Natural Language Processing

Class 05: Grammars and Dependency parsing

- Grammar Formalisms
- Constituency
- Heads and Dependency
- Dependency Grammars
- Transition-based Dependency Parsing

23 March 2018

Katrien Beuls

VUB VRIJE UNIVERSITEIT BRUSSEL

Syntax (or grammar)

Refers to the way words can be arranged in a given language

Grammars (and parsing) are key components in many applications

- Grammar checkers
- Dialogue management
- Question answering
- Information extraction
- Machine translation

Syntax

Key notions that we'll cover

- Constituency
- Heads

Key formalism

- Context-free grammars

Constituency

Basic idea: groups of words behave as a single unit or phrase

- NOUN PHRASE
  - single words: she, Michael
  - phrases: the house, New York, my well-structured course notes

External evidence for the NP as a constituent: examples can all precede verbs

Constituency

Example: it makes sense to say that these are all noun phrases in English

Harry the Horse	a high-class spot such as Mindy's
the Broadway coppers	the reason he comes into the Hot Box
they	three parties from Brooklyn

Why? One piece of evidence is that they can all precede verbs

- This is external evidence

Grammars and constituency

There is nothing easy or obvious about how we come up with

- right set of constituents and
- the rules that govern how they combine

There are many different theories of grammar and competing analyses of the same data

Context-free grammars

Context-free grammars (CFGs) allow to model constituency

Consist of

- Terminals: Words (for now)
- Non-terminals
- Constituents in a language: E.g. noun phrase, verb phrase, sentence
- Rules: equations that consist of a single non-terminal on the left and any number of terminals and non-terminals on the right

Some NP rules

$$
\begin{aligned}
NP &\rightarrow Det\ Nominal \\
NP &\rightarrow ProperNoun \\
Nominal &\rightarrow Noun \mid Nominal\ Noun
\end{aligned}
$$

These rules describe two kinds of NPs:

- One that consists of a determiner followed by a nominal
- Another that says that proper names are NPs
- Third rule illustrates
  - A disjunction
  - two kinds of nominals
  - A recursive definition:
    Same non-terminal on the right and left-side of the rule

Example Grammar

Grammar Rules			Examples
S	→	*NP VP*	I + want a morning flight
NP	→	*Pronoun*	I
	|	*Proper-Noun*	Los Angeles
	|	*Det Nominal*	a + flight
Nominal	→	*Nominal Noun*	morning + flight
	|	*Noun*	flights
VP	→	*Verb*	do
	|	*Verb NP*	want + a flight
	|	*Verb NP PP*	leave + Boston + in the morning
	|	*Verb PP*	leaving + on Thursday
PP	→	*Preposition NP*	from + Los Angeles

Generativity

As with FSAs and FSTs, you can view these rules as either analysis or synthesis machines

- Generate strings in the language
- Reject strings not in the language (recognizer)
- Impose structures (trees) on strings in the language (parser)

What is a grammar formalism for?

Distinguish strings that are in the language from those that are not in the language.

- The girl sees a boy.
- *Girl the the.
  - No derivation exists using the grammar rules.

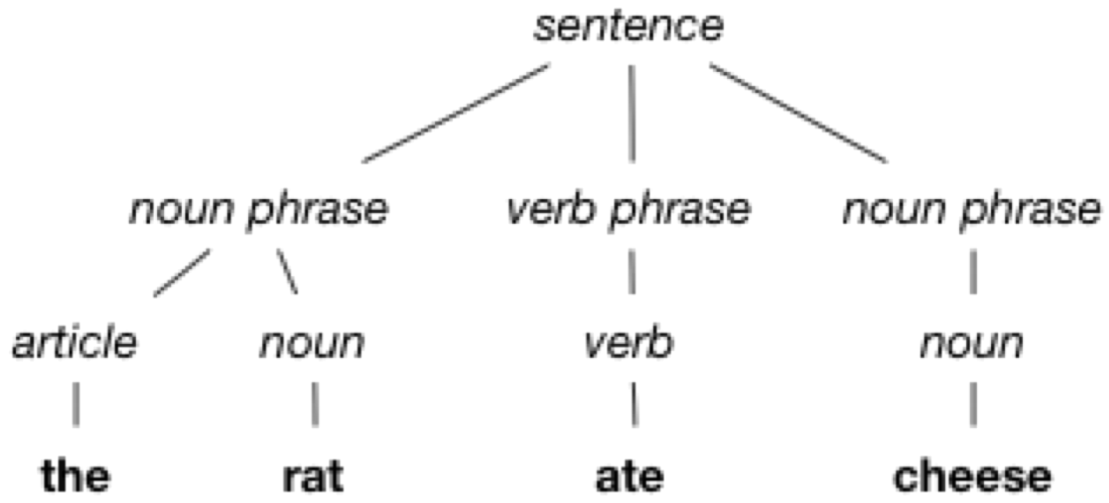
Identify a structure for the sentence.

Derivations

A derivation is a sequence of rules applied to a string that accounts for that string

- Covers all the elements in the string
- Covers only the elements in the string

Represented as a parse tree



The rat ate cheese

S

NP VP

NP V NP

Det N V NP

Det N V N

The N V N

The rat V N

The rat ate N

The rat ate cheese

S → NP VP
NP → Det N
NP → N
VP → V NP
DET → the
N → rat
N→ cheese
V → ate

Definition

More formally, a CFG consists of

N a set of **non-terminal symbols** (or **variables**)

Σ a set of **terminal symbols** (disjoint from N)

R a set of **rules** or productions, each of the form $A \rightarrow \beta$,
where A is a non-terminal,
β is a string of symbols from the infinite set of strings $(\Sigma \cup N)*$

S a designated **start symbol**

Parsing

Parsing is the process of taking a string and a grammar and returning one or more parse trees for that string

Analogous to running a finite-state transducer with a tape

- It’s just more powerful
- Remember this means that there are languages we can capture with CFGs that we can’t capture with finite-state methods

Grammar equivalence

Strong equivalence

- They generate the same set of strings and assign the same phrase structure to each sentence

Weak equivalence

- They generate the same set of strings but do not assign the same phrase structure to each sentence

Chomsky Normal Form

A → B C

A -> a

Either two non-terminals or one terminal on the right-hand side

Binary branching!

Any grammar can be converted into a weakly equivalent CNF grammar

A -> B C D

A -> B X

X -> C D

An English grammar fragment

Sentences

Noun phrases

- Agreement

Verb phrases

- Subcategorization

Sentence types

Declaratives: *A plane left.*
S → NP VP

Imperatives: *Leave!*
S → VP

Yes-No Questions: *Did the plane leave?*
S → AUX NP VP

WH Questions: *When did the plane leave?*
S → WH-NP Aux NP VP

Noun phrases

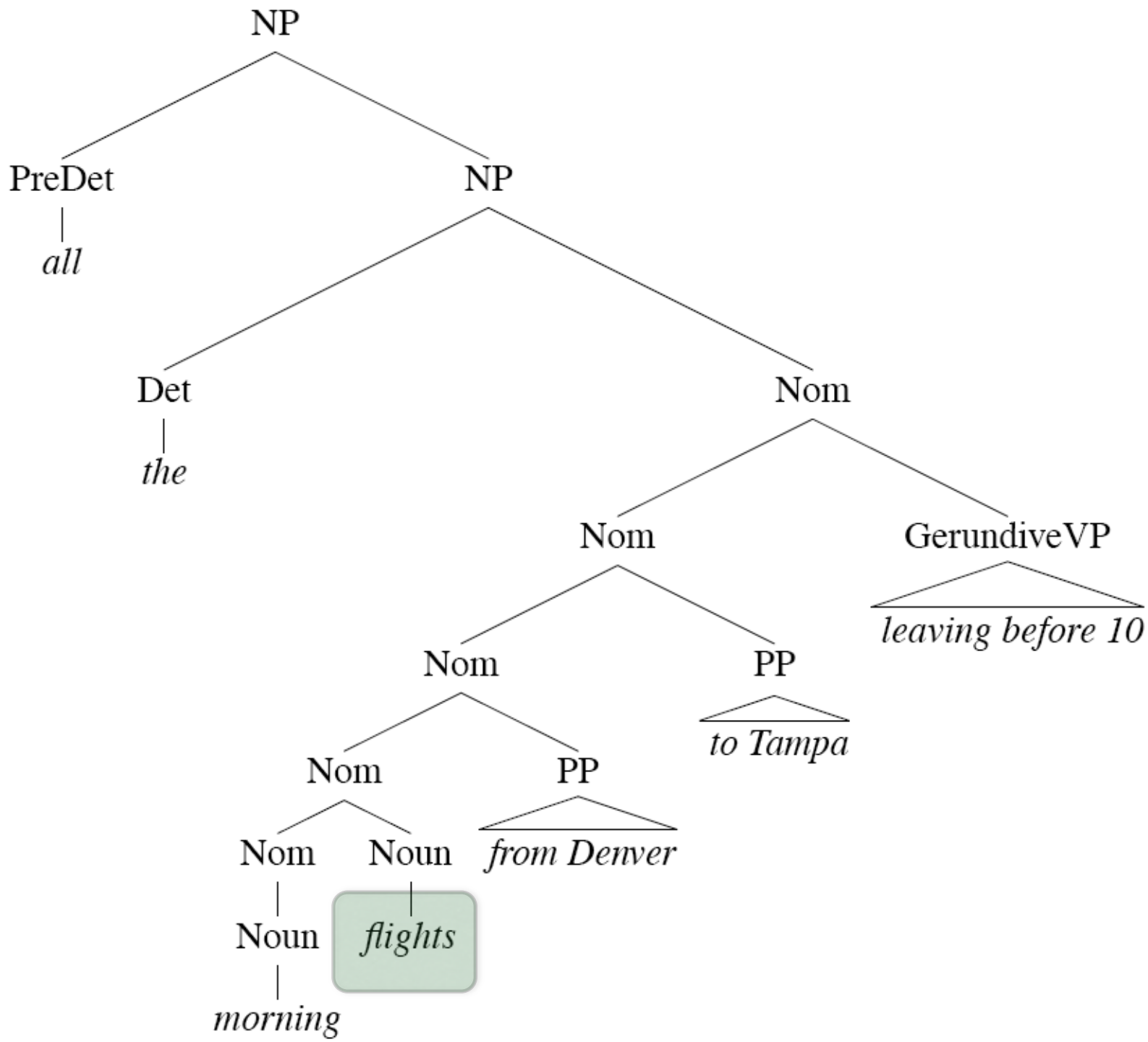
Consider the following rule in more detail…

