Introduction to NLP

262.

Features and Unification

Need for Feature-based Grammars

- Example (number agreement)
  - *The dogs bites*
- Example (count/mass nouns)
  - *many water*
- Example in French (number and person agreement w/subject)
  - Paul est parti, Michelle est partie, Ils sont partis, Elles sont parties
- Example in French (number and person agreement w/direct object)
  - Je l'ai vu (I saw him), Je l'ai vue (I saw her)
- Idea
  - $S \rightarrow$ NP VP

    (but only if the person of the NP is equal to the person of the VP)

Parameterized Grammars

- Parameterized rules, e.g.,
  - S → NP[person,number,”nominative” VP[person,number]
  - VP[person,number] → V[person,number] NP[person,number,”accusative”
  - NP[“first”,number,”nominative”] → DET[number]N[number]
- Appropriate modifications are needed to the parser

Unification Grammars

- Various unification grammar formalisms
  - LFG, HPSG, FUG
- Handle agreement
  - e.g., number, gender, person
- Unification
  - Two constituents can be combined only if their features can 'unify'
- Feature structures (FS or FD)
  - Nested structures that represent all features in an attribute-value matrix
  - Values are typed, so GENDER=PLURAL is not allowed
  - FSs can also be represented as graphs (DAG)
  - Feature paths (from root to a node in the graph)

Example in NLTK

```
import nltk;
from __future__ import print_function
from nltk.featstruct import FeatStruct
from nltk.sem.logic import Variable, VariableExpression, Expression

fs1 = FeatStruct(number='singular', person=3)
print (fs1)
[ number = 'singular' ]
[ person = 3          ]

fs2 = FeatStruct(type='NP', agr=fs1)
print (fs2)
[ agr  = [ number = 'singular' ] ]
[        [ person = 3          ] ]
[                                ]
[ type = 'NP'                    ]
```

http://www.nltk.org/howto/featstruct.html

Feature Unification

- Graph-matching
- Recursive definition
  - Two FSs unify if they can be merged into a consistent FS
  - Leaf nodes unify if:
    - They are the same
    - One can “subsume” the other
    - Special case: One or both are blank

Feature Unification

$$\begin{bmatrix} \text{CAT} & \text{NP} \\ \text{PERSON} & 3 \end{bmatrix} \sqcup \begin{bmatrix} \text{CAT} & \text{NP} \\ \text{NUMBER} & \text{SINGULAR} \end{bmatrix}$$

$$\begin{bmatrix} \text{CAT} & \text{NP} \\ \text{NUMBER} & \text{SINGULAR} \\ \text{PERSON} & 3 \end{bmatrix}$$

Feature Unification

$$\begin{bmatrix} \text{CAT} & \text{NP} \\ \text{PERSON} & 3 \end{bmatrix} \sqcup \begin{bmatrix} \text{CAT} & \text{NP} \\ \text{PERSON} & 1 \end{bmatrix}$$

FAILURE

Example in NLTK

```
fs2 = FeatStruct(type='NP', agr=fs1)
print (fs2)
[ agr  = [ number = 'singular' ] ]
[        [ person = 3          ] ]
[                                ]
[ type = 'NP'                    ]
fs3 = FeatStruct(agr=FeatStruct(number=Variable('?n')),
subj=FeatStruct(number=Variable('?n')))
print(fs3)
[ agr  = [ number = ?n ] ]
[                        ]
[ subj = [ number = ?n ] ]
print(fs2.unify(fs3))
[ agr  = [ number = 'singular' ] ]
[        [ person = 3          ] ]
[                                ]
[ subj = [ number = 'singular' ] ]
[                                ]
[ type = 'NP'                    ]
```

http://www.nltk.org/howto/featstruct.html

Agreement with Features

- $S \rightarrow NP\ VP$

  {NP PERSON} = {VP PERSON}

- $S \rightarrow Aux\ NP\ VP$

  {Aux PERSON} = {NP PERSON}

- Verb $\rightarrow$ bites

  {Verb PERSON} = 3

- Verb $\rightarrow$ bite

  {Verb PERSON} = 1

Types in Semantics

- ***e*** – entities, ***t*** – facts
- **<e,t>** : unary predicates – maps entities to facts
- **<e,<e,t>>** : binary predicates
- **<<e,t>,t>** : type-raised entities
- Examples:
  - “Jorge”, “he”, A123: e
  - “Janice likes cats”: t
  - “likes”: <e,<e,t>>
  - “likes cats”: <e,t>
  - “every person”: <<e,t>,t>

Type Coercion

- Programming languages
  - How is it done in your favorite programming language?
- Examples in natural language
  - I had a coffee this morning (-> one cup of coffee)
  - I tried two wines last night (-> two types of wine)
  - I had fish for dinner (-> some fish, not “a fish”)

Subtypes and Selectional Restrictions

- Type hierarchy
  - object > edible object > fruit > banana
  - noun > count noun
  - noun > mass noun
- Selectional restrictions
  - Some verbs can only take arguments of certain types
  - Example: eat + “edible object”, believe + “idea“
- Selectional restrictions and type coercion (metonymy)
  - I have read this title (“title” -> “book”)
  - I like Shakespeare (“Shakespeare” -> “works by Shakespeare”)

