			Prep [3,4]	PP [3,5]
				NP, Proper- Noun [4,5]

# Parsing Algorithm: CKY

How do we fill in the matrix?

0 Book 1 the 2 flight 3 through 4 Houston 5

Each cell (i,j) of matrix is filled with a set of non terminals, representing all constituents from position i to j.

Consider the cell (0,1). Fill it with the non terminals that represent the constituency of the word from position 0 to 1, i.e. “Book”.

“**Book**” can be a imperative sentence S, it can be a VP (book a seat), or just a Verb (book from home). It can also be part of a group of nouns (book worm)- so a Nominal, or only one Noun.

The cell representing the entire input is [0,n].

Book	the	flight	through	Houston
S, VP, Verb Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	S,VP,X2 [0,5]
	Det [1,2]	NP [1,3]	[1,4]	NP [1,5]
		Nominal, Noun [2,3]	[2,4]	Nominal [2,5]
			Prep [3,4]	PP [3,5]
				NP, Proper- Noun [4,5]

# Parsing Algorithm: CKY

We have the constituents, but haven't recorded the structural relation they are in to each other as we parse.

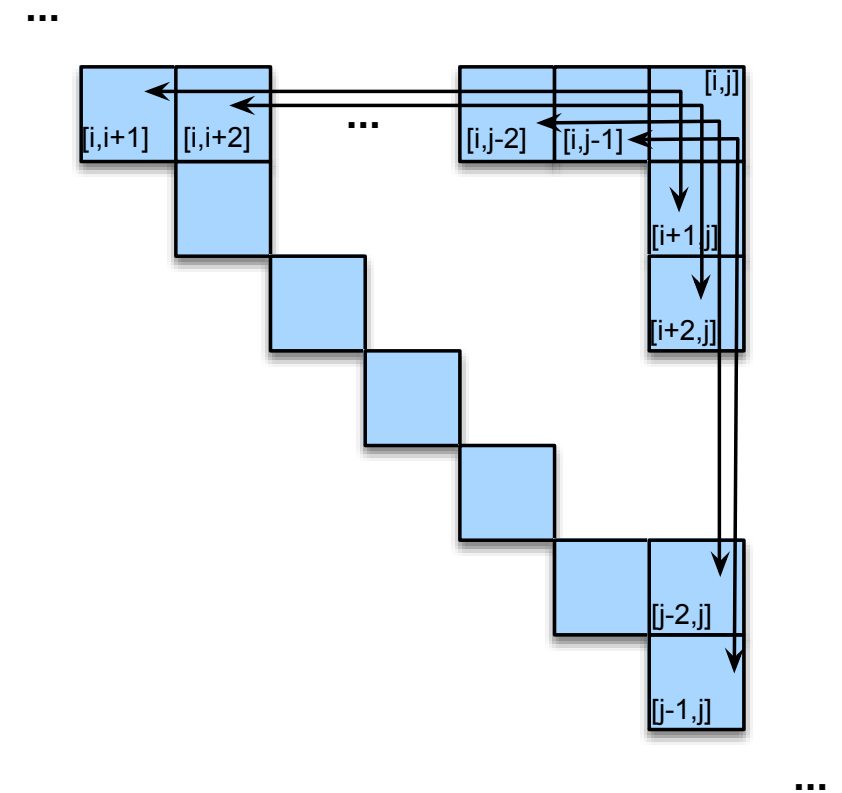
We need to record these to get the different parses/syntactic trees.

# Parsing Algorithm: CKY

After filling the table, start with the S cell of the table, cell (0,n). In the presence of ambiguity, we want to return all possible parses of S.

Make each non-terminal point to cells from which it was derived in the whole matrix.

Now all parses are returned by choosing the S from cell [0,n] and retrieving all of its component constituents from the matrix, by following the pointers.



# Examples

[3,5] points to [3,4] and [4,5].

This means that the PP in [3,5], i.e.  
 “through Houston” can be generated by  
 Prep NP  
 or  
 Prep Proper-Noun

In this case, it is Prep Proper-Noun

<i>Book</i>	<i>the</i>	<i>flight</i>	<i>through</i>	<i>Houston</i>
S, VP, Verb, Nominal, Noun [0,1]	[0,2]	S,VP,X2 [0,3]	[0,4]	[0,5]
	Det [1,2]	NP [1,3]	[1,4]	NP [1,5]
		Nominal, Noun [2,3]	[2,4]	[2,5]
			Prep ← [3,4]	PP ↓ [3,5]
				NP, Proper- Noun [4,5]

# Examples

[2,5] points to [2,3] and [3,5].

This means that the Nominal in [2,5],  
i.e. “flight through Houston” can be  
generated by  
Nominal PP  
or  
Noun PP

In this case, it is Noun PP

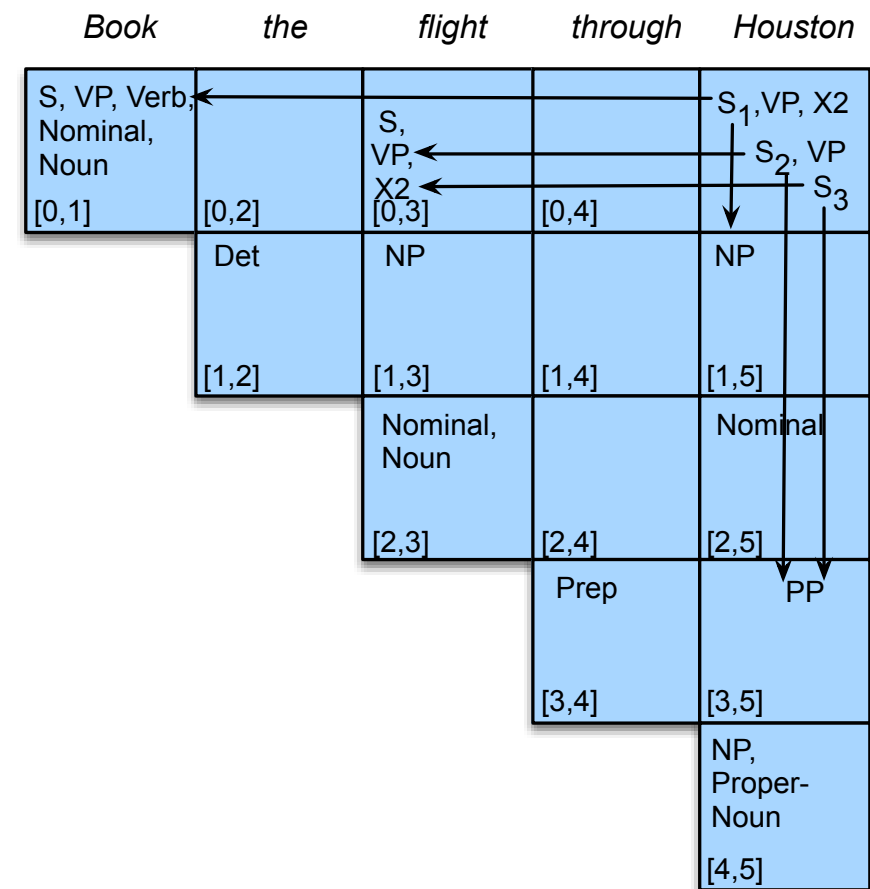
<i>Book</i>	<i>the</i>	<i>flight</i>	<i>through</i>	<i>Houston</i>
S, VP, Verb, Nominal, Noun [0,1]	[0,2]	S, VP, X2 [0,3]	[0,4]	[0,5]
	Det [1,2]	NP [1,3]	[1,4]	NP [1,5]
		Nominal, Noun [2,3]	[2,4]	Nominal [2,5]
			Prep [3,4]	PP [3,5]
				NP, Proper- Noun [4,5]

# Examples

and so on ... in particular ...

The S1 in [0,5] points to Verb in [0,1] and NP in [1,5], which gives the parse Verb NP (the correct one).

Book, which flight? the flight that goes through Houston.

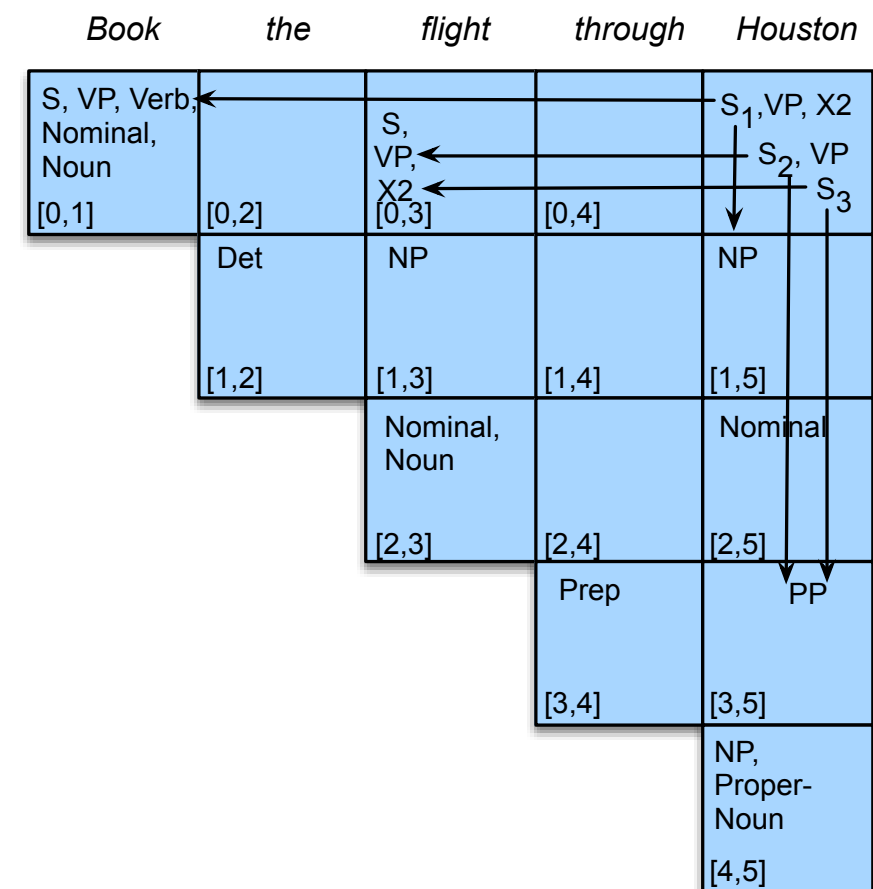




# Examples

The S2 in [0,5] points to VP in [0,3] and PP in [3,5], which gives the parse VP PP (which is slightly wrong).