NP → Det Nominal

Most of the complexity of English noun phrases is hidden in this rule

Consider the derivation for the following example

- *All the morning flights from Denver to Tampa leaving before 10*


NP
PreDet
NP
all
Det
Nom
the
Nom
GerundiveVP
Nom
PP
leaving before 10
Nom
PP
to Tampa
Nom
Noun
from Denver
Noun
flights
morning

NP structure

This NP is really about *flights*. That's the central critical noun in this NP. Let's call that the head.

We can dissect this kind of NP into:

- the constituents that can come before the head, and
- the constituents that can come after it.

Agreement

"Features associated with one linguistic unit become associated with another unit and then possibly overtly expressed"

Example: in English, determiners and head nouns in NPs have to agree in number

This flight
Those flights

**This flights*
**Those flight*

Problem

Our earlier NP rules are clearly deficient because they did not capture this constraint

$$NP \rightarrow Det\ Nominal$$

- Our grammar accepts, and assigns correct structures, to grammatical examples (*this flight*)
- But also accepts incorrect examples (**these flight*)

Such a rule is said to overgenerate

Verb phrases

English VPs consist of a head verb and 0 or more following constituents, called arguments

Examples:

$VP \rightarrow Verb$ disappear

$VP \rightarrow Verb\ NP$ prefer a morning flight

$VP \rightarrow Verb\ NP\ PP$ leave Boston in the morning

$VP \rightarrow Verb\ PP$ leaving on Thursday

Subcategorization

Not all verbs are allowed to participate in all the VP rules

Subcategorize verbs in a language according to the sets of VP rules they participate in

This is a modern take on the traditional notion of transitive/intransitive

Modern grammars may have 100s of subcategorization classes

Subcategorization

Sneeze: *John sneezed*

Find: *Please find* [*a flight to NY*]$_{NP}$

Help: *Can you help* [*me*]$_{NP}$ [*with a flight*]$_{PP}$

Give: *Give* [*me*]$_{NP}$ [*a cheaper fare*]$_{NP}$

Give: *Give* [*a cheaper fare*]$_{NP}$ [*to me*]$_{PP}$

Prefer: *I prefer* [*to leave earlier*]$_{TO\text{-}VP}$

Told: *I was told* [*United has a flight*]$_{S}$

…

Subcategorization

**John sneezed the book*

**I prefer United has a flight*

**Give with a flight*

As with agreement phenomena, we need a way to formally express the constraints

Why?

The VP rules we've seen so far overgenerate

- They permit the presence of strings containing verbs and arguments that don't go together

Example:

$$VP \rightarrow V\ NP$$

therefore

Sneezed the book is a VP because "sneeze" is a verb and "the book" is a valid NP

Possible CFG Solution for Agreement

Can handle agreement like this:

$SgS \rightarrow SgNP\ SgVP$

$PlS \rightarrow PlNp\ PlVP$

$SgNP \rightarrow SgDet\ SgNom$

$PlNP \rightarrow PlDet\ PlNom$

$PlVP \rightarrow PlV\ NP$

$SgVP \rightarrow SgV\ Np$

…

Can use the same trick for all the verb/VP classes

CFG solution for agreement

Pros

- It works and stays within the power of CFGs

Cons

- It is not elegant
- It does not scale
  - The interaction among the various constraints explodes the number of categories and rules in the grammar
- Still overgenerates!
  - Can’t deal with unbounded dependency

CFG accounts for basic syntactic structure in English

But there are problems.
CFG can't handle unbounded dependencies, e.g.

- *Fred likes that piano*
- *John likes the piano [that Fred likes]*
- *John likes the piano [that Sally thinks [that Fred likes]]*
- *John likes the piano [that Sally thinks [that Mary knows [that Fred likes]]]*

There are simpler, more elegant, solutions that go beyond the formal power of CFGs

- LFG, HPSG, Construction grammar, LTAG, etc.

Can we lexicalize a CFG?

A formalism is lexicalized if every elementary structure contains at least one terminal symbol (lexical item).

Is this CFG lexicalized?

i. *S → NP VP*

ii. *VP → really VP*

iii. VP → V NP

iv. V → likes

v. NP → John

vi. NP → Lyn

Can we lexicalize a CFG?

Combine rules, i., iii. and iv. into a single rule:

S → NP likes NP

to get

i. *S → NP likes NP*

ii. *VP → really VP*

iii. *NP → John*

iv. *NP → Lyn*

Dependency in grammar

Bounded dependencies between elements of the same clause:

- Gilbert likes himself/showed himself a movie/
  *thinks Hermione likes himself.
- Gilbert persuaded George to take a bath/
  *that Hermione advised to take a bath

Unbounded dependencies, such as relative clauses which involve dependency between nouns and verbs across clause boundaries:

- A man who Gilbert thinks he likes
- CFG can’t handle unbounded dependencies.
  It overgenerates!

Unbounded dependency

Dependent element is not in the same clause, can be moved further and further away

Topicalized sentences:

- You should *sell that bike* soon.
- *That bike* you should *sell* soon.
- *That bike* I think you should *sell* soon.
- *That bike* I heard they think you should *sell* soon.

Wh-Questions:

- *Which bike* should you *sell* soon?
- *Which bike* do they think you should *sell* soon?

More unbounded dependency

Object relative clauses:

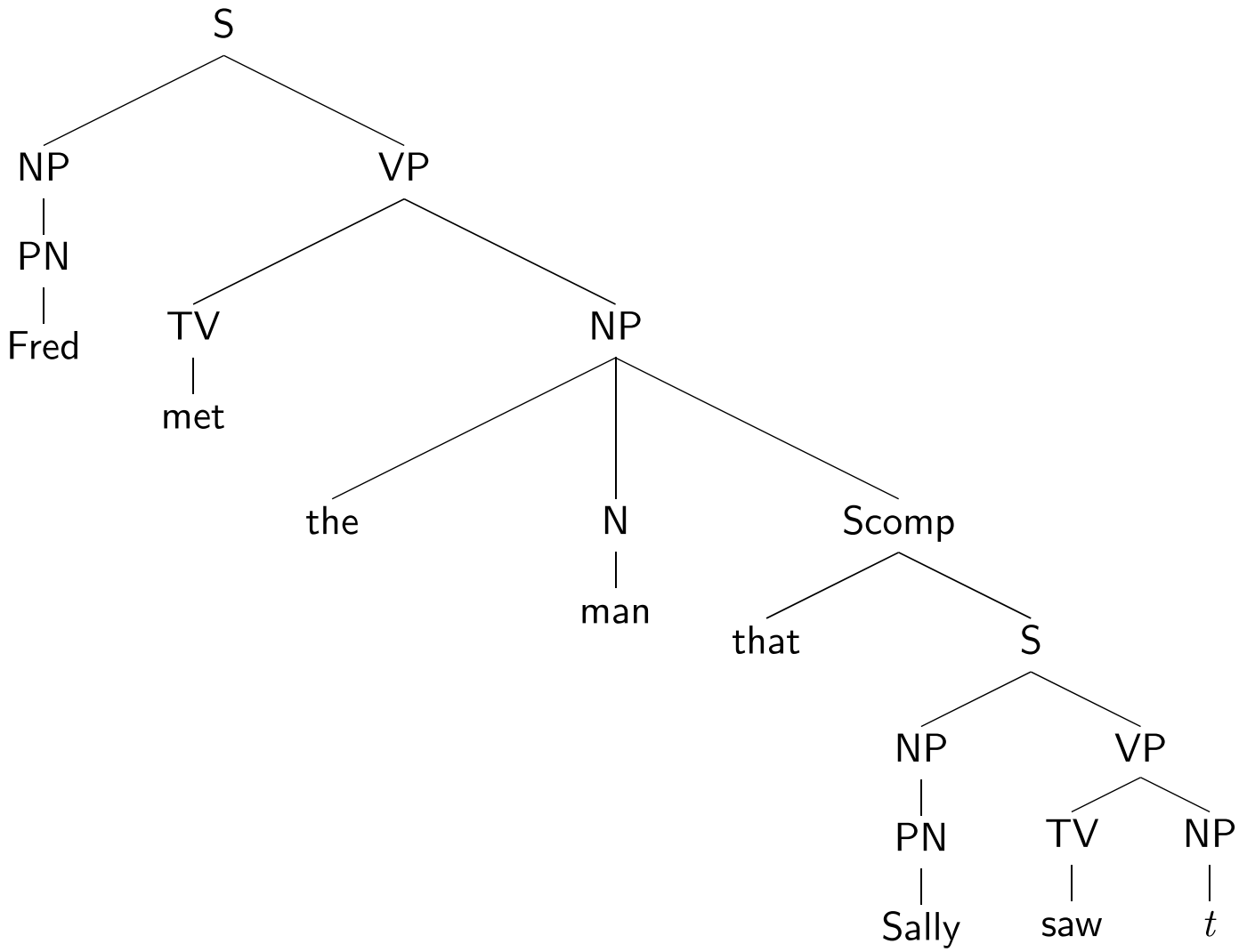
- Fred *likes that man*.
- John likes the *man* [that Fred *likes*]
- John likes the *man* [that Sally thinks [that Fred *likes*]]
- …