Subcategorization with Features

- VP $\rightarrow$ Verb

  {VP SUBCAT} = {Verb SUBCAT}
  {VP SUBCAT} = INTRANS

- VP $\rightarrow$ Verb NP

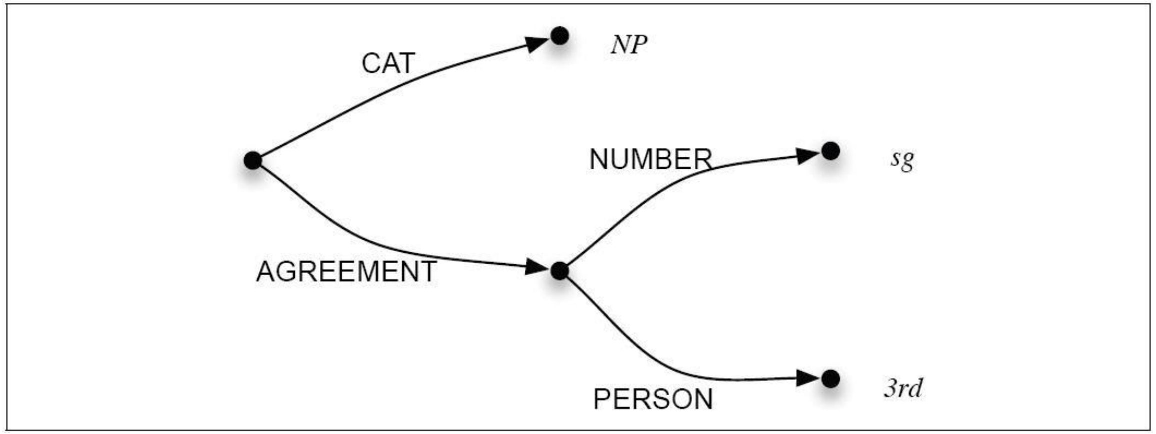
  {VP SUBCAT} = {Verb SUBCAT}
  {VP SUBCAT} = TRANS

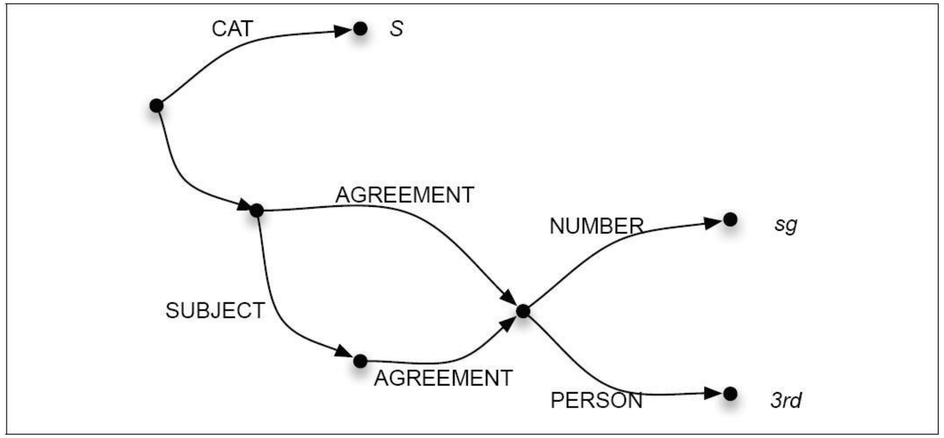
- VP $\rightarrow$ Verb NP NP

  {VP SUBCAT} = {Verb SUBCAT}
  {VP SUBCAT} = DITRANS

Representing FSs as DAGs

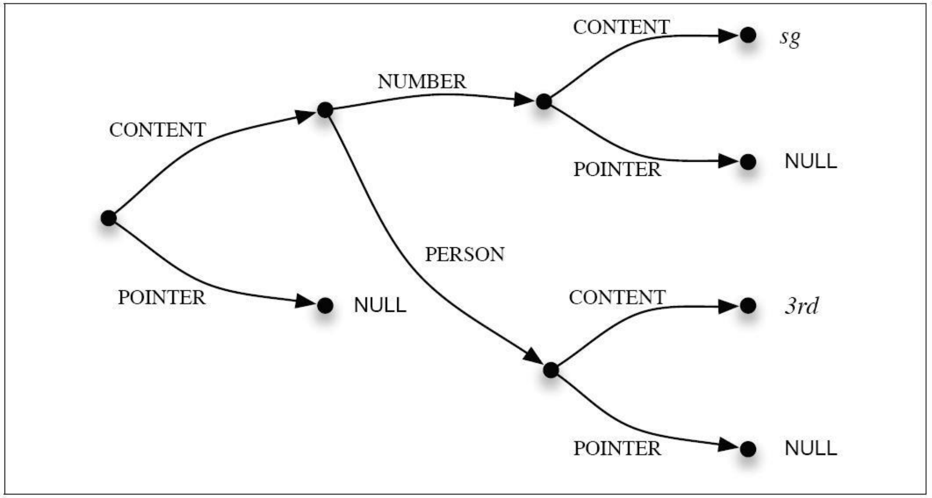
- FS = feature structure
- DAG = directed acyclic graph (not a tree and not an arbitrary graph)