Book the flight, how? through Houston

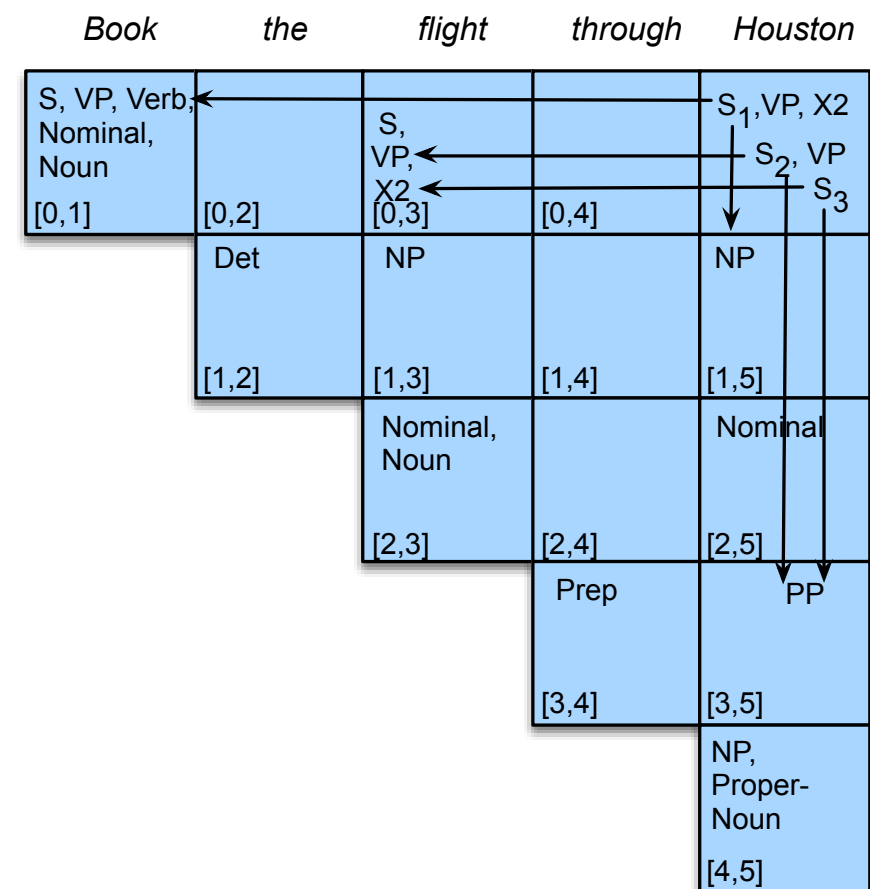


# Examples

The S2 in [0,5] points to VP in [0,3] and PP in [3,5], which gives the parse VP PP (which is slightly wrong).

Book the flight, how? through Houston

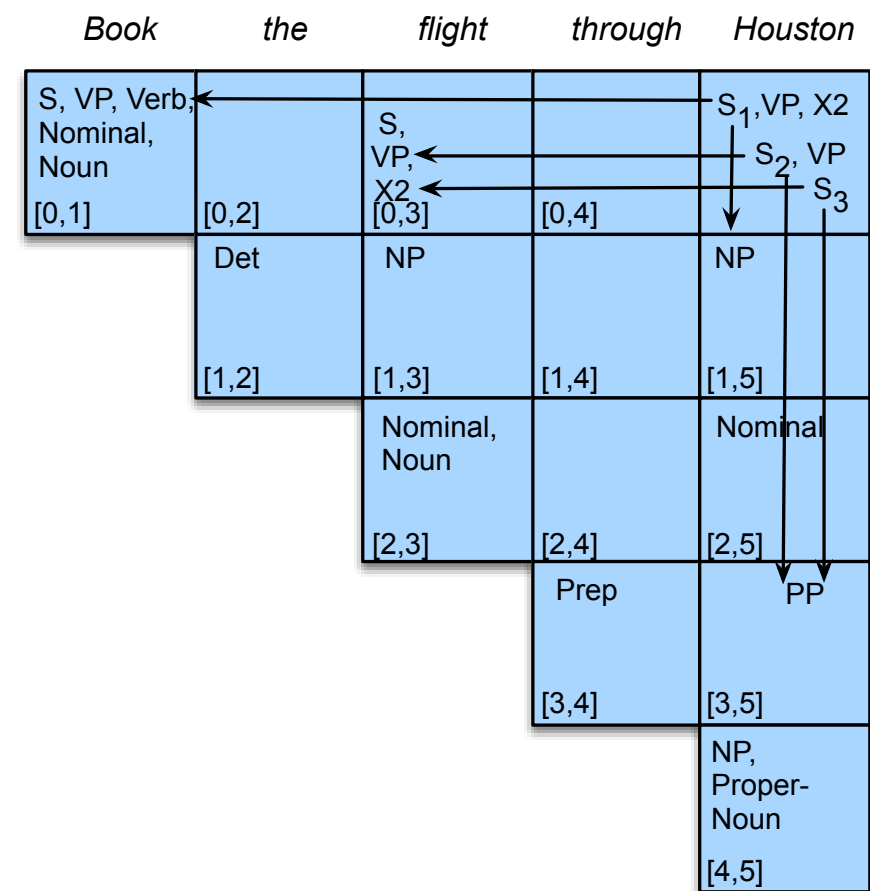
Can you do the S3?



# Examples

The S3 in [0,5] points to the X2 (Verb NP) in [0,3] and the PP in [3,5].  
This gives the parse Verb NP PP (also slightly wrong).

Book the flight. How? through  
Houston, e.g. while passing through  
Houston, etc.



# Dependency Grammars

# Recap: Generative and Logical Grammars

- In generative (or phrase-structure) grammars, the basic construct is that of phrases of languages and the relationships between them.
- For instance, for CFGs we have phrases such as:

NP, VP, PP

- and that they are connected to each other via rules such as:

S → NP VP  
VP → Verb NP PP

# Shortcomings: Generative and Logical Grammars

- Generative grammars have been extensively used by linguistics to develop extensive theories of language constructs. But their drawbacks are ...
- 1- Their phrase structures are not very intuitive to non-linguists.
- 2- More importantly, they differ hugely from one another for different languages.
  - For instance for language with a different word order, one has to sit down and develop a whole new set of rules.

# Shortcomings: Generative and Logical Grammars

- In **Farsi**, the sentence structure is Sbj-Obj-Verb (SOV), as opposed to in English where it is Sbj-Verb-Obj (SVO). So we have to change our  $S \rightarrow NP VP$  rule to something like:

$$S \rightarrow NP \ NP \ VP$$

where

$$VP \rightarrow \text{verb}$$
$$NP \rightarrow \text{Nom PP} \mid \text{PP Nom}$$

# Shortcomings: Generative and Logical Grammars

- In languages with a relatively free word order such as **Czech**, one is in more trouble and has to add a rule for every possible location of a phrase. That is:

$S \rightarrow NP VP NP \mid NP NP VP \mid VP NP NP$

- Given that there may be different possibilities changes within each NP and VP, huge set of rules will be generated for these languages.



# Shortcomings: Generative and Logical Grammars

- Logical grammars suffer from similar problems for languages with different or flexible word order. For instance for Farsi, we have to change the type of the transitive verb from

$(NP \backslash S) / NP$

to

$NP / NP / S$

- to parse “John loves Pizza” in Farsi, which has order:

John	Pizza	loves.
NP	<u>NP NP / NP / S</u>	
<u>NP</u>	NP / S	
	S	

# Shortcomings: Generative and Logical Grammars

- And for Czech, we have to add many verb types to our lexicon, e.g.

(NP\S)/NP,  
NP/NP/S,  
S/NP/NP

- In this approaches, although the number of rules stays the same (the same 2 cancelation schemata for any language), the lexicon becomes huge.

# Dependency Grammars

We can overcome some of these problems with **dependency grammars**.

Dependency grammars are based on the notion of a **dependency relation**.

# Dependency Grammars

- Dependency grammars are based on the notion of **dependency relation**.
- These are an extension of the traditional notion of **grammatical relation**, examples of which are:

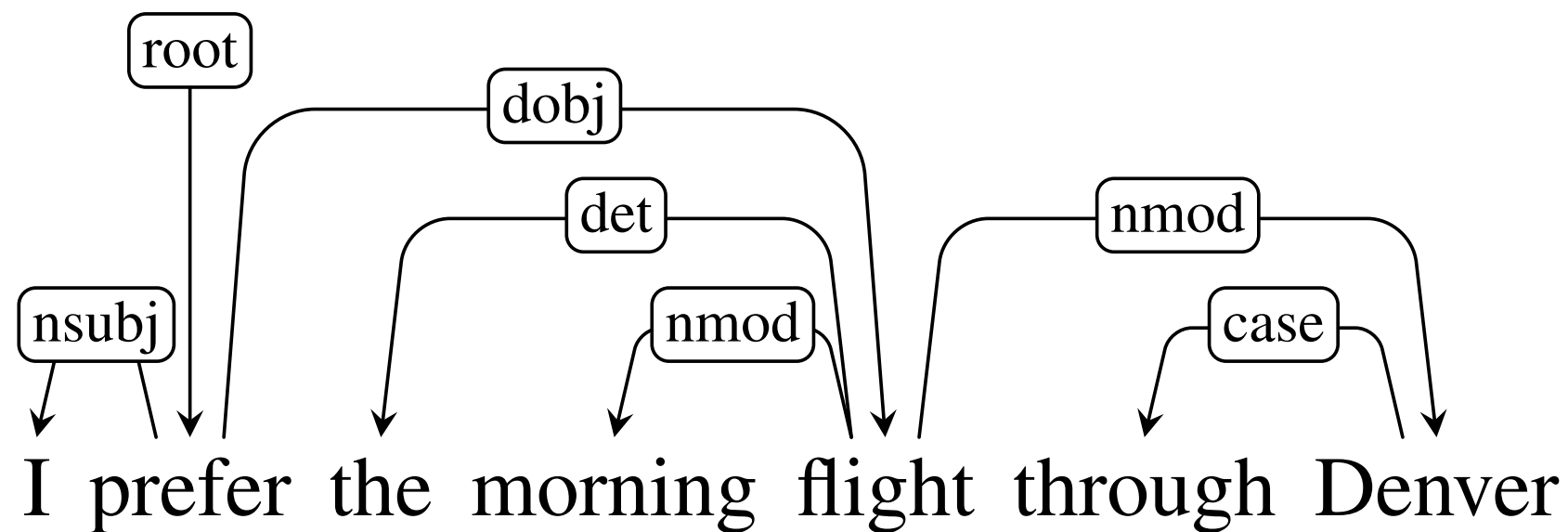
**nsubj, dobj, det, mod...**

- As such, they are more intuitive. Also, all languages have such relations in them, although may be their order in a sentence differs. But that is not a problem, since dependency grammars do not assume a fixed word order.
- A dependency grammar is expressed in terms of a **graph**, sometimes refined to a **tree**.

# Dependency Grammars

A dependency grammar is expressed in terms of a **graph**, sometimes refined to a **tree**.

Here is an example of a dependency graph (which is a tree):