Context free generation

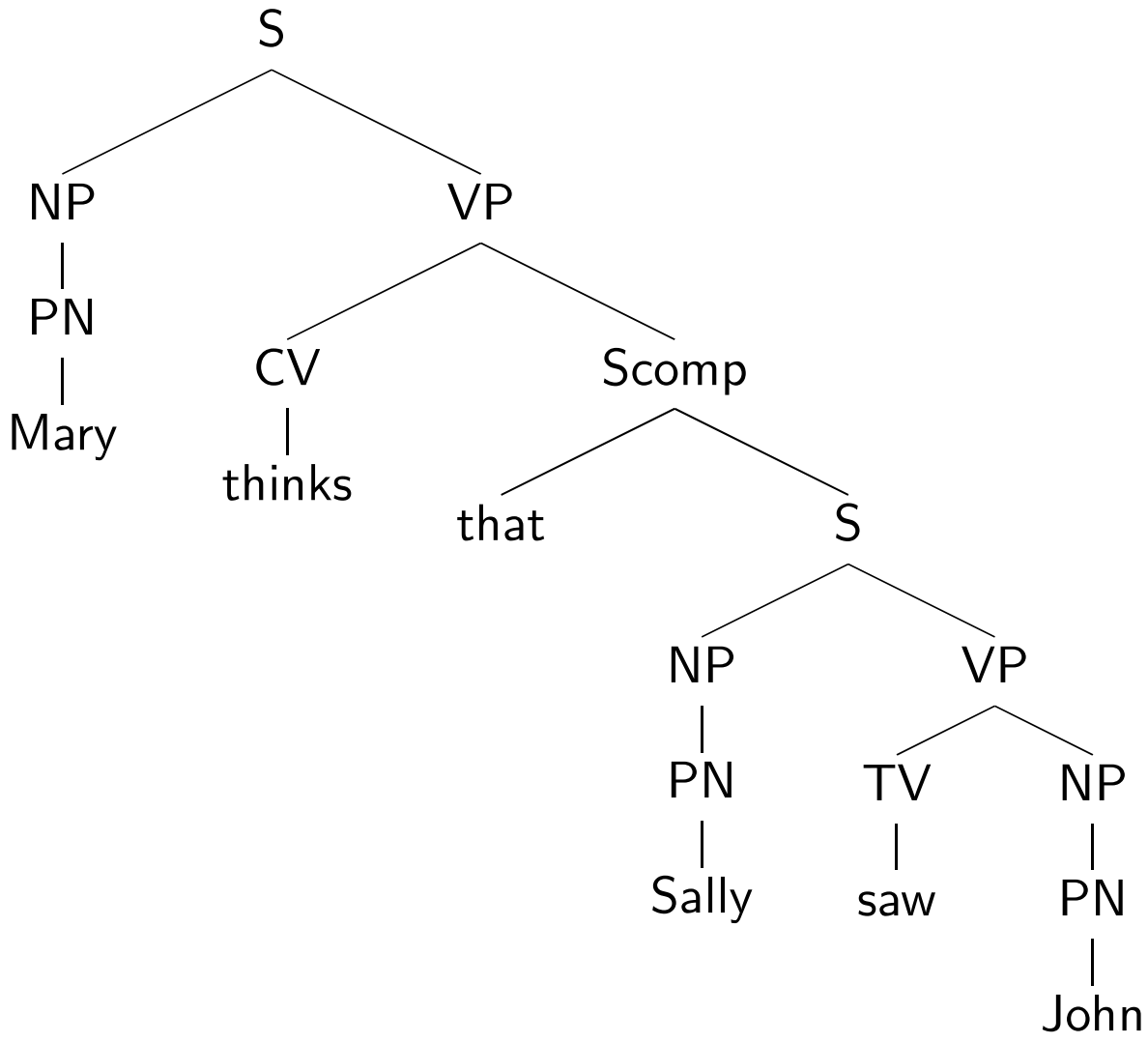
We can write a CFG that generates unbounded dependencies

$$
\begin{aligned}
S &\rightarrow NP \quad VP \\
NP &\rightarrow \begin{cases} the \quad N \quad (Scomp) \\ t \\ \{John, Fred, Sally, Mary, \ldots\} \end{cases} \\
VP &\rightarrow \begin{cases} TV \quad NP \\ CV \quad Scomp \end{cases} \\
Scomp &\rightarrow that \quad S \\
N &\rightarrow \{man, woman, boy, girl, \ldots\} \\
TV &\rightarrow \{see, like, loath, \ldots\} + s \\
CV &\rightarrow \{see, know, dream, \ldots\} + s
\end{aligned}
$$

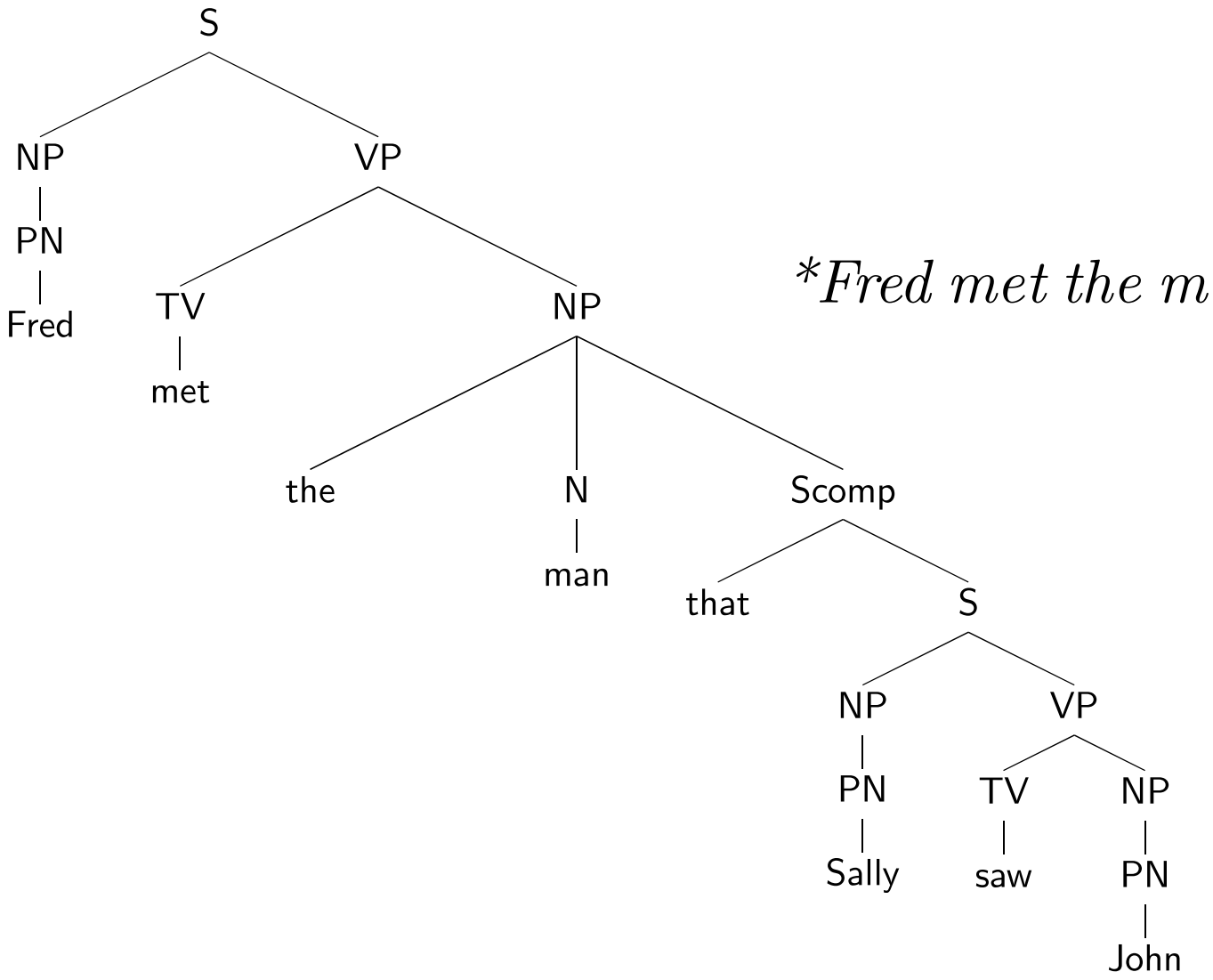
Fred met the man that Sally saw



Mary thinks that Sally saw John



But the grammar overgenerates...