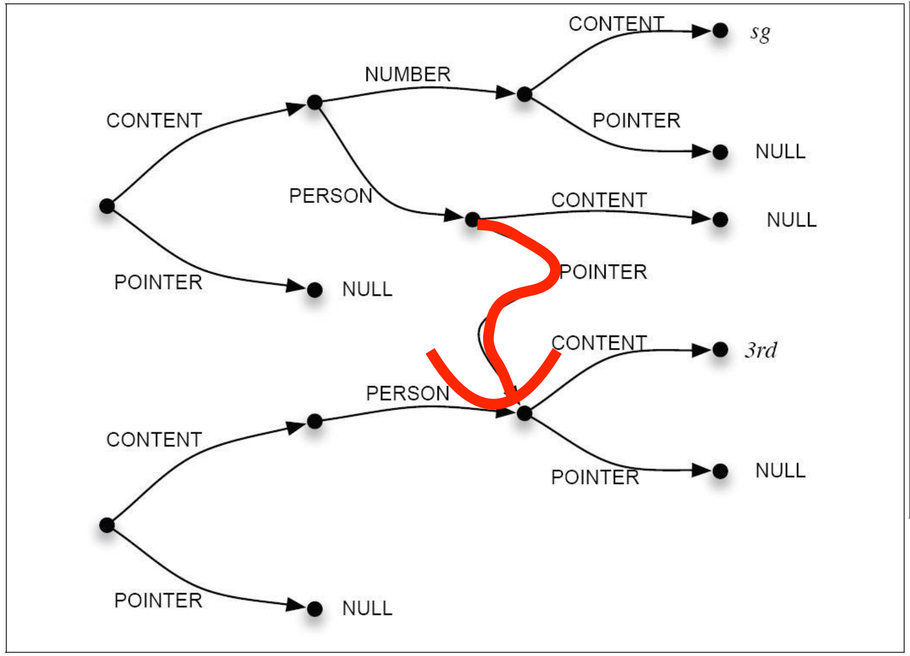


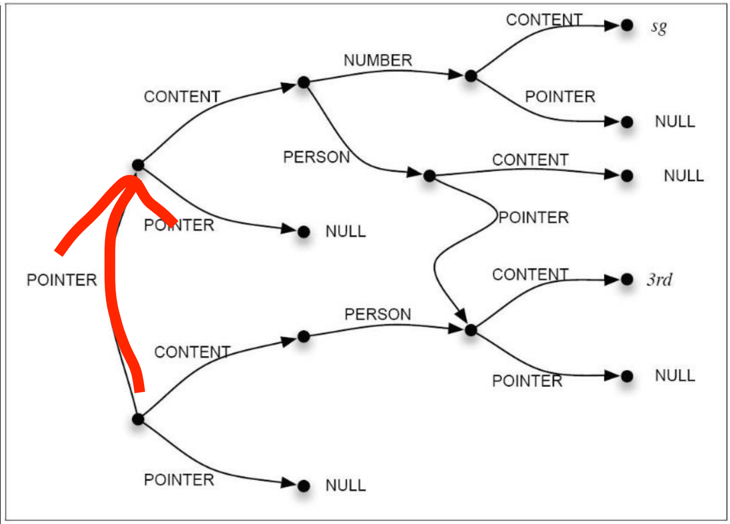


[Example from Jurafsky and Martin]

FS Unification







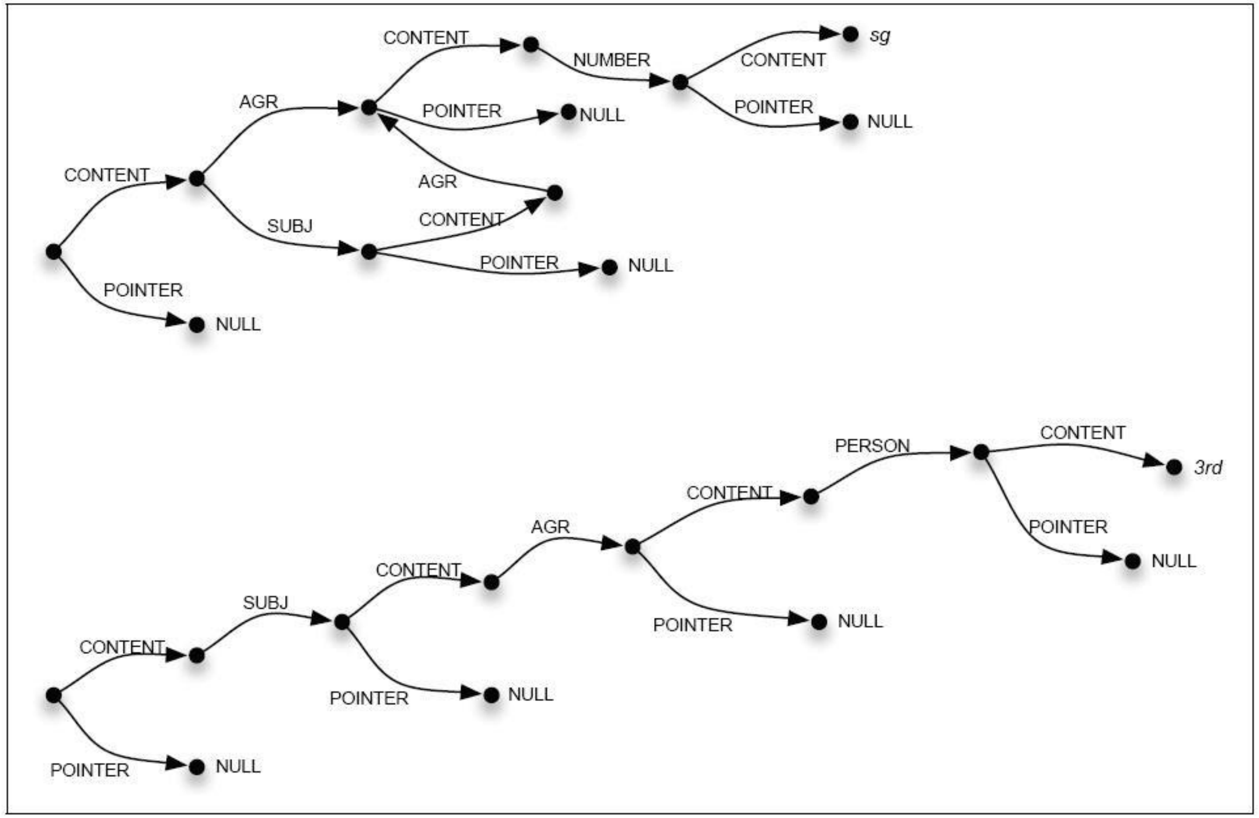
[Example from Jurafsky and Martin]