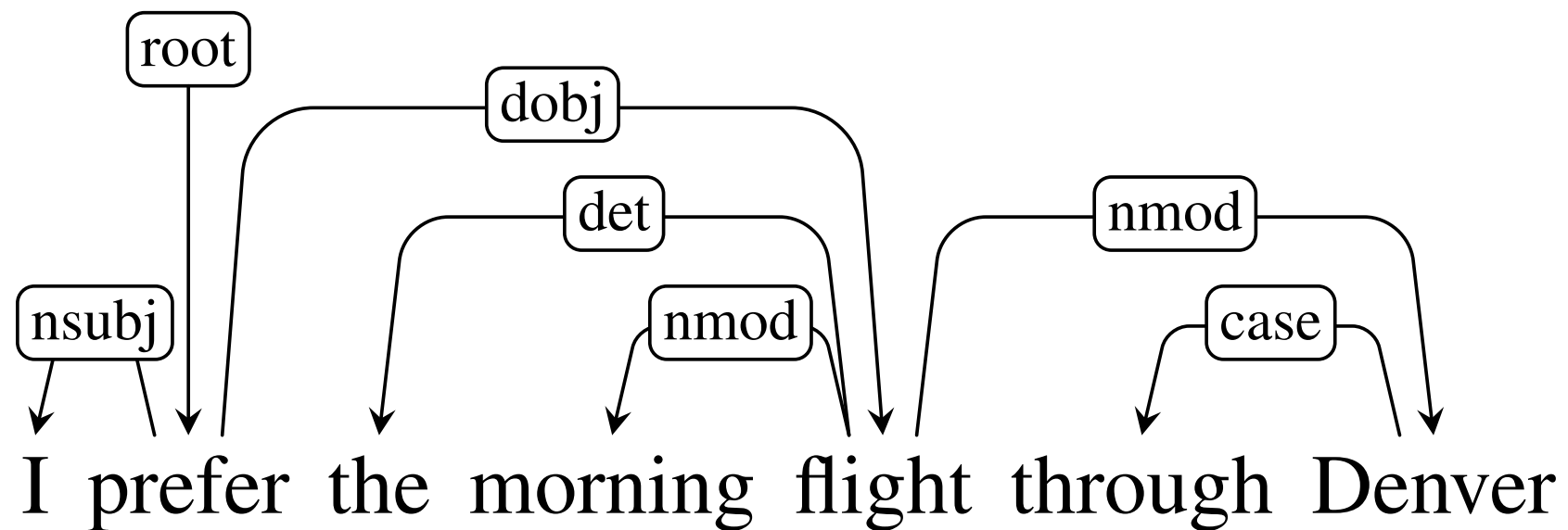


# Some Formal Definitions

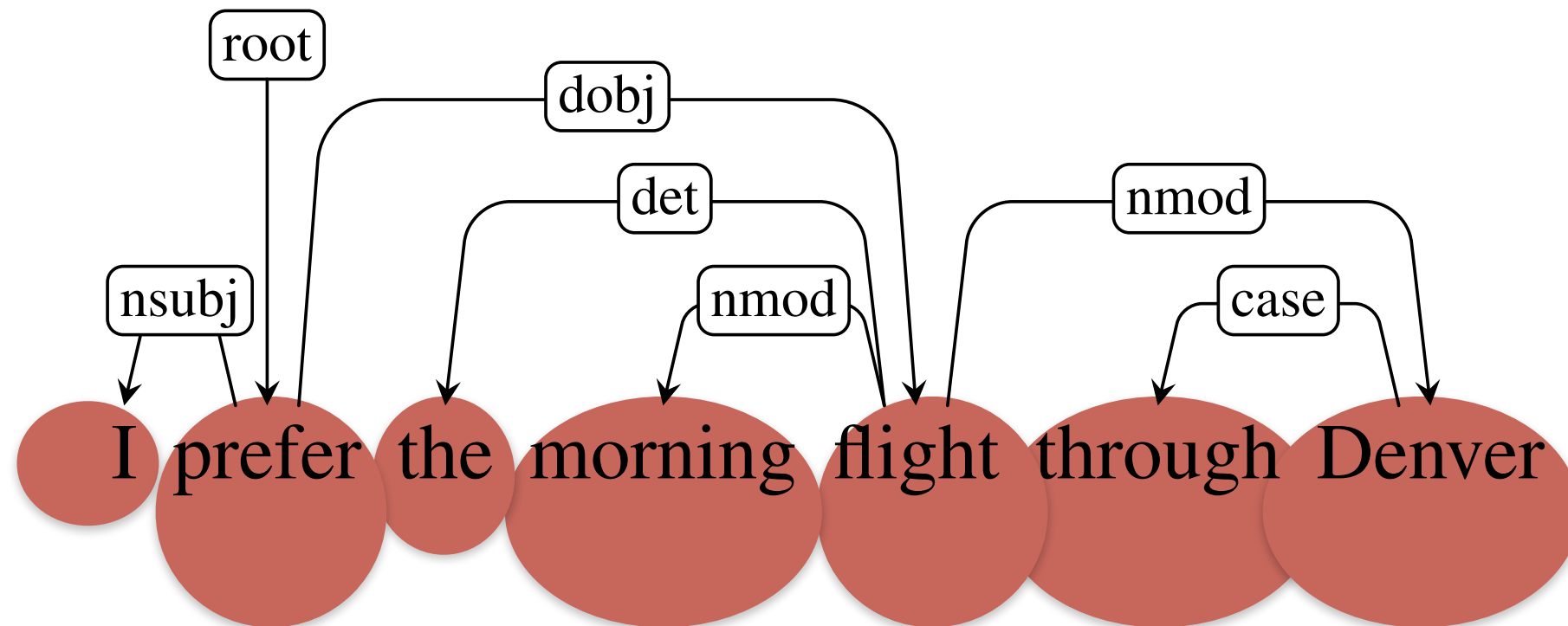
The most general definition of dependency grammars is that of a **dependency structure**:  $G = (V, A)$ . This is a graph  $G$  with:

- 1- a set of **vertices** (nodes)  $V$ , which are labelled by the words of sentences we are analysing.
- 2- a set of **arcs** (edges)  $A$ , labelled by the dependency relation between the two words it is connecting.

# Some Formal Definitions



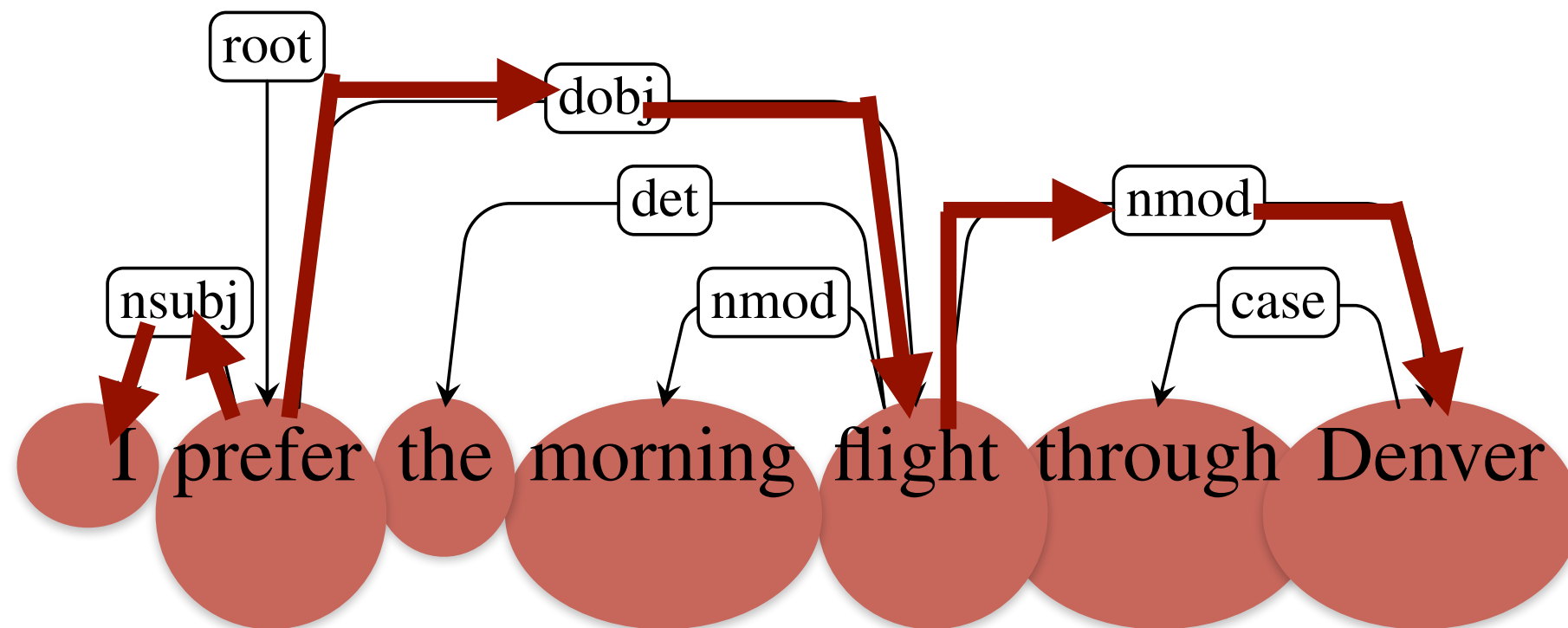
# Some Formal Definitions



The **vertices** of the graph.