Fred met the man that Sally saw John

Dependency grammars

VRIJE
UNIVERSITEIT
BRUSSEL

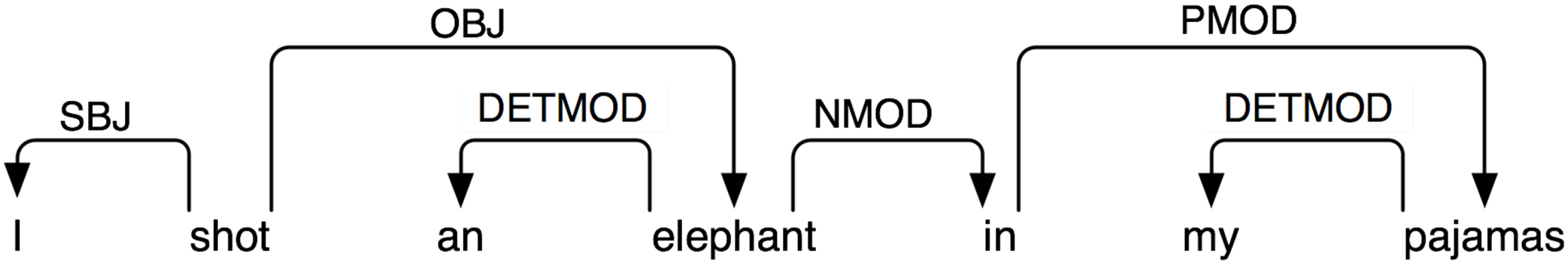
Phrase structure vs. dependencies

Phrase structure grammar: how words and sequences of words combine to form constituents

Dependency grammar: how words relate to other words

- Binary asymmetric relation that holds between a **head** and its **dependents**
- Head of a sentence is usually taken to be the tensed verb
- Every other word is either dependent on the sentence head or connects to it through a path of dependencies

Labeled directed graph



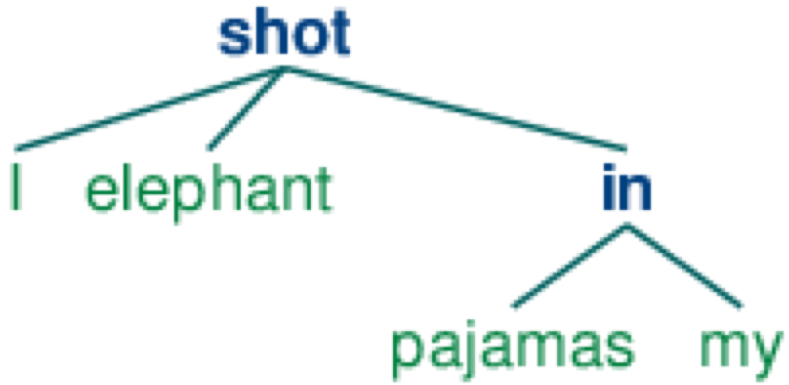
Arrow points from head to dependents

Arcs labeled with grammatical function

Projective dependencies

When edges can be drawn above the words without crossing

Attachment ambiguity (alternative approach):



Non-projective dependencies

Languages with more flexible word order

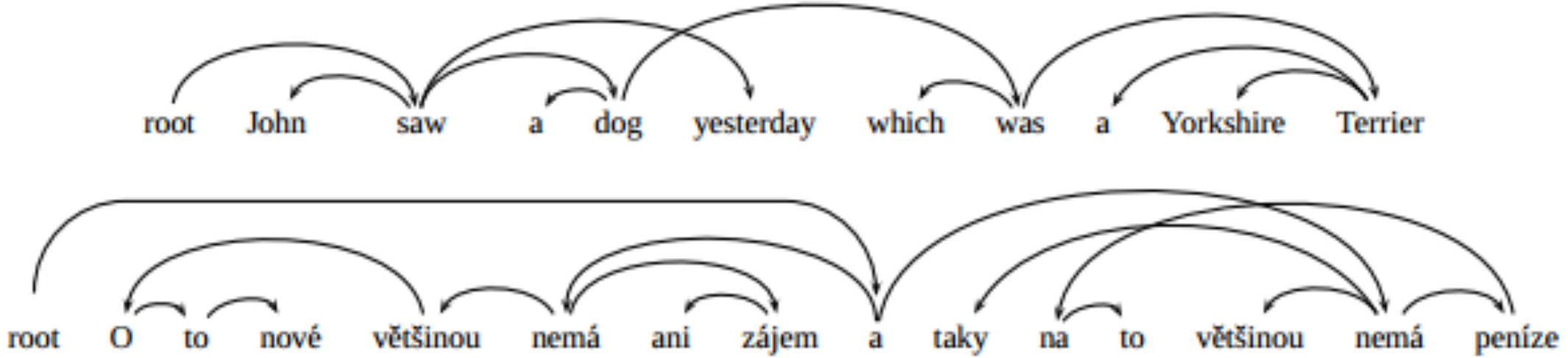


Figure 2: Non-projective dependency trees in English and Czech.

McDonald et al (2005) 'Non-projective Dependency Parsing using Spanning Tree Algorithms' (EMNLP)

Criteria for choosing head/dependent in a construction

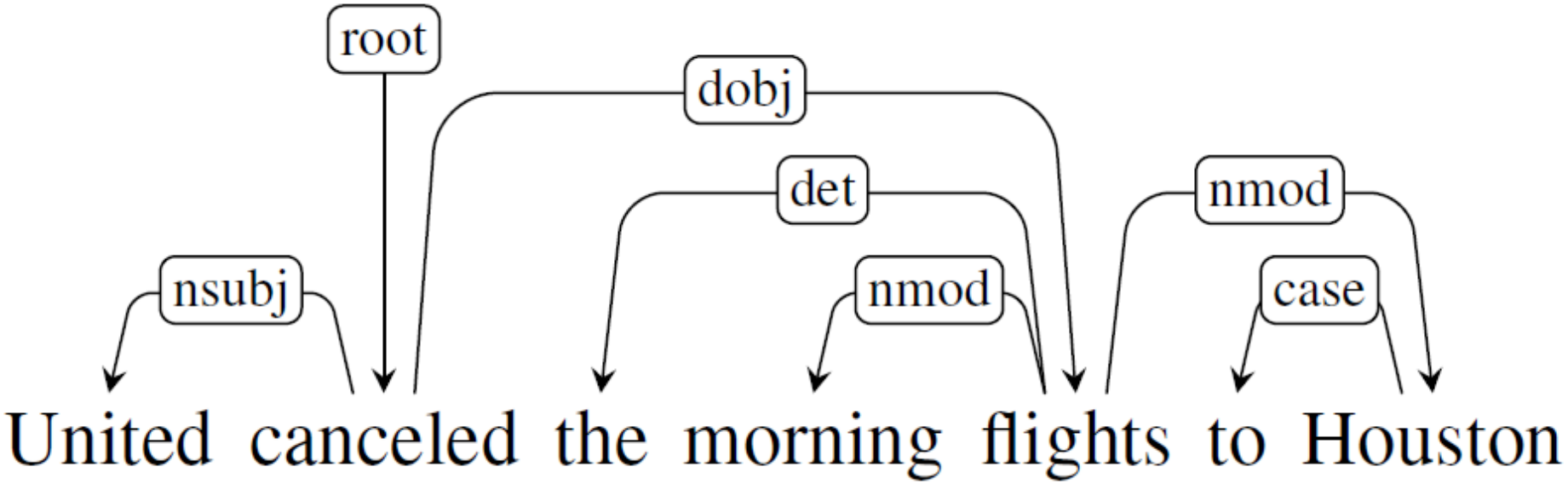
1. H determines the distribution class of C; or alternatively, the external syntactic properties of C are due to H.
2. H determines the semantic type of C.
3. H is obligatory while D may be optional.
4. H selects D and determines whether it is obligatory or optional.
5. The morphological form of D is determined by H (e.g. agreement or case government).

Head-dependency relations

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

Figure 14.2 Selected dependency relations from the Universal Dependency set. (de Marneffe et al., 2014)

Head-dependency relations



Head-dependency relations

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*.

Figure 14.3 Examples of core Universal Dependency relations.