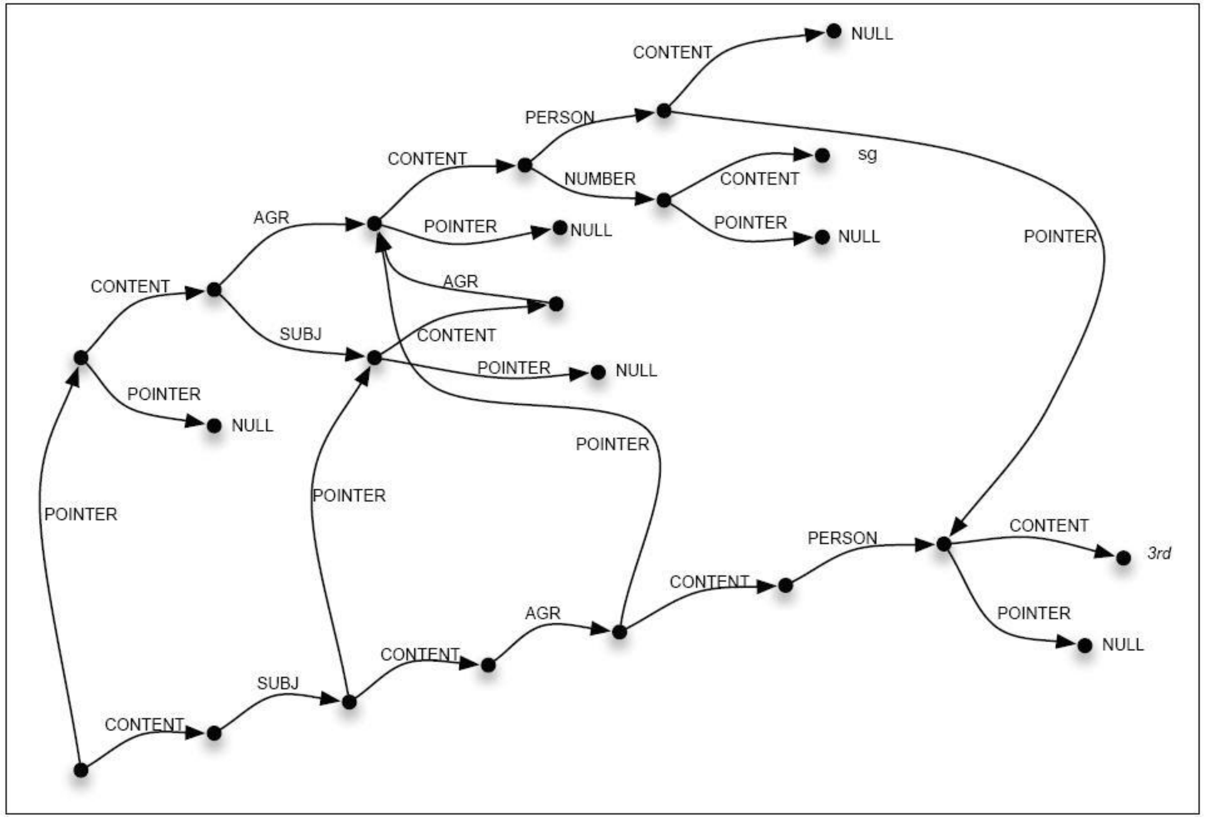
Unification Procedure

```
function UNIFY(f1-orig, f2-orig) returns f-structure or failure

 f1 ← Dereferenced contents of f1-orig
 f2 ← Dereferenced contents of f2-orig

 if f1 and f2 are identical then
    f1.pointer ← f2
    return f2
 else if f1 is null then
   f1.pointer ← f2
   return f2
 else if f2 is null then
   f2.pointer ← f1
   return f1
 else if both f1 and f2 are complex feature structures then
    f2.pointer ← f1
    for each f2-feature in f2 do
      f1-feature ← Find or create a corresponding feature in f1
      if UNIFY(f1-feature.value, f2-feature.value) returns failure then
        return failure
    return f1
 else return failure
```

[Example from Jurafsky and Martin]

FS Unification





[Example from Jurafsky and Martin]

Subsumption

- Unification of a more general concept with a more specific concept
- “undefined” is the most general concept
- “fail” is the least general concept

Subcategorization

Noun Phrase Types		
There	nonreferential there	**There** *is still much to learn*
It	nonreferential it	**It** *was evident that my ideas*
NP	noun phrase	*As he was relating* **his story**
Preposition Phrase Types		
PP	preposition phrase	*couch their message* **in terms**
PPing	gerundive PP	*censured him* **for not having intervened**
PPpart	particle	*turn it* **off**
Verb Phrase Types		
VPbrst	bare stem VP	*she could* **discuss it**
VPto	to-marked infin. VP	*Why do you want* **to know***?*
VPwh	wh-VP	*it is worth considering* **how to write**
VPing	gerundive VP	*I would consider* **using it**
Complement Clause types		
Sfin	finite clause	*maintain* **that the situation was unsatisfactory**
Swh	wh-clause	*it tells us* **where we are**
Sif	whether/if clause	*ask* **whether Aristophanes is depicting a**
Sing	gerundive clause	*see* **some attention being given**
Sto	to-marked clause	*know* **themselves to be relatively unhealthy**
Sforto	for-to clause	*She was waiting* **for him to make some reply**
Sbrst	bare stem clause	*commanded* **that his sermons be published**
Other Types		
AjP	adjective phrase	*thought it* **possible**
Quo	quotes	*asked* **"What was it like?"**

[Example from Jurafsky and Martin]

Subcategorization

Subcat	Example
Quo	asked [$_{Quo}$ "What was it like?"]
NP	asking [$_{NP}$ a question]
Swh	asked [$_{Swh}$ what trades you're interested in]
Sto	ask [$_{Sto}$ him to tell you]
PP	that means asking [$_{PP}$ at home]
Vto	asked [$_{Vto}$ to see a girl called Evelyn]
NP Sif	asked [$_{NP}$ him] [$_{Sif}$ whether he could make]
NP NP	asked [$_{NP}$ myself] [$_{NP}$ a question]
NP Swh	asked [$_{NP}$ him] [$_{Swh}$ why he took time off]

[Example from Jurafsky and Martin]

Introduction to NLP

263.

Tree Adjoining Grammars

The Chomsky Hierarchy

- Regular languages
  - Recognized by Finite State Automata
- Context-free languages
  - Recognized by Push Down Automata
- Context-sensitive languages
- Recursively enumerable languages

The Chomsky Hierarchy

- Regular languages
- Context-free languages
- **Mildly context-sensitive languages**
- Context-sensitive languages
- Recursively enumerable languages

Mildly Context-Sensitive Grammars

- Superset of CFG
- Polynomial parsing
  - $O(n^6)$
- Constant growth property
  - (string length grows linearly)
- Cannot handle the language of strings with the same number of as, bs, and cs.