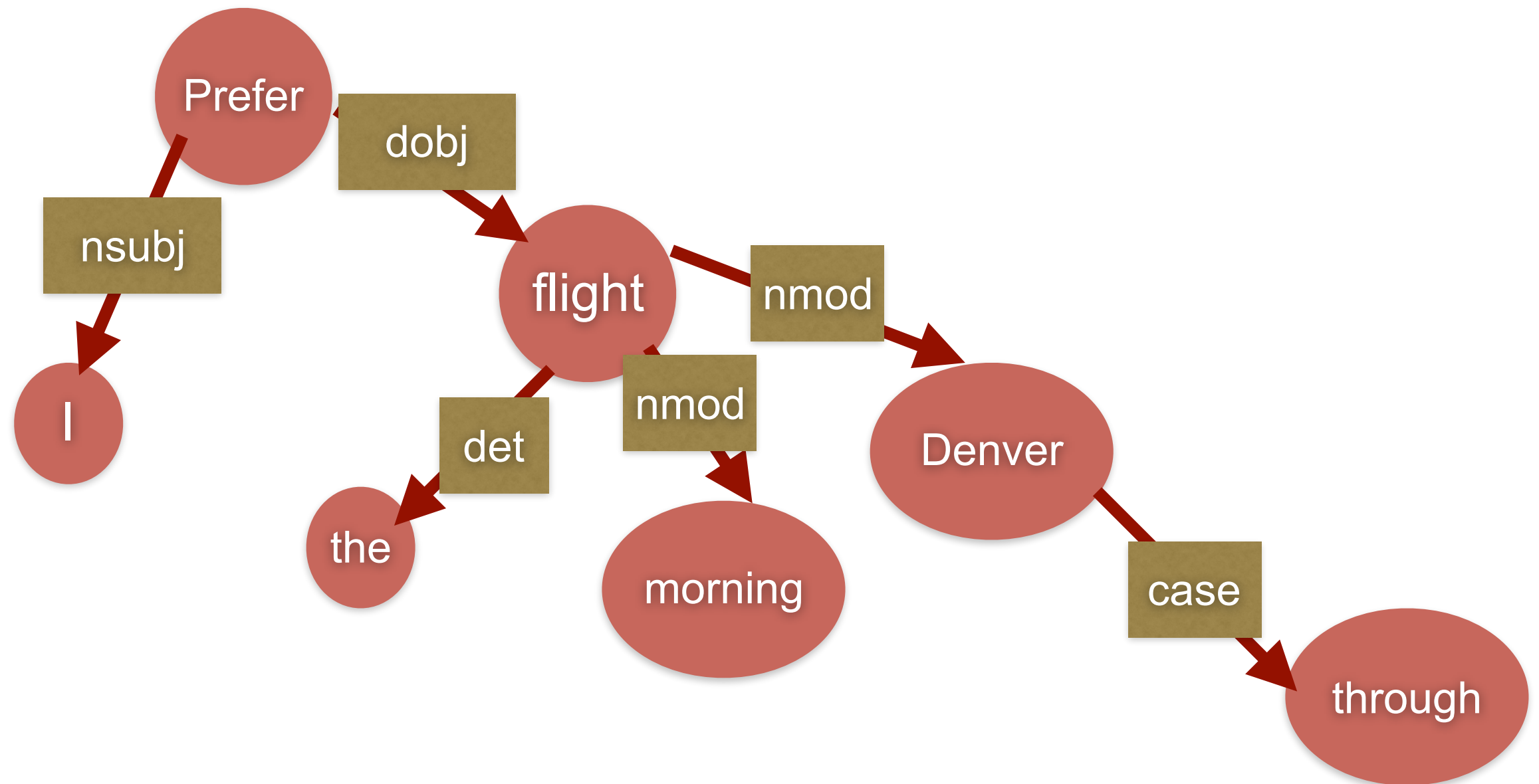
# Some Formal Definitions



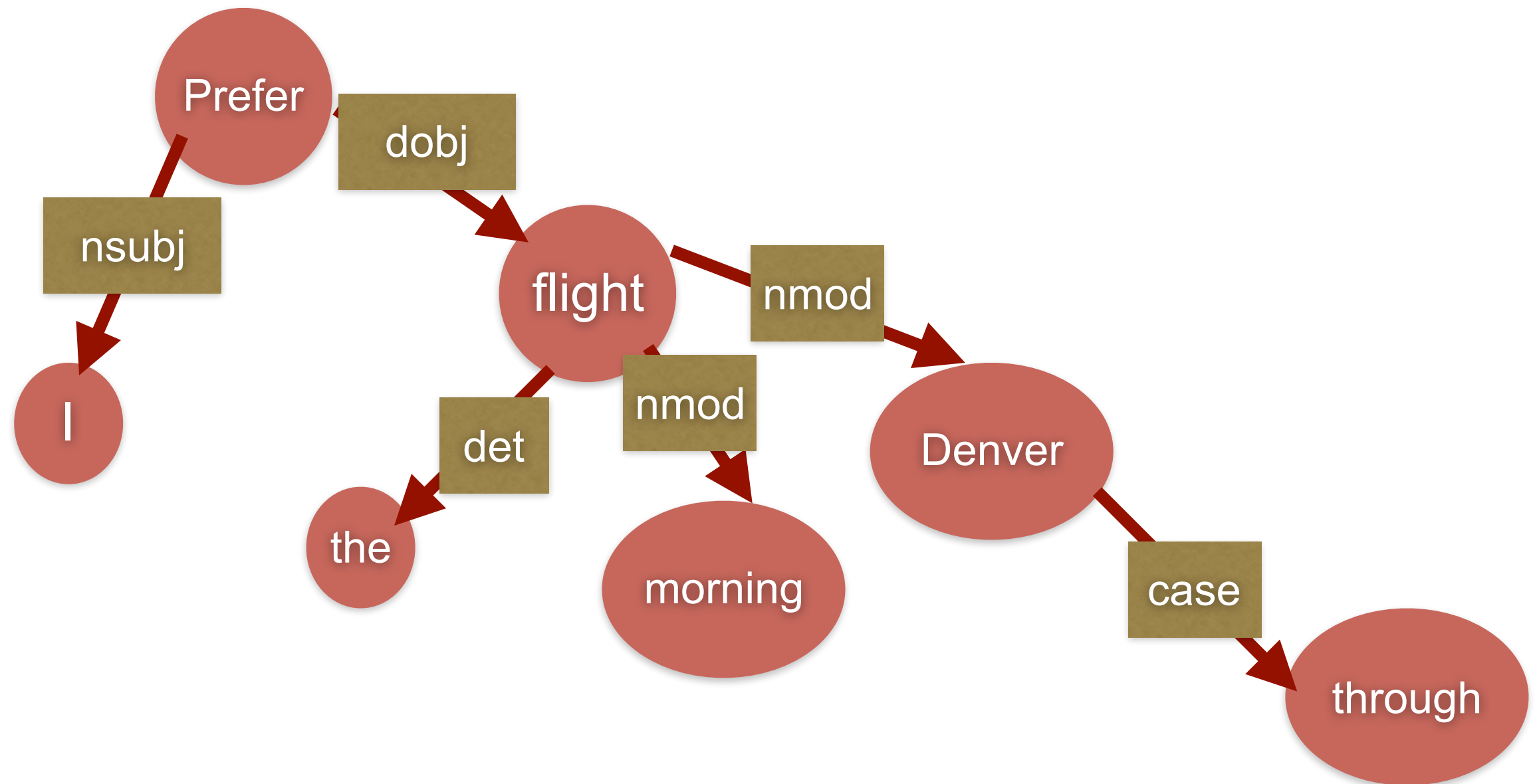
The **arcs** of the graph.

# Some Formal Definitions

We can redraw this to make it look more graph-like:



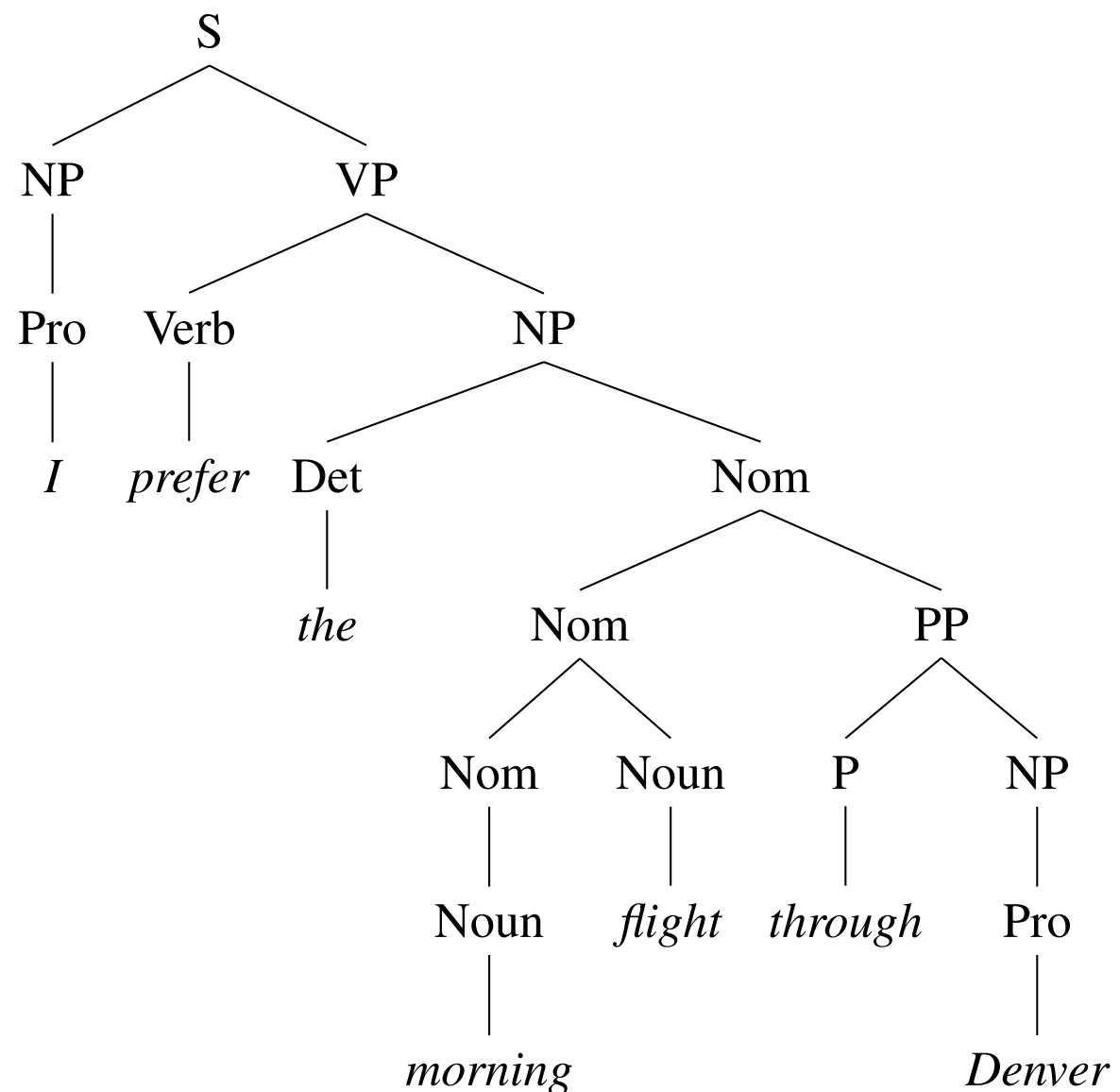
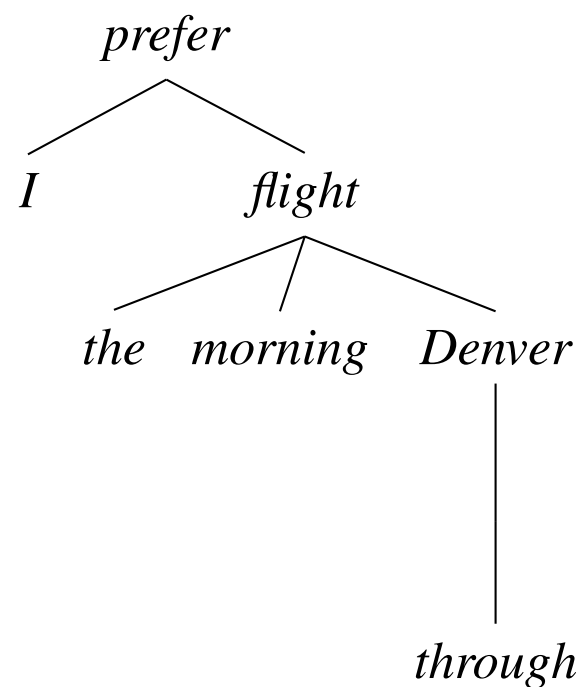
# Some Formal Definitions



As a matter of fact ... this is a tree!

# Comparison to Generative Trees

Dependency trees are simpler than the generative ones.



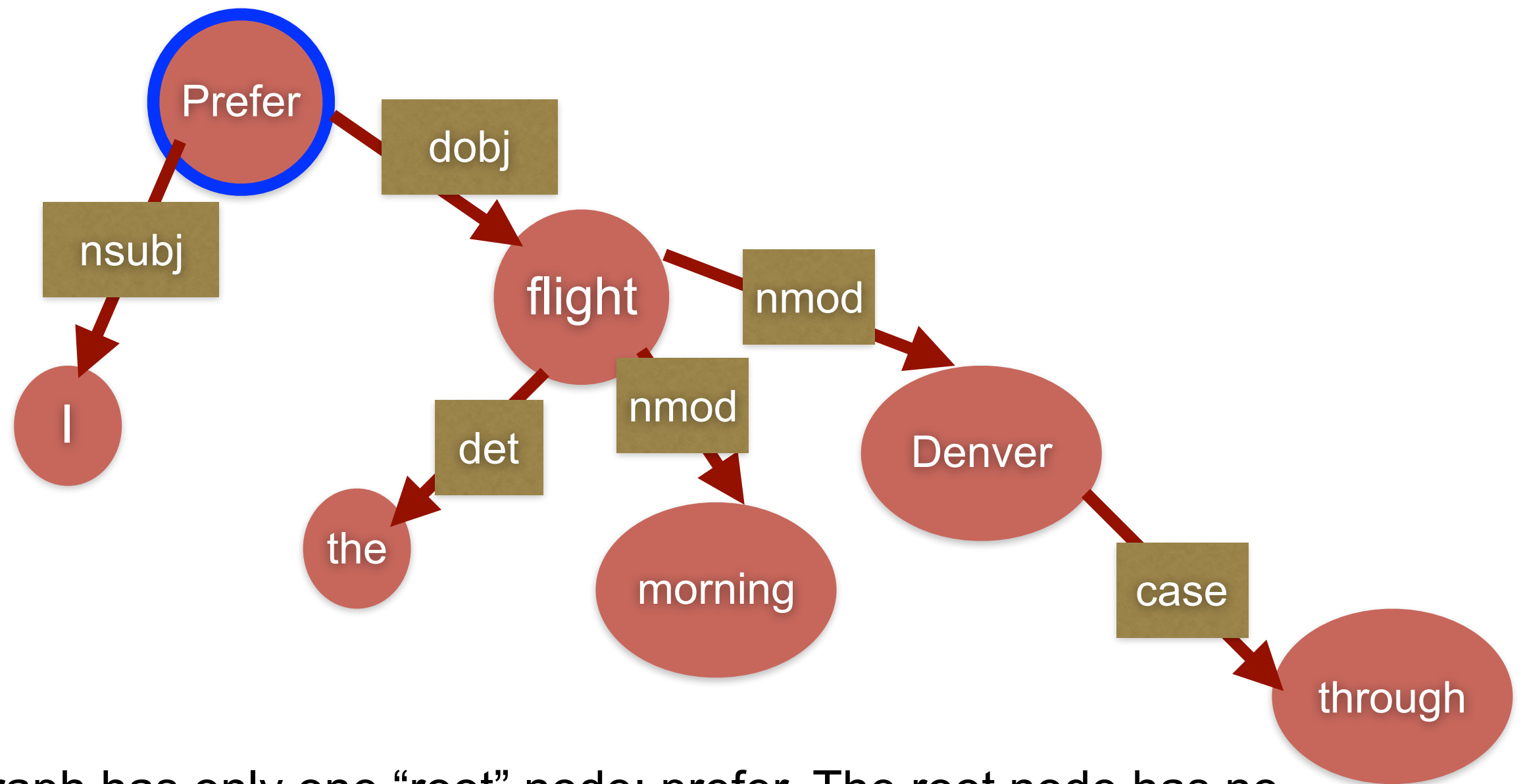
# Some Formal Definitions

Each language might pose different restrictions on its dependency structures. A most common restriction used by many parsers is assuming that these graphs are trees.