Advantages of DG vs. CFG

Can handle free word order:

- Word order information is separated from dependency information
- Can easily handle unbounded dependencies

(but be aware of non-projectivity)

Dependency relations ≈ semantic relations

- Directly useful for applications like
  - co-reference resolution,
  - question answering,
  - information extraction,
  - semantic analysis

Parsing

Syntactic parsing

A **parser** processes input sentences according to the productions of a grammar, and builds one or more constituent structures that conform to the grammar.

- Beyond n-grams
- Procedural interpretation of the grammar

It searches through the space of trees licensed by a grammar to find one that has the required sentence along its fringe

Why parsing?

Permits a grammar to be evaluated against a collection of test sentences (discover mistakes in grammatical analysis)

Serve as a model of psycholinguistic processing

Many NLP applications involve parsing at some point

- QA systems undergo parsing as an initial step

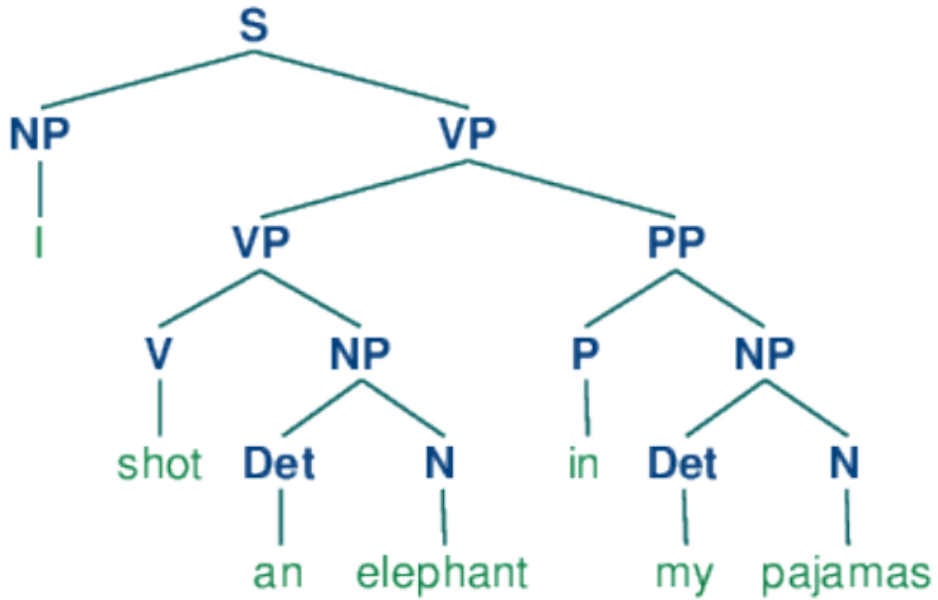
Grammars are pure declarative formalism, they do not specify *how* a parse should be computed

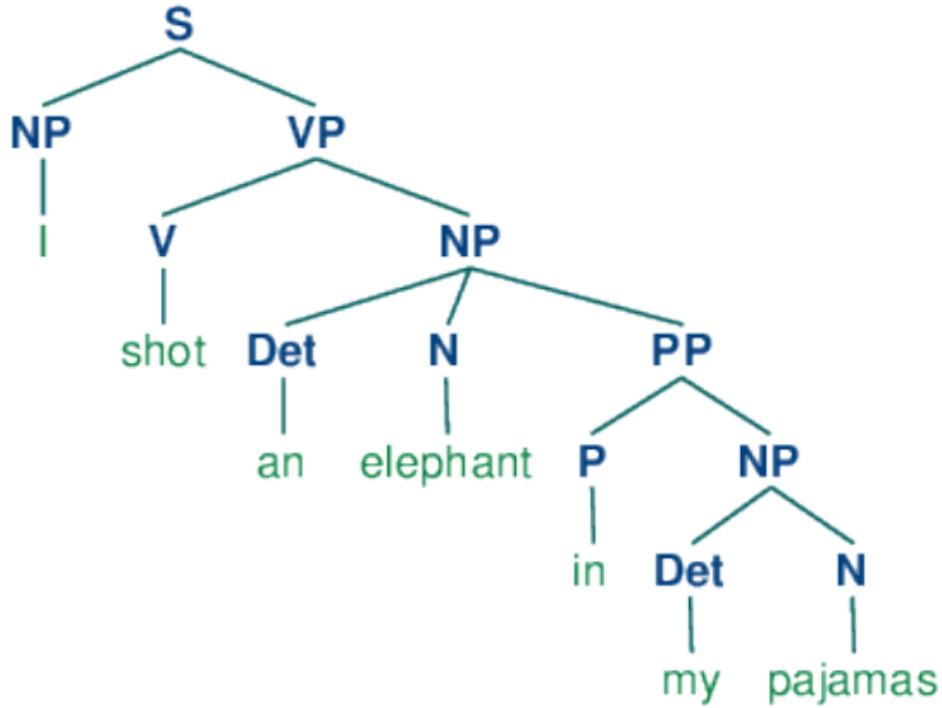
=> PARSING algorithms

Main problem: *structural* ambiguity

Attachment ambiguity

- "While hunting in Africa, I shot an elephant in my pajamas. How he got into my pajamas, I don't know."
  (Groucho Marx movie *Animal Crackers* (1930))





Main problem: *structural* ambiguity

Co-ordination ambiguity

old men and women

1. *[old [men and women]]*
2. *[old men] and [women]*

⇒ *Need for* ***syntactic disambiguation***

Parsing as a search problem: Search the correct parse in the space of possible parses (phrase structure trees and/or dependency trees)

Approaches to Syntactic parsing

Parsing as search: constraints to guide the search algorithm (whichever specific algorithm is chosen)

Two types of constraints

1. TOP-DOWN: Find all trees whose root is the START symbol and cover all the input => goal directed search using prior information (rationalist tradition)
2. BOTTOM-UP: Whatever the final parse of a sentence, the leaves must be the words of the sentence => data-directed search based on available evidence (empiricist tradition)