[https://en.wikipedia.org/wiki/Mildly_context-sensitive_grammar_formalism]

[Example from Julia Hockenmaier]

Other Formalisms

- Tree Substitution Grammar (TSG)
  - Terminals generate entire tree fragments
  - TSG and CFG are formally equivalent

Mildly Context-Sensitive Grammars

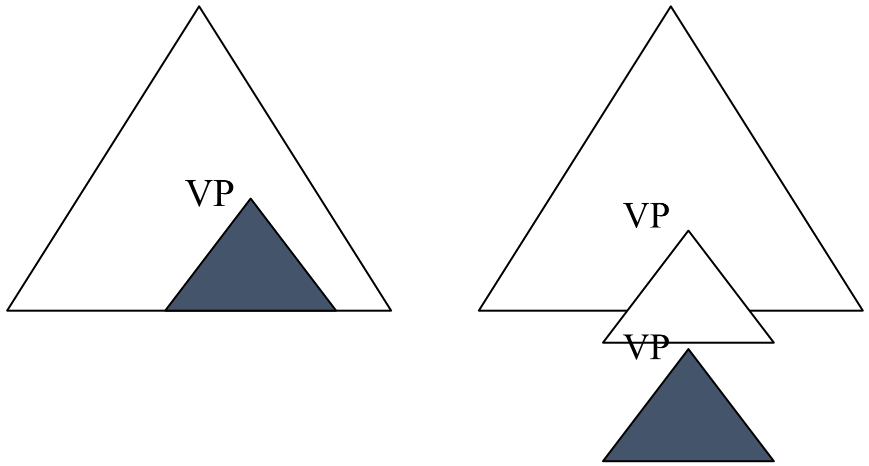
- More powerful than TSG
- Examples:
  - Tree Adjoining Grammar (TAG)
  - Combinatory Categorial Grammar (CCG)

(Tree) Operations in TAG

- Substitution
  - Insert an initial tree to the bottom of a tree
- Adjunction (not in TSG)
  - Insert an auxiliary tree fragment in the middle of a tree
  - Used for long-distance dependencies and for optional modifiers
- Features
  - Each elementary tree has features that can be associated with the top half and with the bottom half
  - Unification is needed
- (Lexicalization)
  - LTAG: each initial or auxiliary tree is labeled with a lexical item

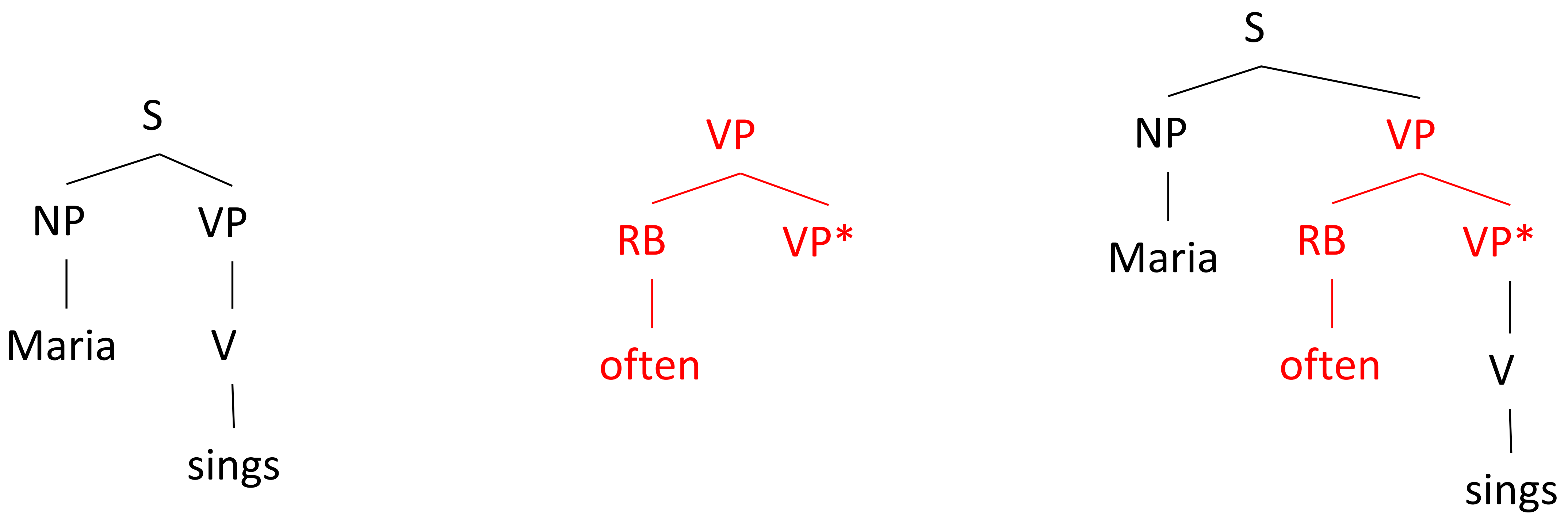
Tree Adjoining Grammar (TAG)