A **dependency tree** is defined as follows (Jurafsky & Martin)

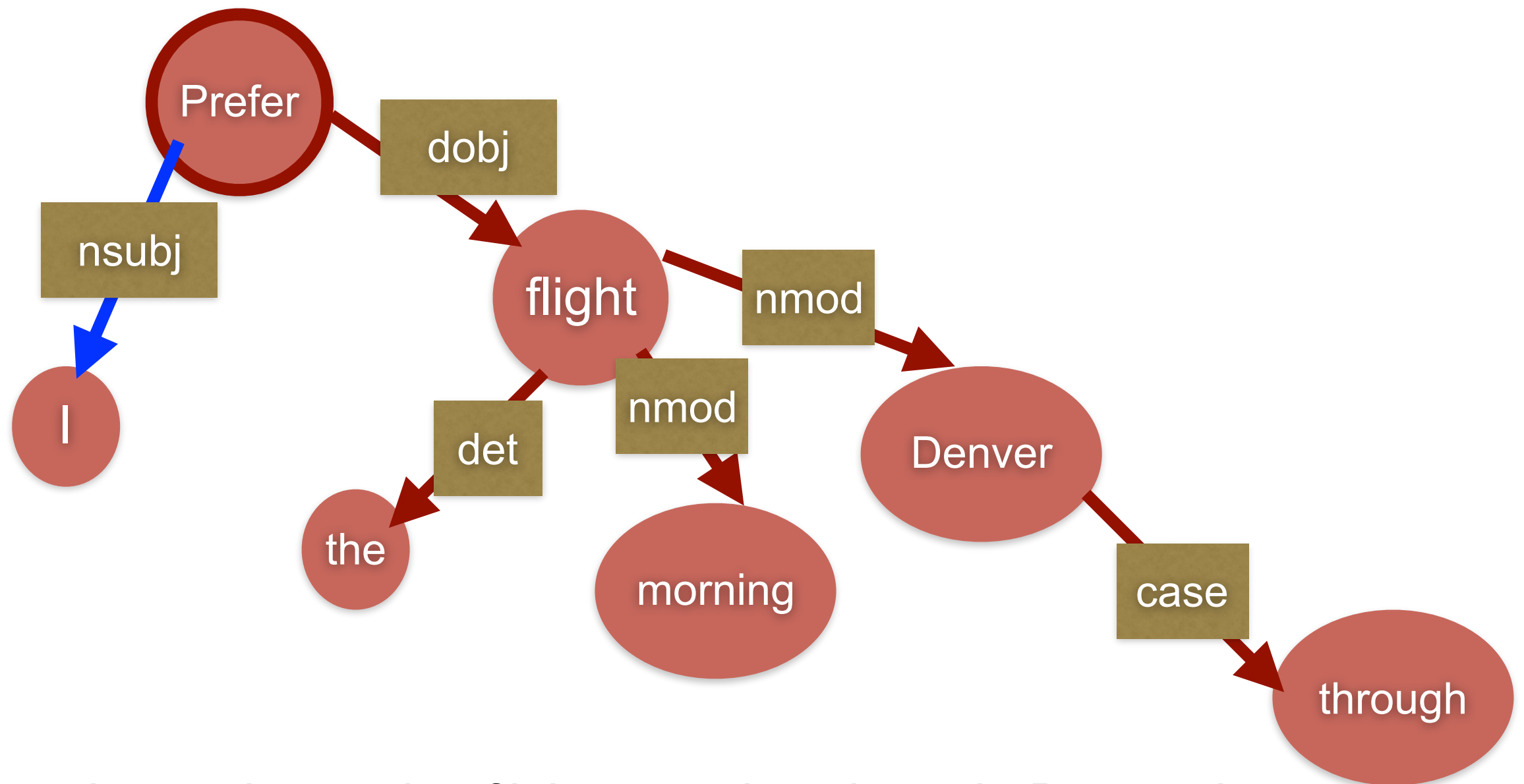
1. There is a single designated root node that has no incoming arcs.
2. With the exception of the root node, each vertex has exactly one incoming arc.
3. There is a unique path from the root node to each vertex in  $V$ .

# Some Formal Definitions



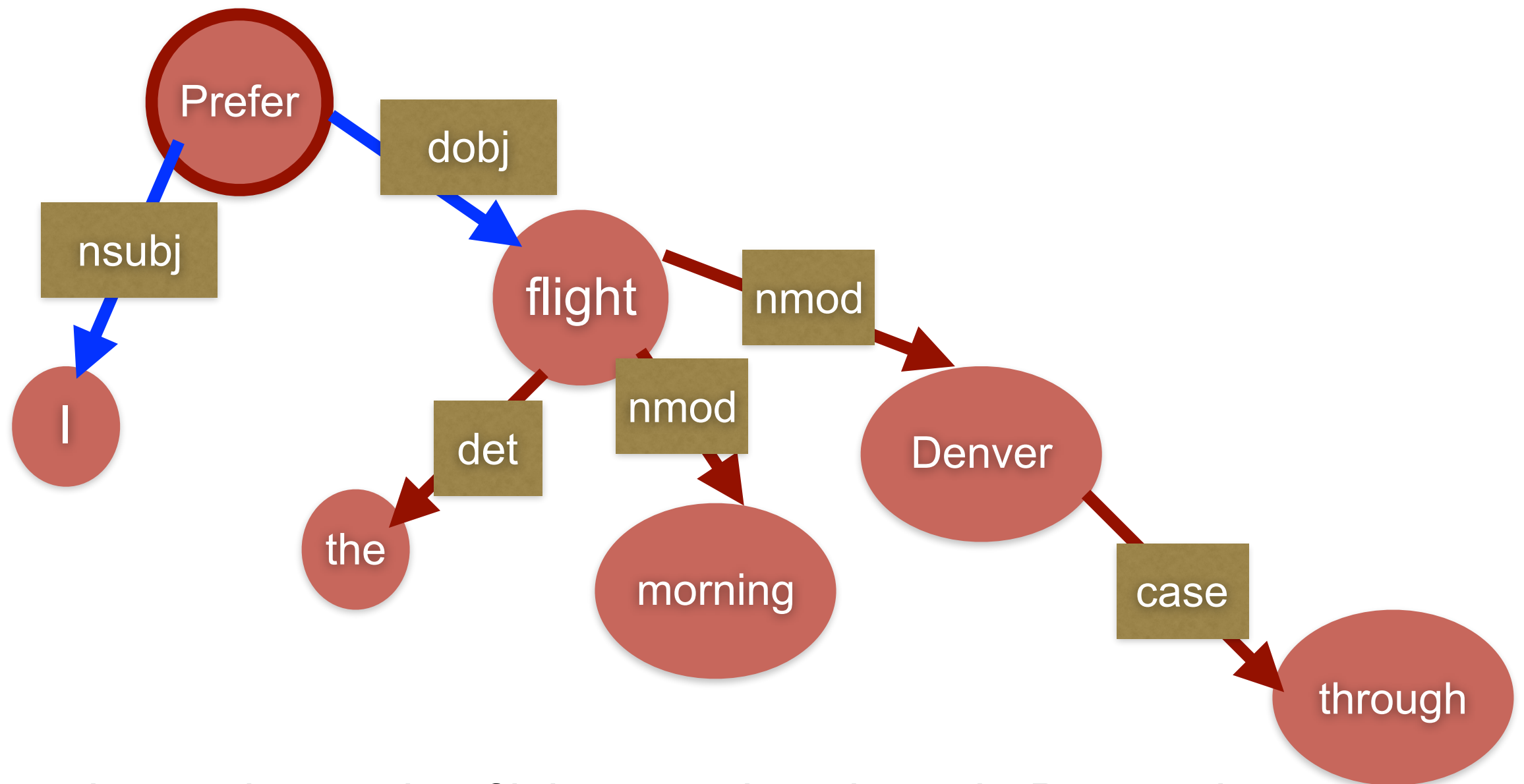
This graph has only one “root” node: prefer. The root node has no incoming edge to it. It only has outgoing edges (2 of them).

# Some Formal Definitions



All the other nodes, I, the, flight, morning, through, Denver have only 1 incoming arc to them. Here is the arc to I.

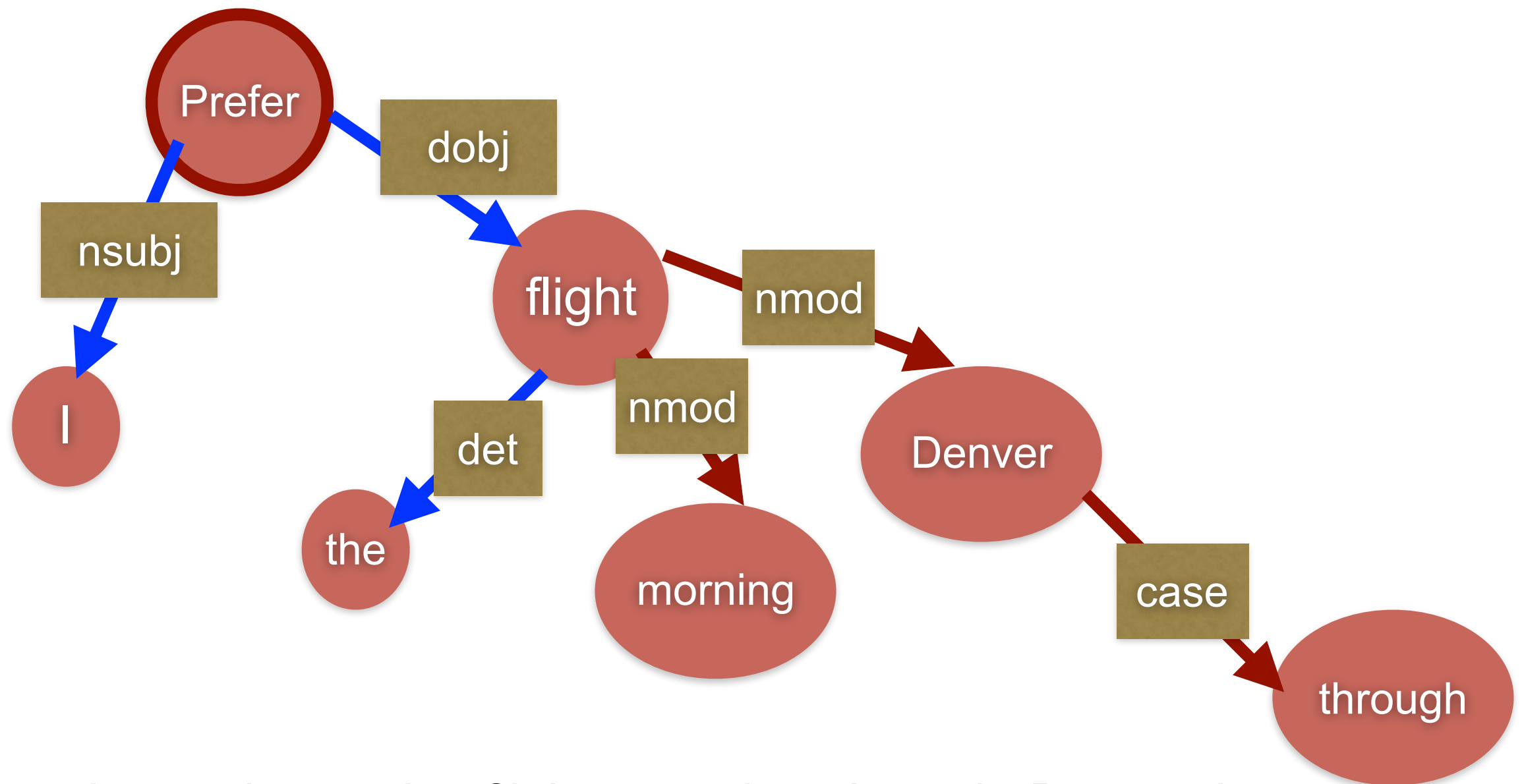
# Some Formal Definitions



All the other nodes, I, the, flight, morning, through, Denver have only 1 incoming arc to them. Here is the arc to flight.