Approaches to Syntactic parsing

TOP-DOWN parsing

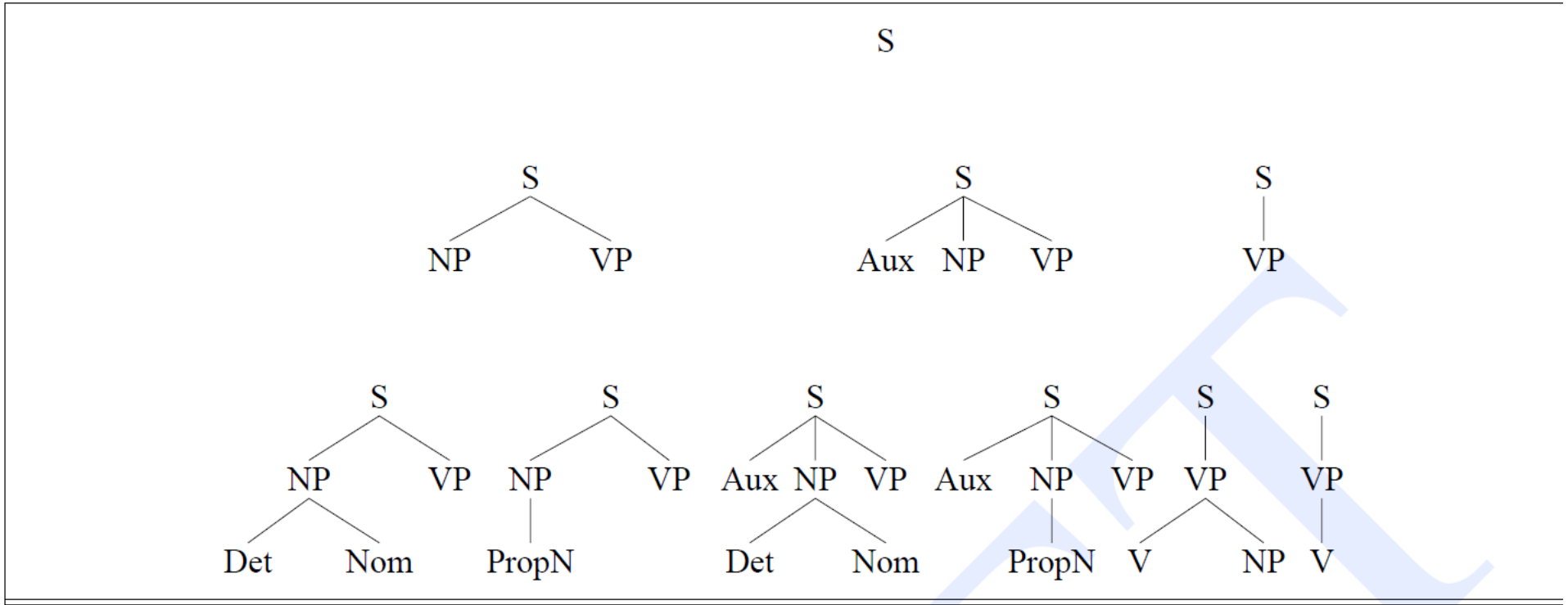
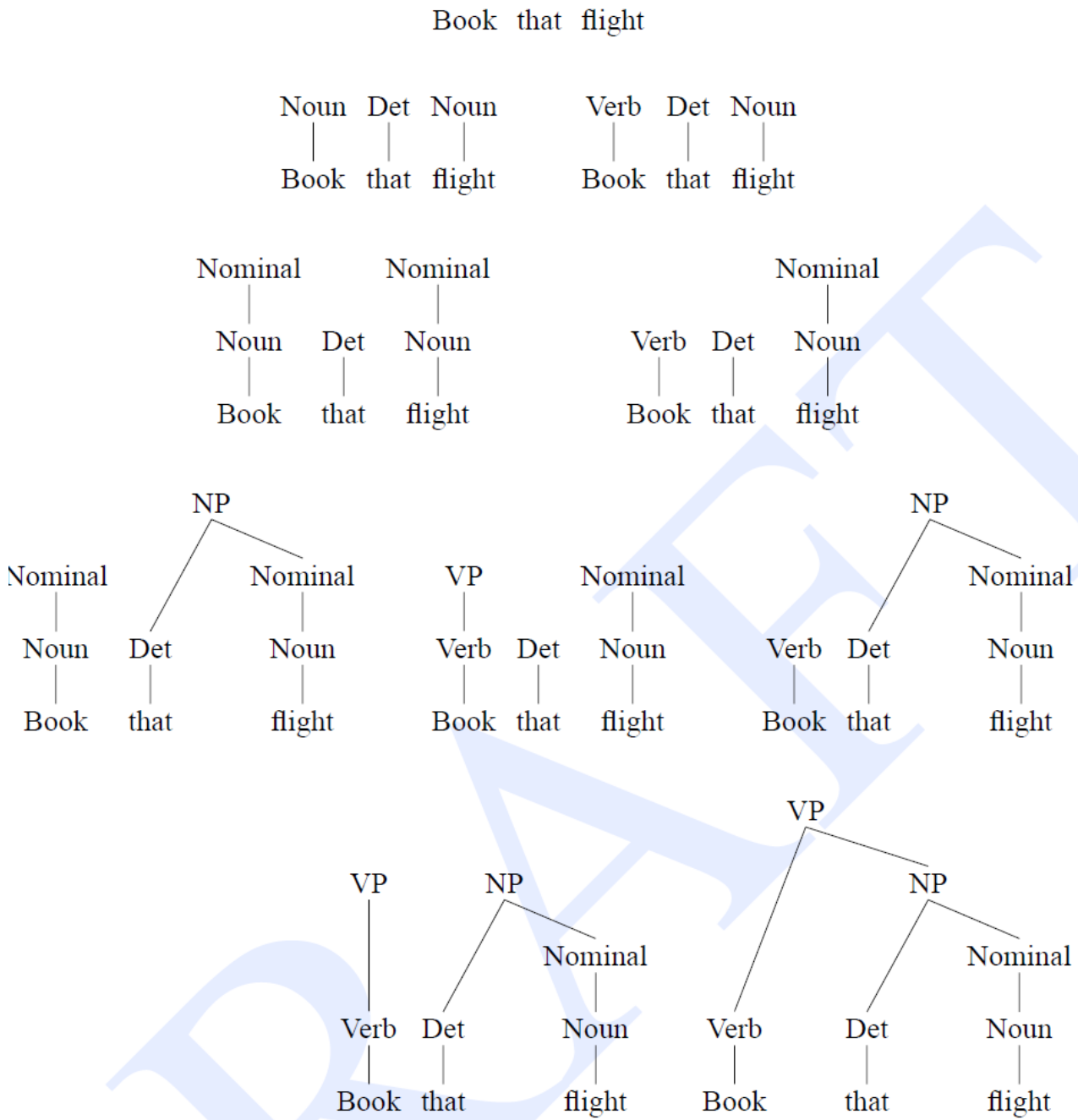


Figure 13.3 An expanding top-down search space. Each ply is created by taking each tree from the previous ply, replacing the leftmost non-terminal with each of its possible expansions, and collecting each of these trees into a new ply.

BOTTOM-UP parsing



An expanding bottom-up search space for the sentence *Book that flight.*
(This figure does not show the final tier of the search with the correct parse tree)

Approaches to syntactic parsing

Choice of parsing algorithm depends on grammar formalism (some algorithms compatible with multiple formalisms)

- CFG parsing (constituency-based)
  - Dynamic programming approaches
    - Cocke-Kasami-Younger (CKY) algorithm (J&M 3rd Ed. Ch.12)
    - CHART parser (J&M 2nd Ed. Ch.13.4.3)
    - Probabilistic CKY (J&M 2nd Ed. Ch.14)
- Unification-based Parsing (feature structures)
  - Fluid Construction Grammar: Transient structures (next week)
- Dependency Parsing (head-dependencies)
  - **Transition-based parsing: Shift-Reduce** (J&M 3rd Ed. Ch.14)
  - Graph-based parsing: maximum spanning tree approach

Dependency parsing

Dependency Trees: G = (V;A)

- a set of vertices V
- set of pairs of vertices A (arcs)

Additional constraints

- 1 root node that has no incoming arcs
- All other vertices have exactly 1 incoming arc
- There is a unique path from the root node to each vertex in V

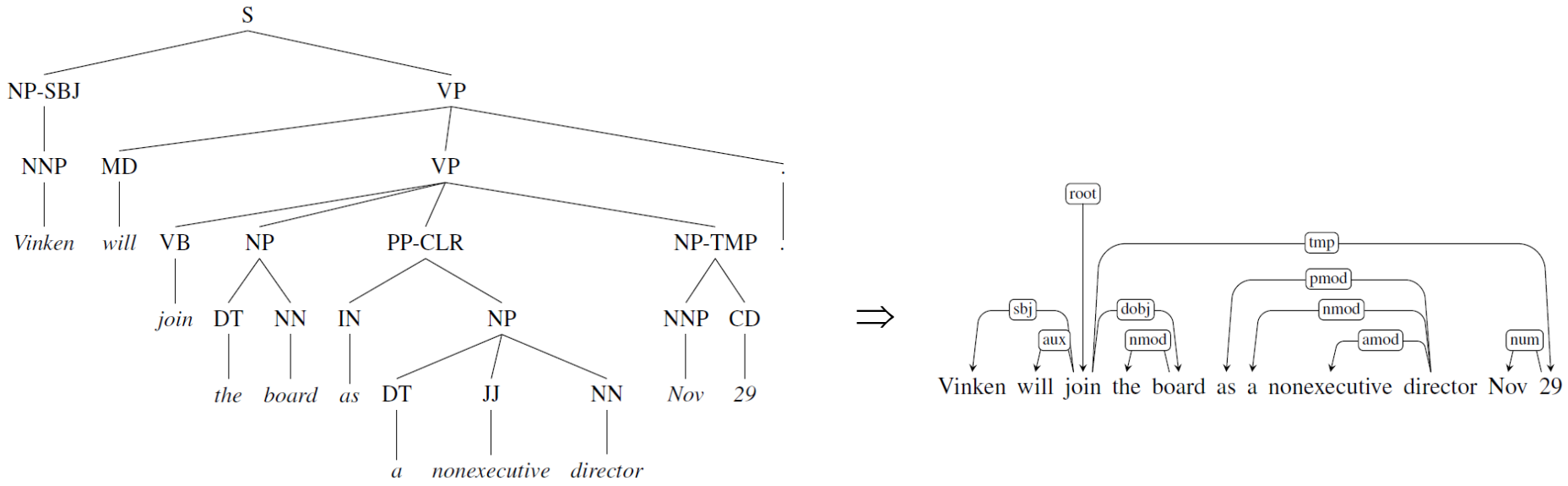
  These constraints ensure that each word has a single head, that the dependency structure is connected, and that there is a single root node from which a unique directed path exists to each of the words in the sentence

- Transition-based approaches: Only projective trees

Dependency treebanks

Training/test data for dependency parsers

- "Native" dependency treebanks for free word-order languages (e.g. Prague Dependency Treebank)
- "translated" from Phrase structure treebanks (Penn)