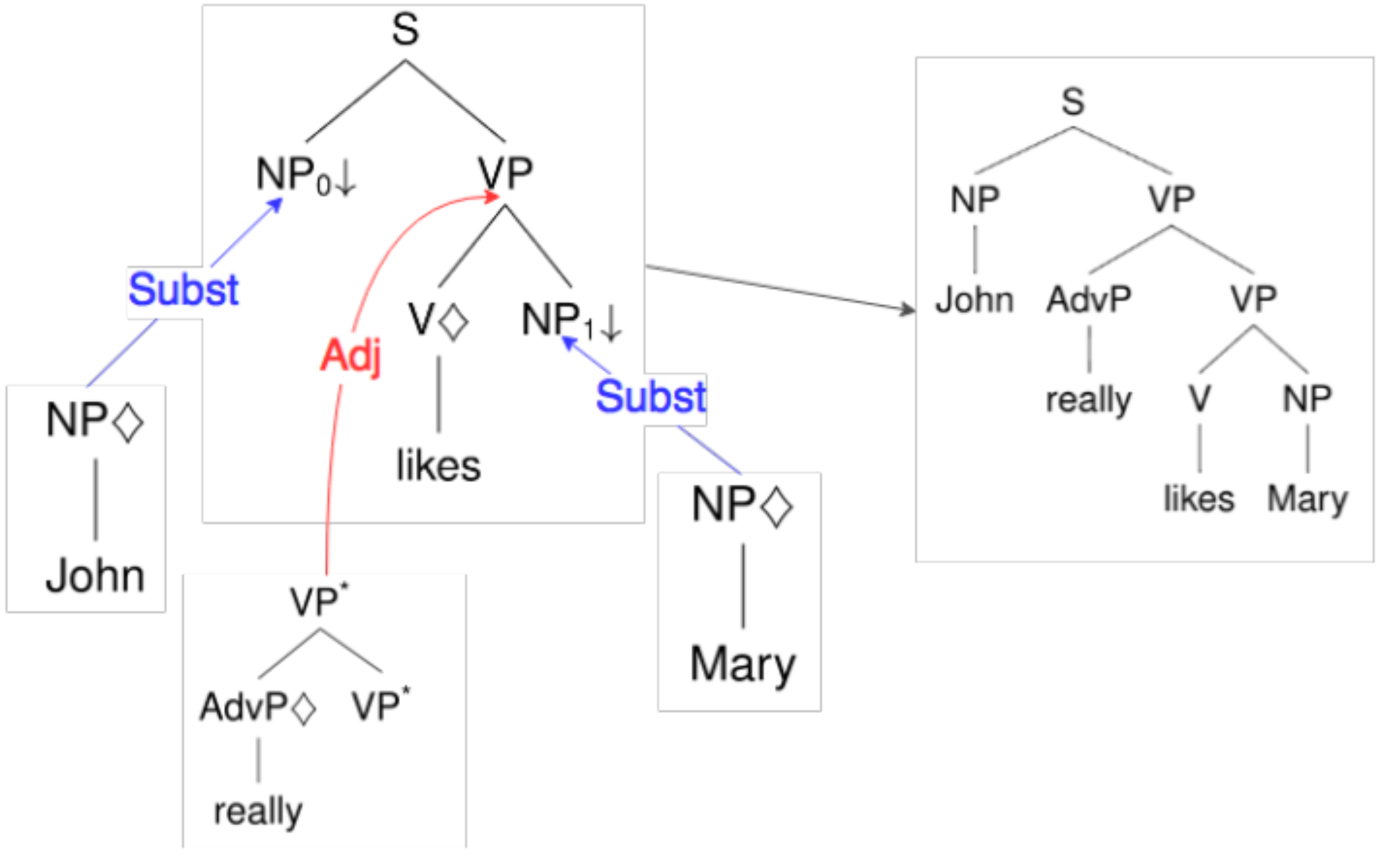
- It can generate languages like $a^n b^n c^n d^n$ or ww (cross-serial dependencies):
  - e.g., Mary gave a book and a magazine to Chen and Mike, respectively.
- Expressive power
  - TAG is formally more powerful than CFG
  - TAG is less powerful than CSG
- Card game online (*broken links*)
  - http://www.ltaggame.com/
  - http://www.ltaggame.com/family.html



TAG Example

- Maria sings
- Maria *often* sings (optional modifier)





Adjunction allows for unbounded recursion while still enforcing agreement.

John smartly occasionally really only likes Mary...

[example from Jungo Kasai]

Introduction to NLP

264.

Combinatory Categorial Grammar (CCG)

Combinatory Categorial Grammar (CCG)

- Complex types
  - E.g., *X/Y* and *X\Y*
  - These take an argument of type *Y* and return an object of type *X*.
  - X/Y – means that Y should appear on the right
  - X\Y – means that Y should appear on the left

Structure of CCG

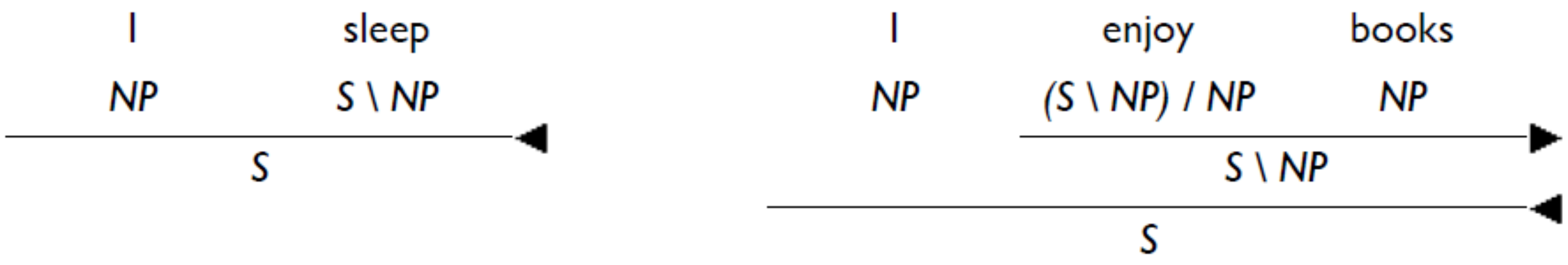
- Categories
- Combinatory rules
- Lexicon

CCG Rules

- Function composition
  - X/Y   Y/Z   ->   X/Z
  - X\Y   Z\X   ->   Z\Y
  - X/Y   Y\Z   ->   X\Z
  - X/Y   Z\X   ->   Z/Y
- Type raising
  - X -> Y/(Y\X)
  - X -> Y\(Y/X)
- Coordination

Example

I	NP
books	NP
sleep	S \ NP
enjoy	(S \ NP) / NP



Example from Jonathan Kummerfeld, Aleka Blackwell, and Patrick Littell

Expressive power

- CCGs can generate the language $a^n b^n c^n d^n$, n>0
- Interesting examples:
  - I like New York
  - I like and hate New York
  - I like and would rather be in New York
  - I gave a book to Chen and a laptop to Jorge
  - I want Chen to stay and Jorge to leave
  - I like and Chen hates, New York
    - Where are the verb phrases?