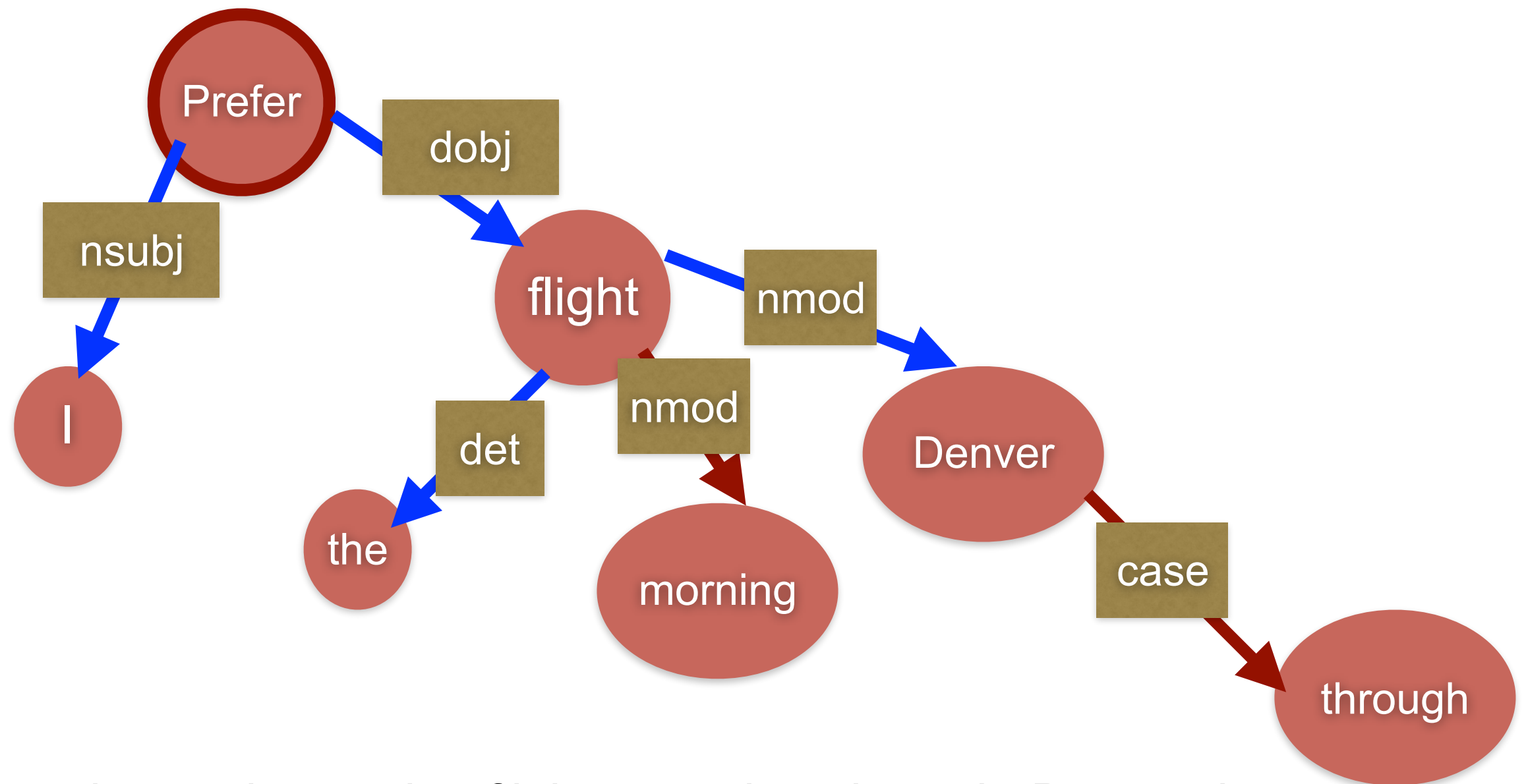


# Some Formal Definitions



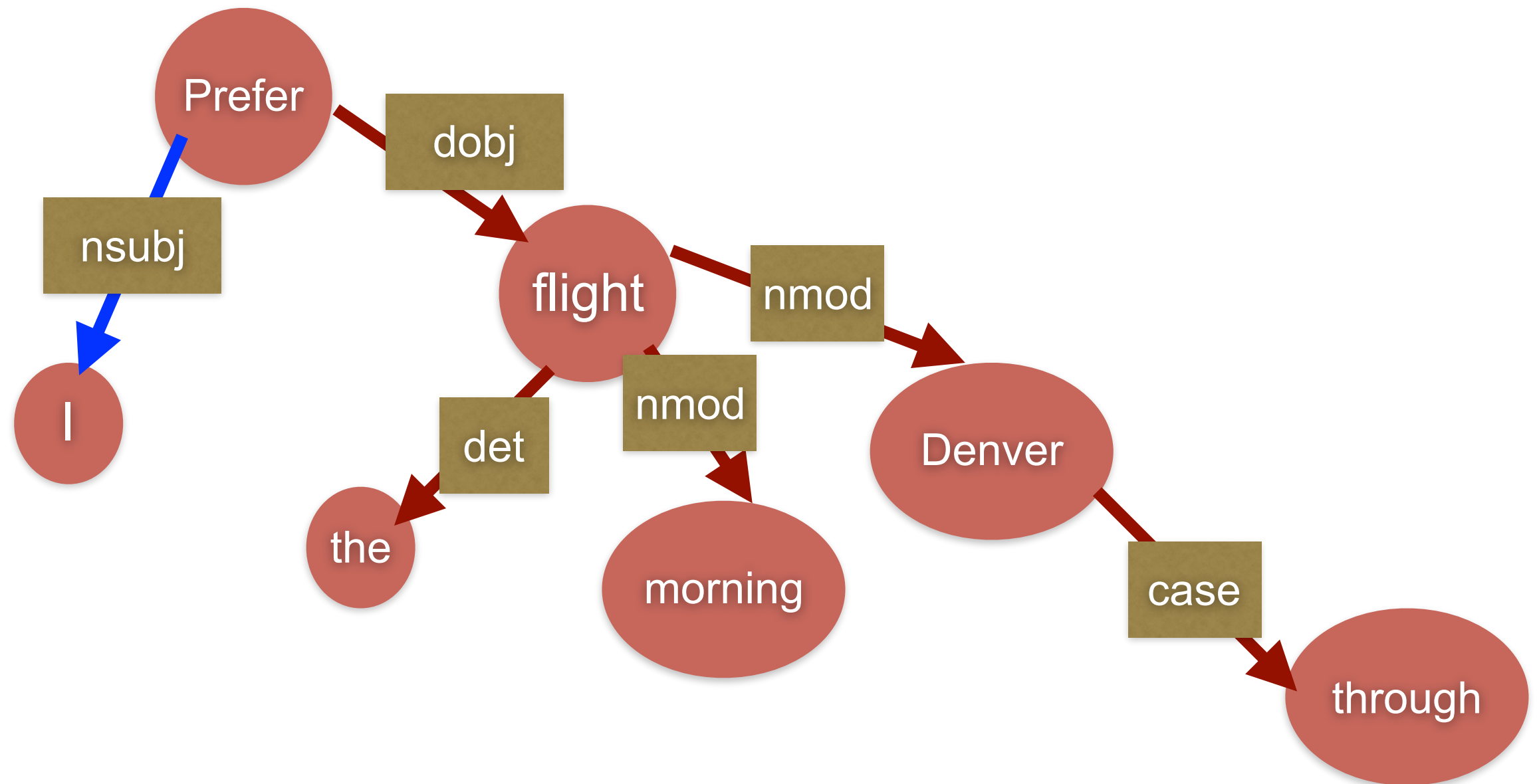
All the other nodes, I, the, flight, morning, through, Denver have only 1 incoming arc to them. Here is the arc to the.

# Some Formal Definitions



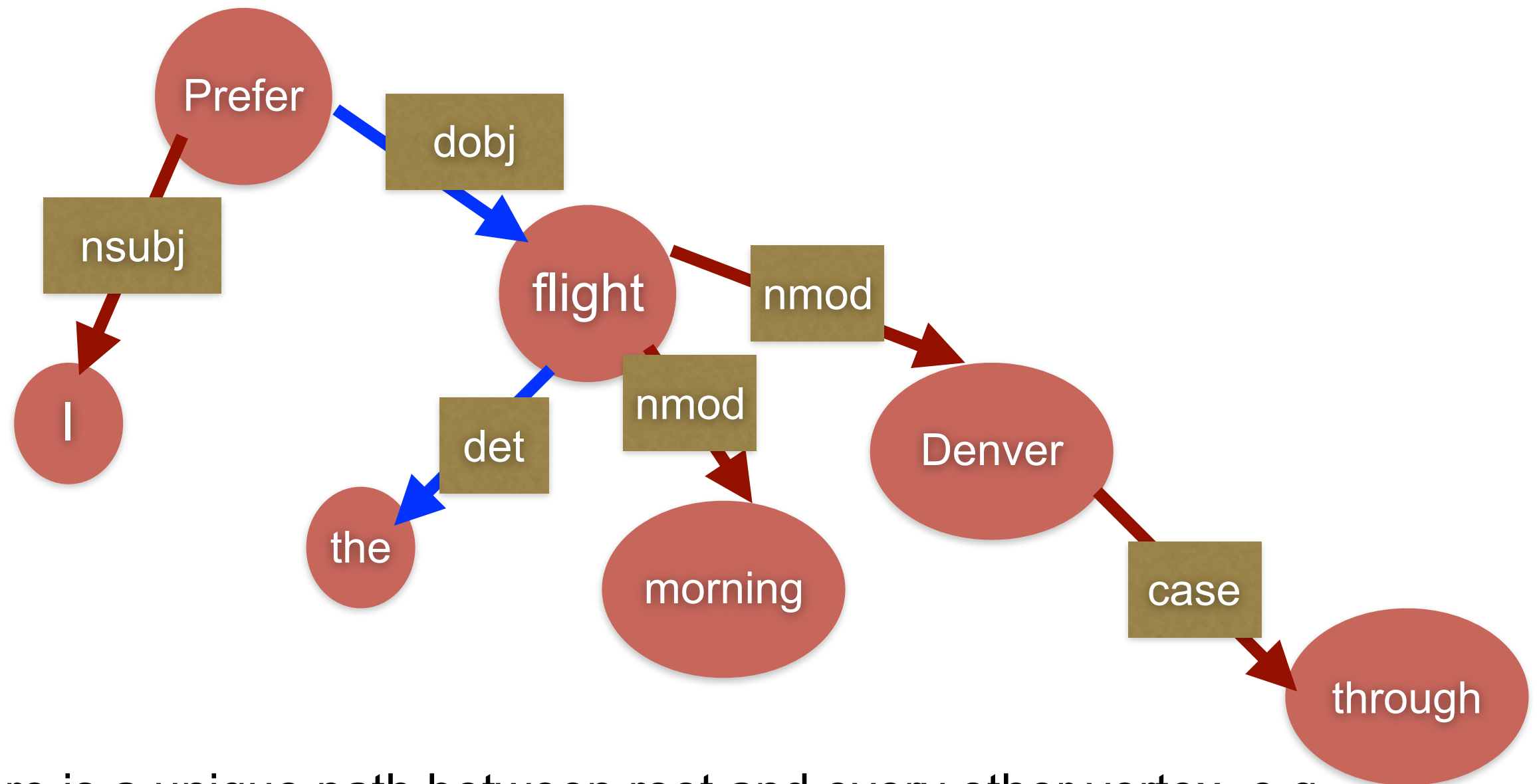
All the other nodes, I, the, flight, morning, through, Denver have only 1 incoming arc to them. Here is the arc to Denver.