Transition-based dependency parsing

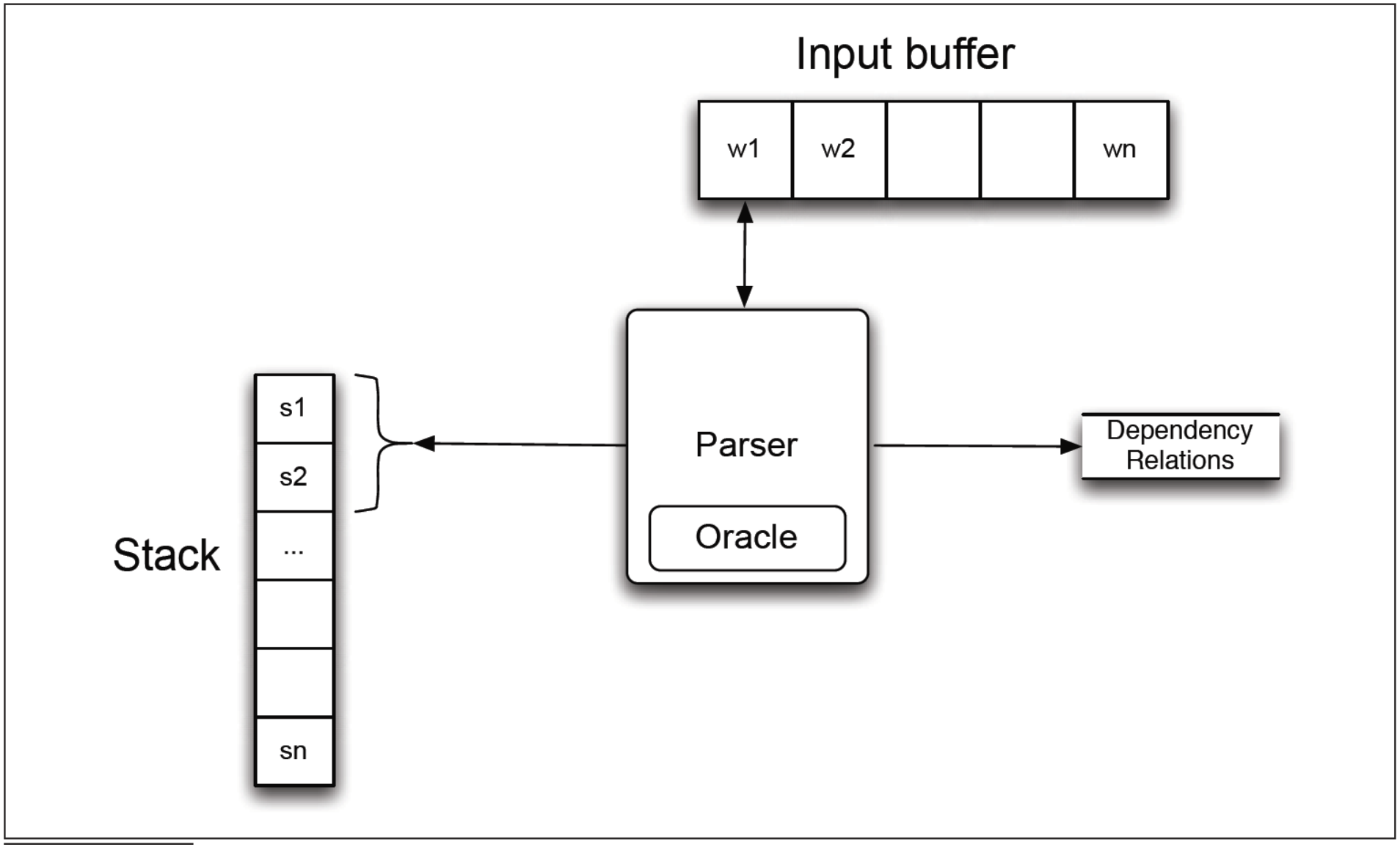
Shift-reduce parsing (Bottom-up parsing technique)

Configuration (= state)

- Dependency grammar defining legal actions
- Stack
- Input buffer: List of tokens
- Output: dependency relations in dep. tree

Parsing process = sequence of transitions through the space of possible configurations

Transition-based dependency parsing



Transition-based dependency parsing

Actions:

- LEFTARC: Assert a head-dependent relation between the word at the top of stack and the word directly beneath it; remove the lower word from the stack. (Not applicable to root as 2nd element)
- RIGHTARC: Assert a head-dependent relation between the second word on the stack and the word at the top; remove the word at the top of the stack;
- SHIFT: Remove the word from the front of the input buffer and push it onto the stack.

Transition-based dependency parsing

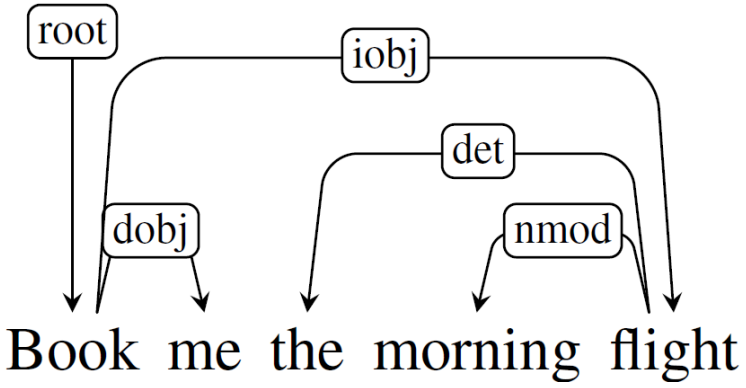
Algorithm:

```
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
```

Figure 14.6 A generic transition-based dependency parser

Example



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	

Figure 14.7 Trace of a transition-based parse.

Transition-based dependency parsing

Labelled dependency trees

Parametrized actions:

- LEFTARC: NSUBJ DOBJ IOBJ ….
- RIGHTARC: NSUBJ DOBJ IOBJ ….
- SHIFT

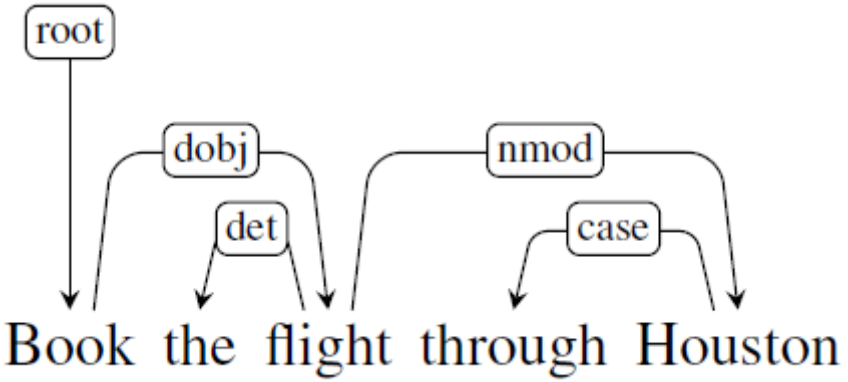
The Oracle (or guide)

Created using training data

Problem: only reduce word (pop from stack) IFF all dependents are processed

Training:

- Choose LEFTARC if it produces a correct head-dependent relation given the reference parse and the current configuration,
- Otherwise, choose RIGHTARC if (1) it produces a correct head-dependent relation given the reference parse and (2) all of the dependents of the word at the top of the stack have already been assigned,
- Otherwise, choose SHIFT.



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

The Oracle (or guide)

Training is done with classifier: SVM, multinom. Regr.

Features (language dependent): Word forms, lemmata, POS, head, dep. Relation

Feature templates:

$$\langle s_1.w, op\rangle, \langle s_2.w, op\rangle \langle s_1.t, op\rangle, \langle s_2.t, op\rangle$$
$$\langle b_1.w, op\rangle, \langle b_1.t, op\rangle \langle s_1.wt, op\rangle$$

Stack	Word buffer	Relations
[root, canceled, flights]	[to Houston]	(canceled → United) (flights → morning) (flights → the)

$$\langle s_1.w = flights, op = shift\rangle$$
$$\langle s_2.w = canceled, op = shift\rangle$$
$$\langle s_1.t = NNS, op = shift\rangle$$
$$\langle s_2.t = VBD, op = shift\rangle$$
$$\langle b_1.w = to, op = shift\rangle$$
$$\langle b_1.t = TO, op = shift\rangle$$
$$\langle s_1.wt = flightsNNS, op = shift\rangle$$

Features based on multiple words

Source	Feature templates		
One word	$s_1.w$	$s_1.t$	$s_1.wt$
	$s_2.w$	$s_2.t$	$s_2.wt$
	$b_1.w$	$b_1.w$	$b_0.wt$
Two word	$s_1.w \circ s_2.w$	$s_1.t \circ s_2.t$	$s_1.t \circ b_1.w$
	$s_1.t \circ s_2.wt$	$s_1.w \circ s_2.w \circ s_2.t$	$s_1.w \circ s_1.t \circ s_2.t$
	$s_1.w \circ s_1.t \circ s_2.t$	$s_1.w \circ s_1.t$	

Figure 14.9 Standard feature templates for training transition-based dependency parsers. In the template specifications s_n refers to a location on the stack, b_n refers to a location in the word buffer, w refers to the wordform of the input, and t refers toval the part of speech of the input.