Examples from Steedman 1996

(6) *Forward Application:* (>)

$$X/Y : f \quad Y : a \quad \Rightarrow \quad X : fa$$

(7) *Backward Application:* (<)

$$Y : a \quad X\backslash Y : f \quad \Rightarrow \quad X : fa$$

They yield derivations like the following:

(8)

$$\begin{array}{ccc}
\text{Mary} & \text{likes} & \text{musicals} \\
\hline
NP_{3sm} : mary' & (S\backslash NP_{3s})/NP : like' & NP : musicals' \\
\cline{2-3}
 & \multicolumn{2}{c}{S\backslash NP_{3s} : like'musicals' \quad >} \\
\hline
\multicolumn{3}{c}{S : like'musicals'mary' \quad <}
\end{array}$$

Examples from Steedman 1996

(9) *Coordination:* $(< \& >)$

$$X \quad conj \quad X \quad \Rightarrow \quad X$$

(10)

$$\begin{array}{ccccc}
\text{I} & \text{loathe} & \text{and} & \text{detest} & \text{opera} \\
\overline{NP} & \overline{(S\backslash NP)/NP} & \overline{CONJ} & \overline{(S\backslash NP)/NP} & \overline{NP} \\
 & \multicolumn{3}{c}{\underline{\hspace{12em}}_{<\&>}} & \\
 & \multicolumn{3}{c}{(S\backslash NP)/NP} & \\
 & \multicolumn{4}{c}{\underline{\hspace{16em}}_{>}} \\
 & \multicolumn{4}{c}{S\backslash NP} \\
\multicolumn{5}{c}{\underline{\hspace{18em}}_{<}} \\
\multicolumn{5}{c}{S}
\end{array}$$

(13) *Forward Composition:* $(> B)$

$$X/Y : f \quad Y/Z : g \quad \Rightarrow \quad X/Z : \lambda x.f(gx)$$

(14)

$$\begin{array}{cccccc}
\text{I} & \text{requested} & \text{and} & \text{would} & \text{prefer} & \text{musicals} \\
\overline{NP} & \overline{(S\backslash NP)/NP} & \overline{CONJ} & \overline{(S\backslash NP)/VP : will'} & \overline{VP/NP : prefer'} & \overline{NP} \\
 & & & \multicolumn{2}{c}{\underline{\hspace{14em}}_{>B}} & \\
 & & & \multicolumn{2}{c}{(S\backslash NP)/NP : \lambda x.\lambda y.will'(prefer'x)y} & \\
 & \multicolumn{4}{c}{\underline{\hspace{20em}}_{<\&>}} & \\
 & \multicolumn{4}{c}{(S\backslash NP)/NP} & \\
 & \multicolumn{5}{c}{\underline{\hspace{24em}}_{>}} \\
 & \multicolumn{5}{c}{S\backslash NP} \\
\multicolumn{6}{c}{\underline{\hspace{26em}}_{<}} \\
\multicolumn{6}{c}{S}
\end{array}$$

Examples from Steedman 1996

(18)

$$
\begin{array}{lcccc}
\text{introduce} & \text{Bill} & \text{to Sue} & \text{and} & \text{Harry} \quad \text{to George} \\
\overline{(VP/PP)/NP} & \overline{(VP/PP)\backslash((VP/PP)/NP)}^{<T} & \overline{VP\backslash(VP/PP)}^{<T} & \overline{CONJ} & \overline{VP\backslash((VP/PP)/NP)}^{<B} \\
 & \multicolumn{2}{c}{\overline{\qquad VP\backslash((VP/PP)/NP) \qquad}^{<B}} & & \\
 & \multicolumn{4}{c}{\overline{\qquad\qquad VP\backslash((VP/PP)/NP) \qquad\qquad}^{<\&>}} \\
\multicolumn{5}{c}{\overline{\qquad\qquad\qquad VP \qquad\qquad\qquad}^{<}}
\end{array}
$$

CCG in NLTK

```
from nltk.ccg import chart, lexicon
lex = lexicon.parseLexicon('''
:- S, NP, N, VP
Det :: NP/N
Pro :: NP
Modal :: S\\NP/VP
TV :: VP/NP
DTV :: TV/NP
the => Det
that => Det
that => NP
I => Pro
you => Pro
we => Pro
chef => N
cake => N
children => N
dough => N
```

```
will => Modal
should => Modal
might => Modal
must => Modal
and => var\\.,var/.,var
to => VP[to]/VP
without => (VP\\VP)/VP[ing]
be => TV
cook => TV
eat => TV
cooking => VP[ing]/NP
give => DTV
is => (S\\NP)/NP
prefer => (S\\NP)/NP
which => (N\\N)/(S/NP)
persuade => (VP/VP[to])/NP
''')
```

http://www.nltk.org/howto/ccg.html

CCG in NLTK

```
parser = chart.CCGChartParser(lex, chart.DefaultRuleSet)
for parse in parser.parse("you prefer that cake".split()):
    chart.printCCGDerivation(parse)
    break
```

```
 you    prefer      that   cake
 NP   ((S\NP)/NP)  (NP/N)   N
-----
 NP
     -------------
      ((S\NP)/NP)
                  --------
                   (NP/N)
                          ------
                            N
                  --------------->
                       NP
     ---------------------------->
              (S\NP)
--------------------------------<
              S
```