# Some Formal Definitions



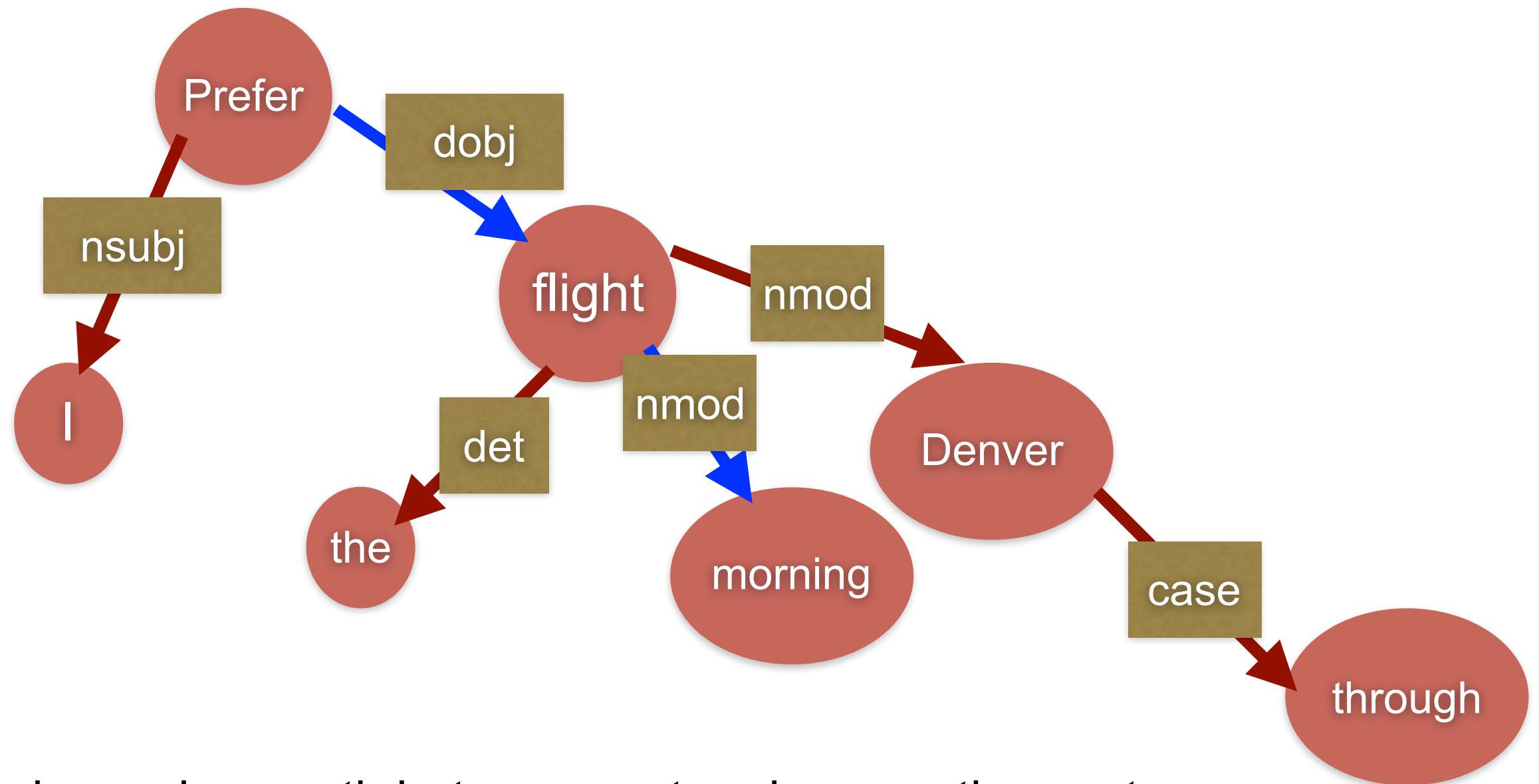
There is a unique path between root and every other vertex, e.g. prefer and I.

# Some Formal Definitions



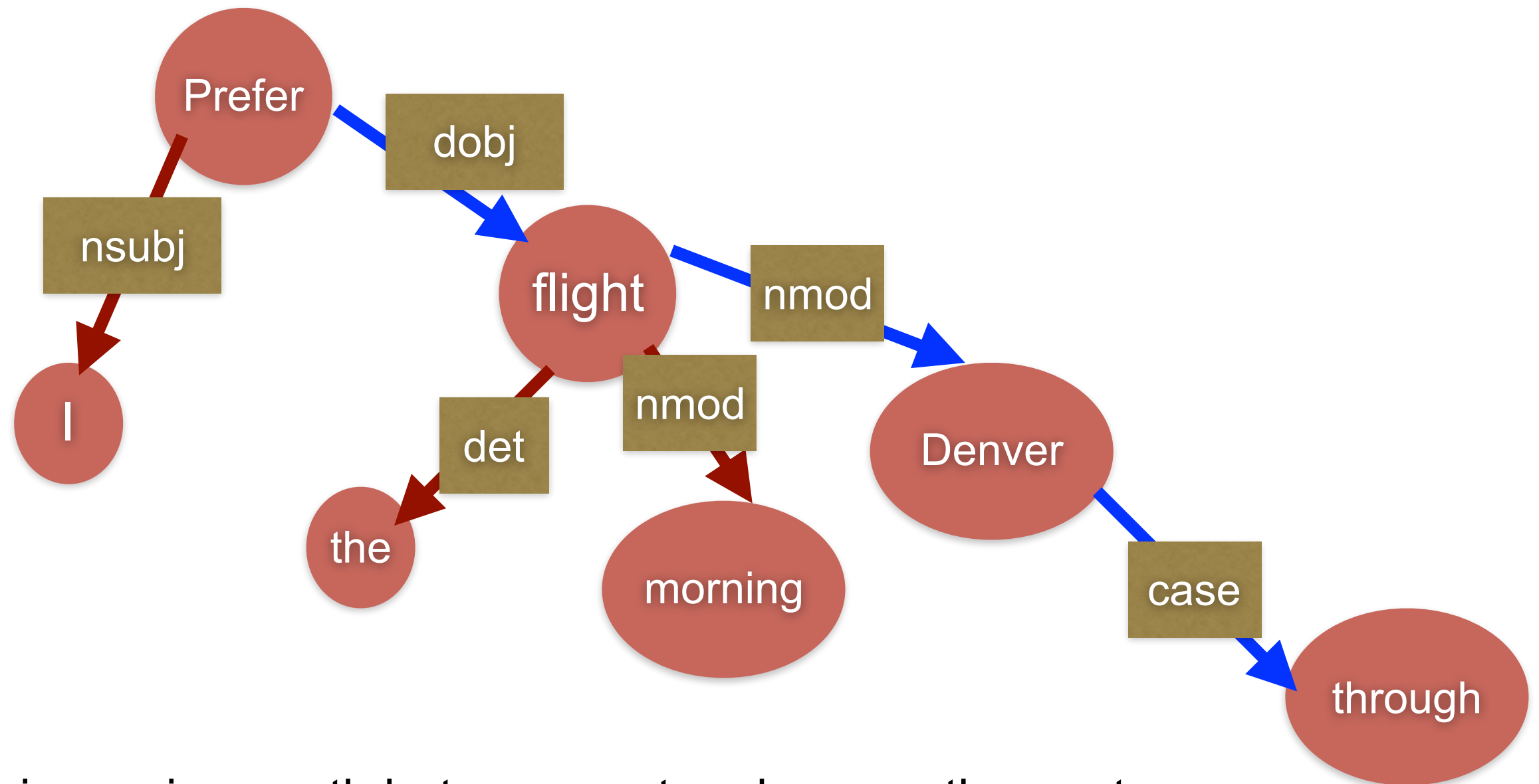
There is a unique path between root and every other vertex, e.g. prefer and the.

# Some Formal Definitions



There is a unique path between root and every other vertex, e.g. prefer and morning.

# Some Formal Definitions



There is a unique path between root and every other vertex, e.g. prefer and through.

# Universal Dependency Relations

The labels of the arcs are taken from tables that can be used for all languages.

<b>Clausal Argument Relations</b>	<b>Description</b>
NSUBJ	Nominal subject
DOBJ	Direct object
IOBJ	Indirect object
CCOMP	Clausal complement
XCOMP	Open clausal complement
<b>Nominal Modifier Relations</b>	<b>Description</b>
NMOD	Nominal modifier
AMOD	Adjectival modifier
NUMMOD	Numeric modifier
APPOS	Appositional modifier
DET	Determiner
CASE	Prepositions, postpositions and other case markers
<b>Other Notable Relations</b>	<b>Description</b>
CONJ	Conjunct
CC	Coordinating conjunction

# Let's practice

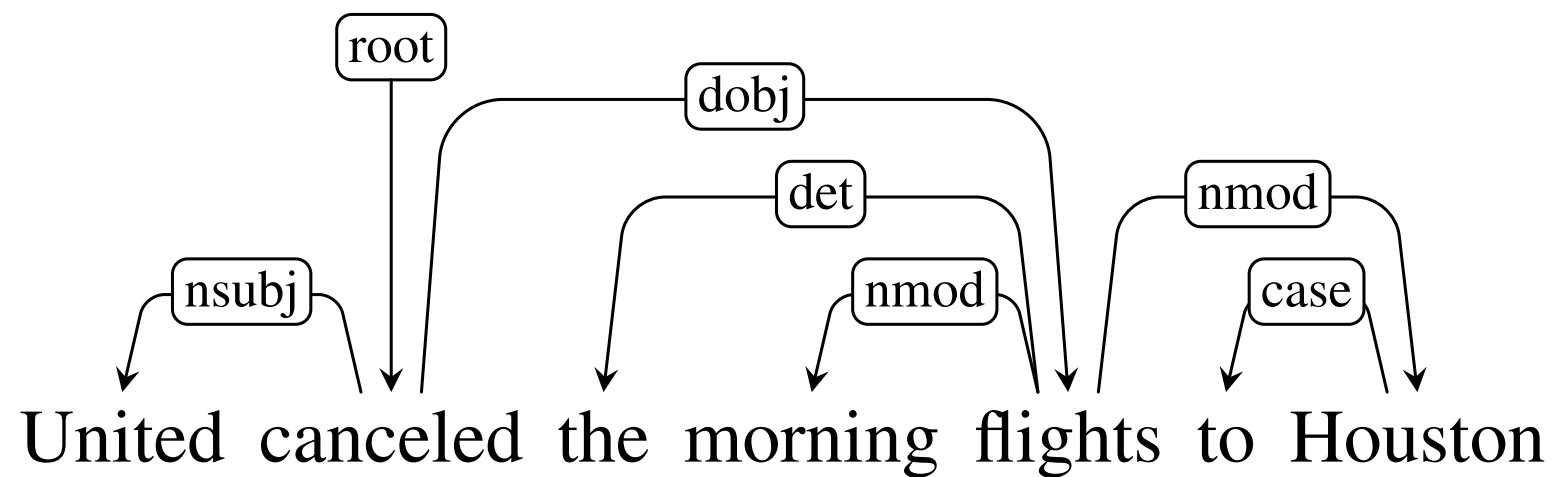
Draw a labelled dependency tree for the following sentence:

United canceled the morning flights to Houston



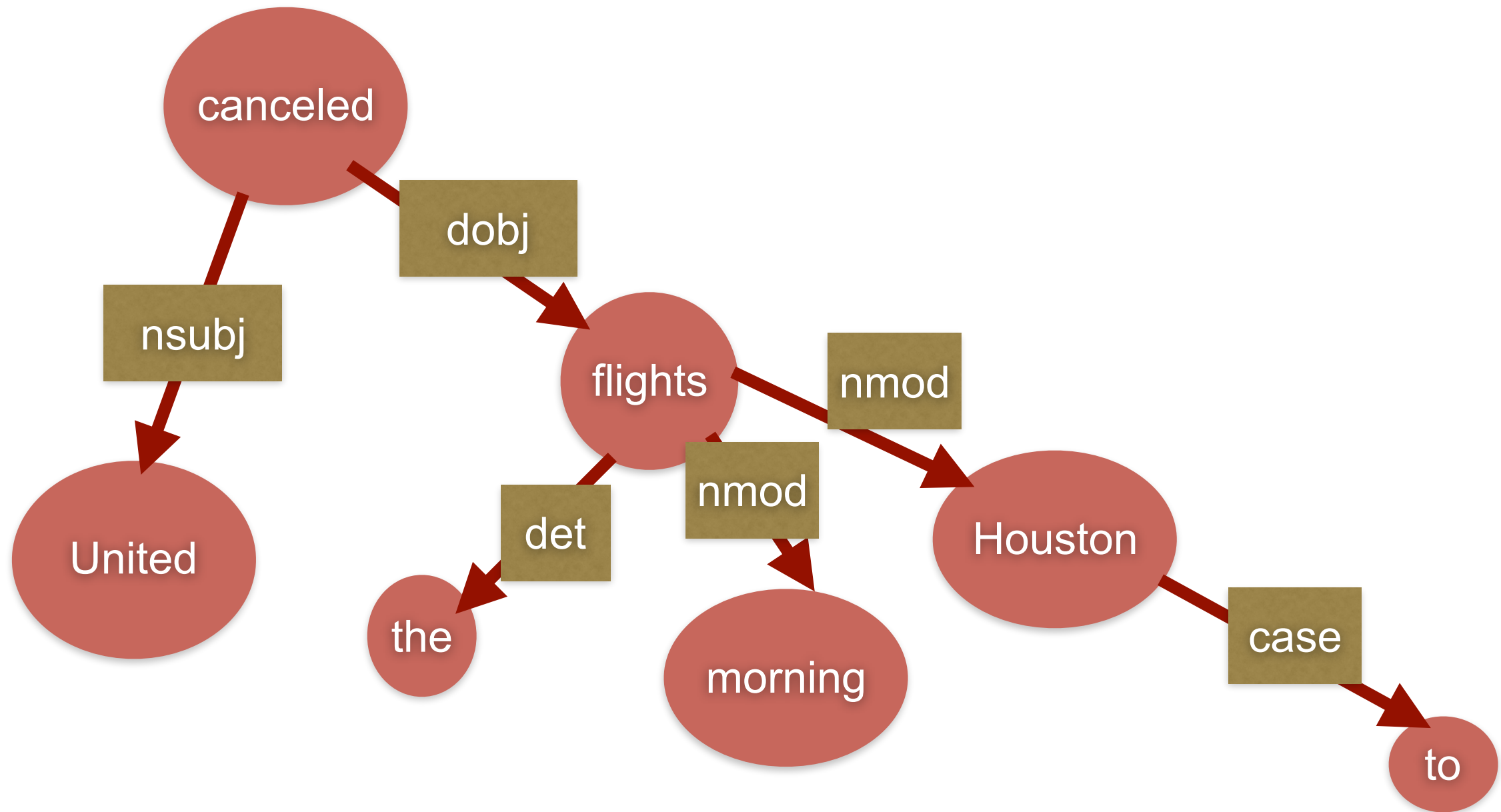
# Let's practice

Draw a labelled dependency tree for the following sentence:



# Let's practice

In the proper form:

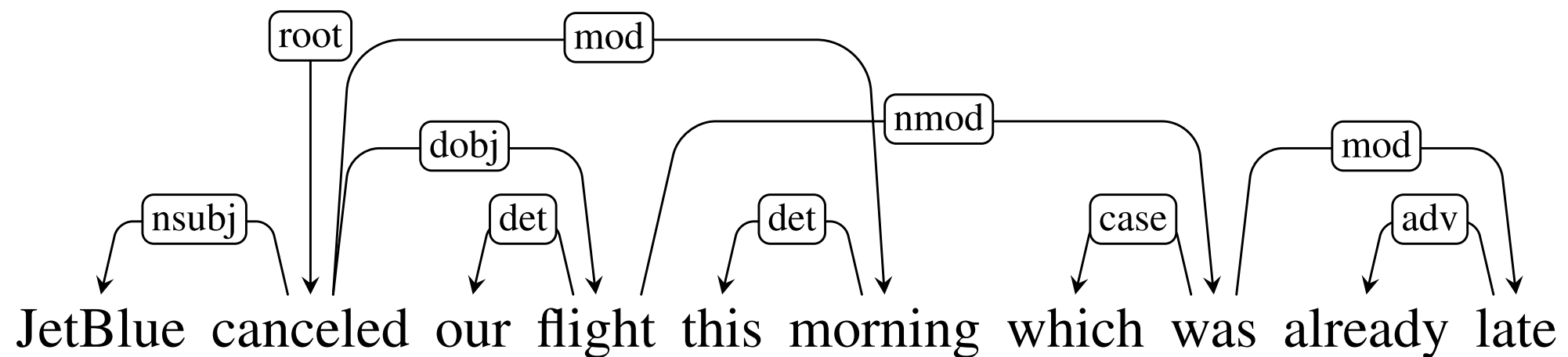


# Projectivity

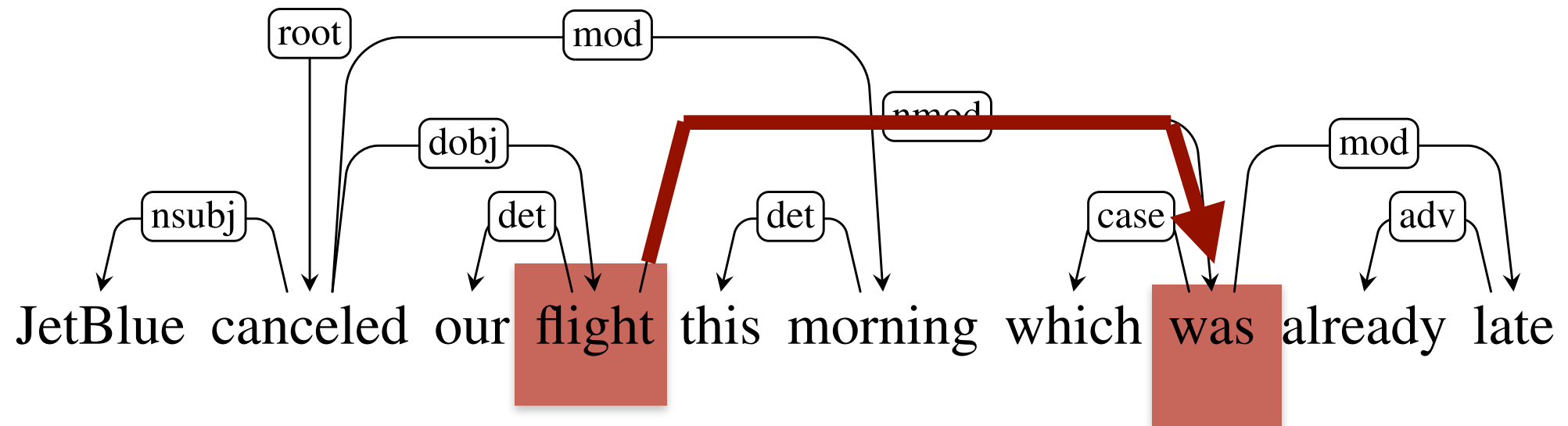
- An arc between  $v$  and  $w$  in a dependency tree is called **projective** if there is a path between  $v$  and every other word that occurs in the sentence between  $v$  and  $w$ .
- A dependency tree is projective if all its arcs are.

# Projectivity

- All our previous example of dependency trees are projective. Here is an example of a tree that is not.

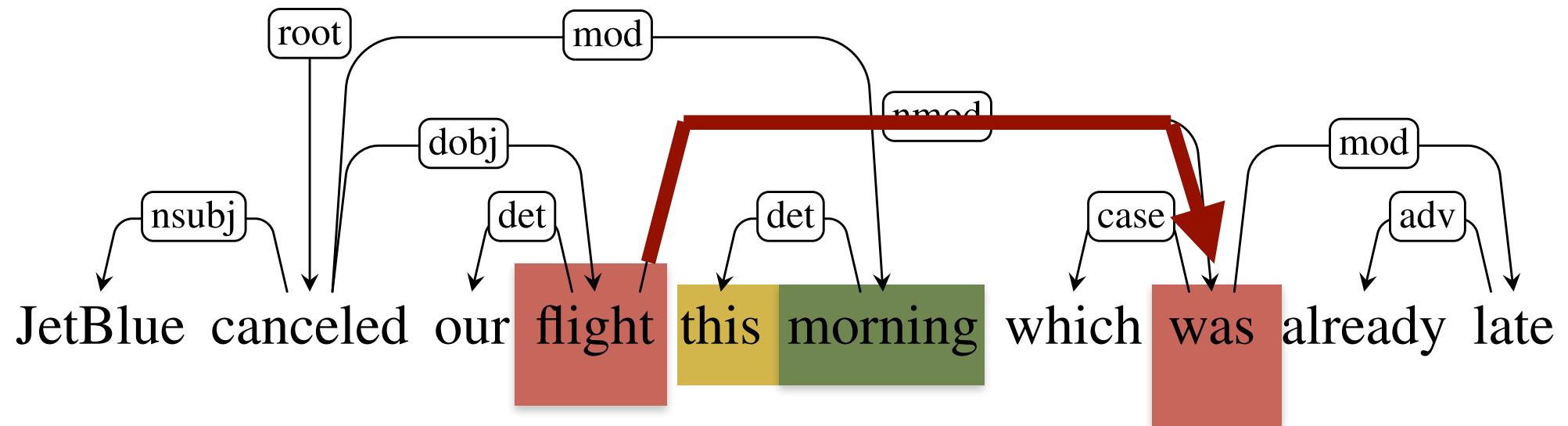


# Projectivity



- Here, there is an arc between “flight” and “was”.
- Words “this” and “morning” are between “flight” and “was”.

# Projectivity



- Here, there is an arc between “flight” and “was”.
- Words “this” and “morning” are between “flight” and “was”.
- But there is no path between “flight” and either of these.

# Projectivity

- Projectivity is shown to be equivalent to the absence of crossings of arcs in dependency trees.
- It is important because: there is an algorithm that translates generative grammar trees to dependency trees. This algorithm only produces projective trees. So one can say that the non-projective trees are either not context free or have grammatical mistakes in them.

# The End of Syntax 2!

There will be a Lab 4 sheet this Monday

Both labs are due on Friday Nov 15th by 11am

Please upload answers on QMPlus.



# Reading

- Jurafsky and Martin (3<sup>rd</sup> Ed) Chapter 13 & start of Chapter 15