Corpus formats for syntactically parsed text

VRIJE UNIVERSITEIT BRUSSEL

Corpus formats for parsed text

Many different formats because of

- specifics of underlying grammar formalism
- language-specific syntactic phenomena
- software development needs for specific parsers

Well known formats

- Penn treebank format (S-expr.)
- TIGER XML

```
(IP-MAT (NP-SBJ (PRO I))
        (VBD saw)
        (NP-OB1 (D the)
                (N man)))
```

```
<s id="s5">
  <graph root="s5_504">
    <terminals>
      <t id="s5_1" word="Die" pos="ART" morph="Def.Fem.Nom.Sg"/>
      <t id="s5_2" word="Tagung" pos="NN" morph="Fem.Nom.Sg.*"/>
      <t id="s5_3" word="hat" pos="VVFIN" morph="3.Sg.Pres.Ind"/>
      <t id="s5_4" word="mehr" pos="PIAT" morph="--"/>
      <t id="s5_5" word="Teilnehmer" pos="NN" morph="Masc.Akk.Pl.*"/>
      <t id="s5_6" word="als" pos="KOKOM" morph="--"/>
      <t id="s5_7" word="je" pos="ADV" morph="--"/>
      <t id="s5_8" word="zuvor" pos="ADV" morph="--"/>
    </terminals>
    <nonterminals>
      <nt id="s5_500" cat="NP">
        <edge label="NK" idref="s5_1"/>
        <edge label="NK" idref="s5_2"/>
      </nt>
      <nt id="s5_501" cat="AVP">
        <edge label="CM" idref="s5_6"/>
        <edge label="MO" idref="s5_7"/>
        <edge label="HD" idref="s5_8"/>
      </nt>
      <nt id="s5_502" cat="AP">
        <edge label="HD" idref="s5_4"/>
        <edge label="CC" idref="s5_501"/>
      </nt>
      <nt id="s5_503" cat="NP">
        <edge label="NK" idref="s5_502"/>
        <edge label="NK" idref="s5_5"/>
      </nt>
      <nt id="s5_504" cat="S">
        <edge label="SB" idref="s5_500"/>
        <edge label="HD" idref="s5_3"/>
        <edge label="OA" idref="s5_503"/>
      </nt>
    </nonterminals>
  </graph>
</s>
```

- CoNLL-U developed for shared tasks in dependency parsing of the *Conference on Computational Natural Language Learning (CoNLL)*

CoNNL-U

Three types of lines:

1. Word lines containing the annotation of a word/token in 10 fields separated by single tab characters; see below.
2. Blank lines marking sentence boundaries.
3. Comment lines starting with hash (#).

Word lines contain the following fields:

1. ID: Word index, integer starting at 1 for each new sentence; may be a range for multiword tokens; may be a decimal number for empty nodes.
2. FORM: Word form or punctuation symbol.
3. LEMMA: Lemma or stem of word form.
4. UPOSTAG: Universal part-of-speech tag.
5. XPOSTAG: Language-specific part-of-speech tag; underscore if not available.
6. FEATS: List of morphological features from the universal feature inventory or from a defined language-specific extension; underscore if not available.
7. HEAD: Head of the current word, which is either a value of ID or zero (0).
8. DEPREL: Universal dependency relation to the HEAD (root iff HEAD = 0) or a defined language-specific subtype of one.
9. DEPS: Enhanced dependency graph in the form of a list of head-deprel pairs.
10. MISC: Any other annotation.

CoNNL-U

ID	Form	Lemma	POS	XPOS	Morphl. Features	Head	Dep.rel	Dep.pairs	Misc
# sent_id = 1									
# text = They buy and sell books.									
1	They	they	PRON	PRP	Case=Nom|Number=Plur	2	nsubj	2:nsubj|4:nsubj	_
2	buy	buy	VERB	VBP	Number=Plur|Person=3|Tense=Pres	0	root	0:root	_
3	and	and	CONJ	CC	_	4	cc	4:cc	_
4	sell	sell	VERB	VBP	Number=Plur|Person=3|Tense=Pres	2	conj	0:root|2:conj	_
5	books	book	NOUN	NNS	Number=Plur	2	obj	2:obj|4:obj	SpaceAfter=No
6	.	.	PUNCT	.	_	2	punct	2:punct	_
# sent_id = 2									
# text = I have no clue.									
1	I	I	PRON	PRP	Case=Nom|Number=Sing|Person=1	2	nsubj	_	_
2	have	have	VERB	VBP	Number=Sing|Person=1|Tense=Pres	0	root	_	_
3	no	no	DET	DT	PronType=Neg	4	det	_	_
4	clue	clue	NOUN	NN	Number=Sing	2	obj	_	SpaceAfter=No
5	.	.	PUNCT	.	_	2	punct	_	_

http://universaldependencies.org/format.html

Evaluating parsers

Evaluating parsers

- Exact Match: the whole dependency tree is correct
- Attachment accuracy: head-dependency relation is correct
  - Labeled attachment accuracy
  - Unlabeled attachment accuracy

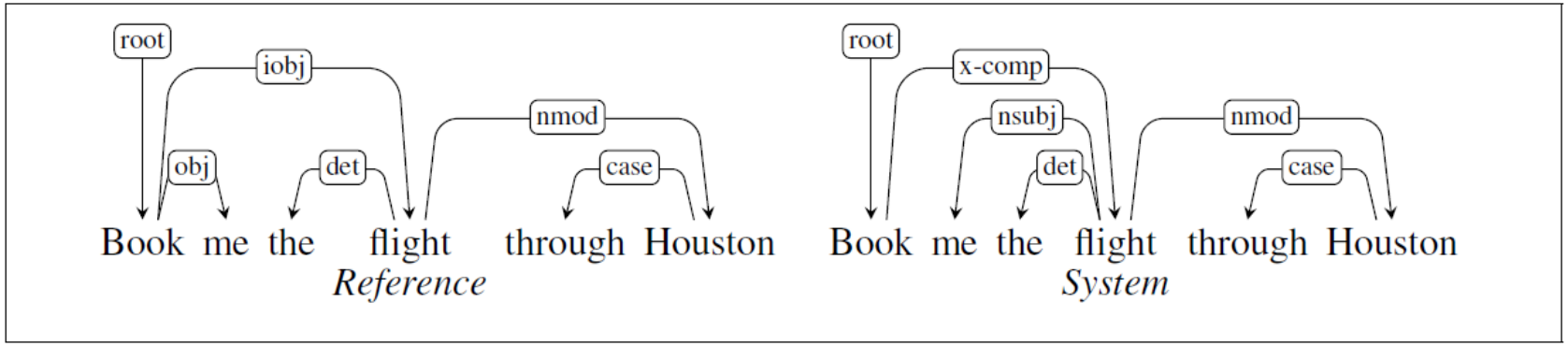


Figure 14.10 Reference and system parses for *Book me the flight through Houston*, resulting in an LAS of 3/6 and an UAS of 4/6.

Homework

Assignment 2

Try out the dependency parsers in NLTK: http://www.nltk.org/_modules/nltk/parse/transitionparser.html

Next week:

Computational construction grammar

Francqui chair: guest lecture

Prof. em. Luc Steels

https://www.arts.kuleuven.be/english/francqui-chair

Language grounding based on cognitive semantics and pragmatics

26 April 2018, 16:00 – 18:00

MSI1 01.16, Mgr. Sencie Instituut Erasmusplein 2, 3000 Leuven

=> No class on 27 April

Assignment 2
Dependency relations & Transition-based parser

VRIJE UNIVERSITEIT BRUSSEL

Part 1: Annotation

Annotate the sentences with universal part-of-speech tags and universal dependency relations

1. I gave an apple to the teacher
2. Mary missed her train to work
3. John gave the teacher a very heavy book
4. The sun shines

5. This is the dog that chased the cat
6. I saw the doctor this morning who is treating me
7. This is the cat that the dog chased
8. John is eager to please

Part 1: Annotation

Format:

- **for all sentences** in the CoNNL-U-format in a separate txt file (utf8)
  - 1 possible parse per sentence
  - only fields 1.ID, 2.Form, 3.Lemma, 4.UPOS, 7.Head, 8.Deprel
  - other fields left unspecified, marked with underscore
- **first sentence** *(I gave an apple to the teacher)* also include in your report
  - A dependency diagram
  - A dependency tree

In your report also discuss these issues:

- Are there s sentences with multiple possible parses?
- Which sentence has non-projective dependency parse is non-projective?

Part 2: Implementation

Write a transition-based dependency parser in Python

- Use the shift-reduce algorithm (see slides, J&M 3rd E. Ch.14)
- Only consider unlabelled dependencies and only 3 actions (leftArc, rightArc, shift)
- INPUT: sentences in CoNNL-U format with only fields *1.ID, 2.Form, 3.Lemma, 4.UPOS* specified
- OUTPUT: (1) CoNNL-U format with fields *7.Head, 8.Deprel* added
  (with all dep-relations simply labelled "DEP")
  (2) a table with the trace of the configurations
- Manually construct a rule-based Oracle
  - Use your annotated **sentences 1 to 4** to extract features templates
  - only use POS-information in the feature templates
  - You can use single-word and double-word feature templates
  - identify features relevant for selecting the correct action
  - Oracle is function that takes set of features for a given configuration, based on your feature template, and uses rules (in the form of (nested) IF statements) to return appropriate action.

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

Part 3: Evaluation

- Run your parser on the sentences 1-4 again
- Manually check the output against your original annotation
- In your report, give the unlabelled attachment accuracy and discuss the errors (if any) and why your think the parser made them

Part 4: Report

Report on parts 1, 2, 3: describe annotation issues, design choices, oracle construction process, and evaluation

Hand in your assignment

April 20 (16h00), PointCarré/email, zip archive with:

1. **report.pdf**
2. **annotation.txt**: manually annotated sentences 1-8 in CoNNL-U
3. **parser.py:** parser code as 1 python script, easily runnable, producing output
4. **input.txt**: input file for your parser with sentences 1-4
5. **output.txt**: parsed output in CoNNL-U format
6. **conftable.txt**: table with the trace of the configurations for s.1-4