http://www.nltk.org/howto/ccg.html

```
for parse in parser.parse("that is the cake which you prefer".split()):
    chart.printCCGDerivation(parse)
    break
```

```
 that      is        the    cake      which       you    prefer
  NP   ((S\NP)/NP)  (NP/N)   N    ((N\N)/(S/NP))  NP   ((S\NP)/NP)
------
  NP
      -------------
       ((S\NP)/NP)
                   --------
                    (NP/N)
                           ------
                             N
                                 ----------------
                                  ((N\N)/(S/NP))
                                                 -----
                                                  NP
                                                 ----->T
                                              (S/(S\NP))
                                                      -------------
                                                       ((S\NP)/NP)
                                                 ------------------>B
                                                       (S/NP)
                                 ---------------------------------->
                                               (N\N)
                           ----------------------------------------<
                                              N
                   ------------------------------------------------>
                                          NP
      ------------------------------------------------------------->
                                 (S\NP)
-------------------------------------------------------------------<
                                 S
```

http://www.nltk.org/howto/ccg.html

CCG

- NACLO problem from 2014 (in two parts)
- Authors: Jonathan Kummerfeld, Aleka Blackwell, and Patrick Littell
- http://www.nacloweb.org/resources/problems/2014/N2014-O.pdf
- http://www.nacloweb.org/resources/problems/2014/N2014-OS.pdf
- http://www.nacloweb.org/resources/problems/2014/N2014-P.pdf
- http://www.nacloweb.org/resources/problems/2014/N2014-PS.pdf

CCG Parsing

- CKY works fine
- http://openccg.sourceforge.net/

Parse 1:

Fruit flies like a banana

N S[dcl]\NP ((S\NP)\(S\NP))/NP NP[nb]/N N

NP NP[nb]

(S\NP)\(S\NP)

S[dcl]\NP

S[dcl]

Parse 2:

Fruit flies like a banana

N (S[dcl]\NP)/PP PP/NP NP[nb]/N N

NP NP[nb]

PP

S[dcl]\NP

S[dcl]

Parse 3:

Fruit flies like a banana

N S[dcl]\NP (S\S)/NP NP[nb]/N N

NP NP[nb]

S[dcl] S\S

S[dcl]

Parse 4:

Fruit flies like a banana

N N\N (S[dcl]\NP)/NP NP[nb]/N N

N NP[nb]

NP S[dcl]\NP

S[dcl]

Parse 5:

Fruit flies like a banana

N/N N (S[dcl]\NP)/NP NP[nb]/N N

N NP[nb]

NP S[dcl]\NP

S[dcl]

http://4.easy-ccg.appspot.com/do_parse?sentence=Fruit+flies+like+a+banana&nbest=5

Exercise

- How do you represent the following categories in CCG
  - Nouns N
  - Adjectives N/N
  - Articles NP/N
  - Prepositions (NP\NP)/NP
  - Transitive verbs (S\NP)/NP
  - Intransitive verbs S\NP

CCG for Dutch Cross-Serial Dependencies

… because I$_1$ Cecilia$_2$ Henk$_3$ the hippopotamuses$_4$ saw$_1$ help$_2$ feed$_{3,4}$.

$$\begin{array}{cccccccc}
\textit{dat} & \textit{ik} & \textit{Cecilia} & \textit{Henk} & \textit{de nijlpaarden} & \textit{zag} & \textit{helpen} & \textit{voeren} \\
\hline
 & NP_1 & NP_2 & NP_3 & NP_4 & ((S\backslash NP_1)\backslash NP_2)/VP & (VP\backslash NP_3)/VP & VP\backslash NP_4
\end{array}$$

$$\frac{(VP\backslash NP_3)/VP \quad VP\backslash NP_4}{(VP\backslash NP_3)\backslash NP_4} >\mathbf{B}_\times$$

$$\frac{((S\backslash NP_1)\backslash NP_2)/VP \quad (VP\backslash NP_3)\backslash NP_4}{(((S\backslash NP_1)\backslash NP_2)\backslash NP_3)\backslash NP_4} >\mathbf{B}^2_\times$$

$$\frac{NP_4 \quad (((S\backslash NP_1)\backslash NP_2)\backslash NP_3)\backslash NP_4}{((S\backslash NP_1)\backslash NP_2)\backslash NP_3} <$$

$$\frac{NP_3 \quad ((S\backslash NP_1)\backslash NP_2)\backslash NP_3}{(S\backslash NP_1)\backslash NP_2} <$$

$$\frac{NP_2 \quad (S\backslash NP_1)\backslash NP_2}{S\backslash NP_1} <$$

$$\frac{NP_1 \quad S\backslash NP_1}{S} <$$

[Example from Stephen Clark]

CCGBank

- Hockenmaier and Steedman (2005)

Sentence 1

```
{S[dcl] {S[dcl] {NP {NP {NP {NP {N {N/N  Pierre}
                                  {N  Vinken}}}
                            {,  ,}}
                        {NP\NP {S[adj]\NP {NP {N {N/N  61}
                                                  {N  years}}}
                                              {(S[adj]\NP)\NP  old}}}}
                    {,  ,}}
                {S[dcl]\NP {(S[dcl]\NP)/(S[b]\NP)  will}
                           {S[b]\NP {S[b]\NP {(S[b]\NP)/PP {((S[b]\NP)/PP)/NP  join}
                                                           {NP {NP[nb]/N  the}
                                                               {N  board}}}
                                             {PP {PP/NP  as}
                                                 {NP {NP[nb]/N  a}
                                                     {N {N/N  nonexecutive}
                                                        {N  director}}}}}
                                    {(S\NP)\(S\NP) {((S\NP)\(S\NP))/N  Nov.}
                                                   {N  29}}}}}
   {.  .}}
```

Pierre	**(N/N)**	Vinken		
61	**(N/N)**	years		
old	**((S[adj]\NP)\NP)**	*Vinken*	years	
will	**((S[dcl]\NP)/(S[b]\NP))**	Vinken	join	
join	**(((S[b]\NP)/PP)/NP)**	*Vinken*	as	board
the	**(NP[nb]/N)**	board		
as	**(PP/NP)**	director		
a	**(NP[nb]/N)**	director		
nonexecutive	**(N/N)**	director		
Nov.	**(((S\NP)\(S\NP))/N)**		join	29