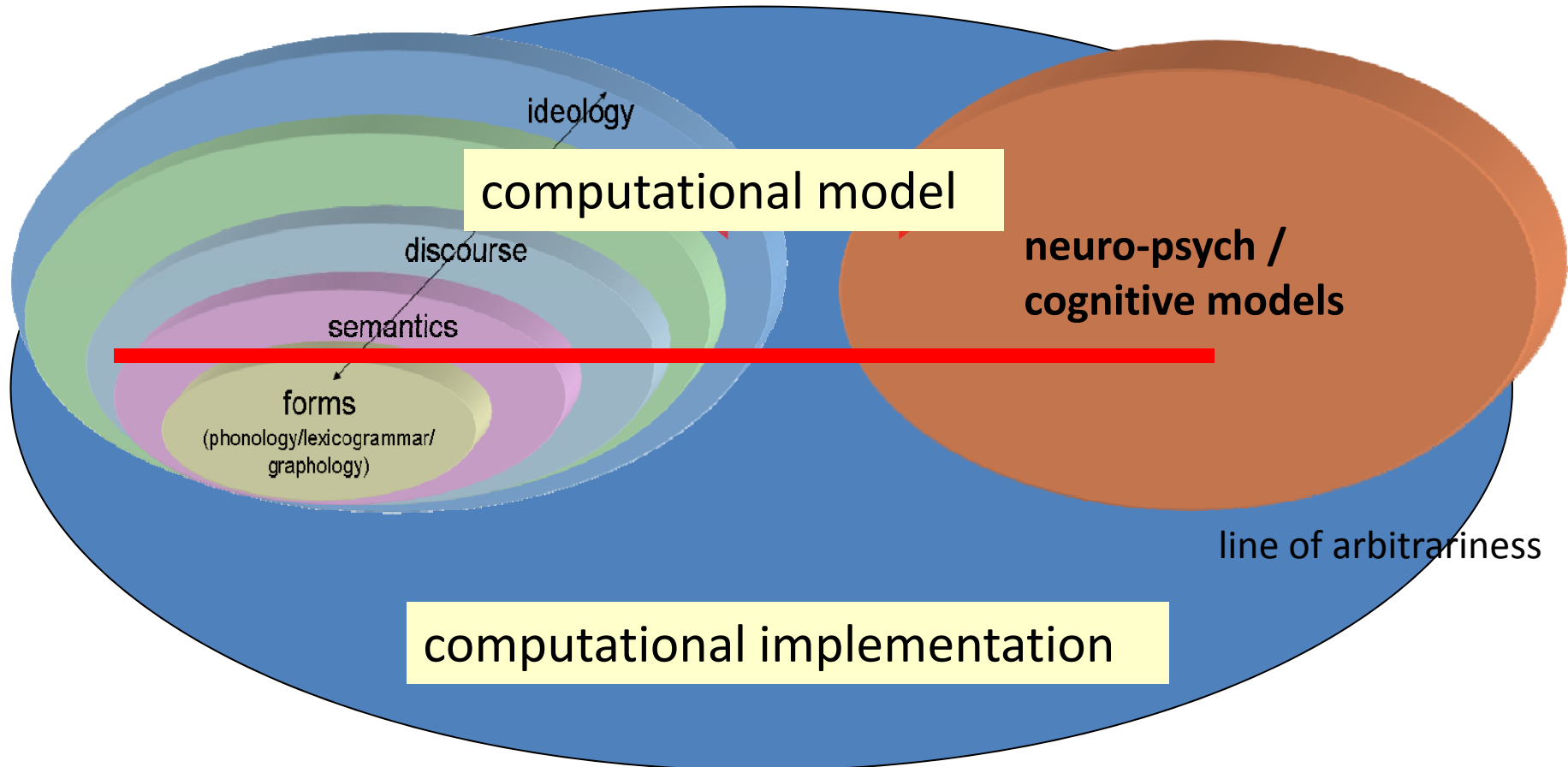


Session 2

- basic theoretical tools of computational linguistics
 - logic: lambda calculus
 - typed feature structures
- Natural Language Analysis
 - problems with context-free grammars
 - fixing the problems
 - fixing the problems caused by the fixes: moving towards mild context-sensitivity with CCGs

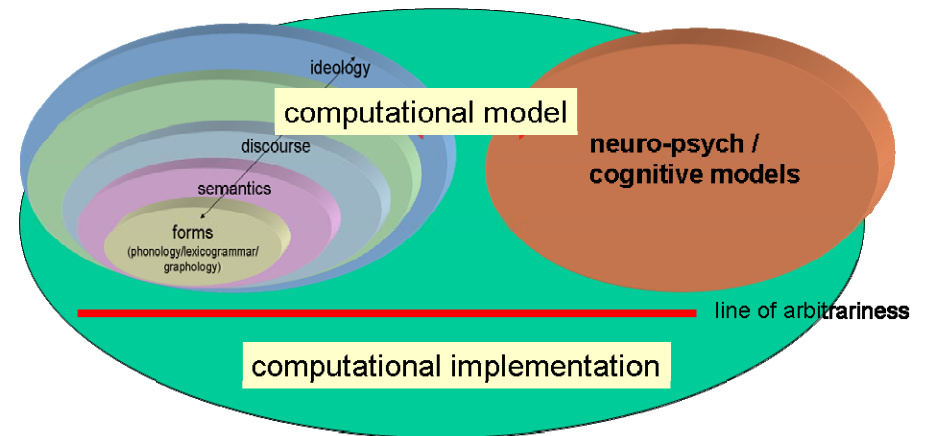
Computational Linguistics



Basic computational modelling tools

- basic computational linguistic tools for representing **semantics**
 - logic
 - event-based semantics
 - ontologies
 - **lambda expressions**
- basic computational linguistic tools for representing **information**
 - **feature structures**
 - **unification**

- generation
- analysis



Refresher : Logical Expressions

chase (x, y) \rightarrow run (x) \wedge run (y)

**Some combination of
predicates and logical
connectors**

Refresher : Logical Expressions

$$\underbrace{\forall x \exists y}_{\text{quantifiers}} \underbrace{\text{chase}(x, y) \rightarrow \text{run}(x) \wedge \text{run}(y)}_{\text{body}}$$

Then you bind the variables with appropriate quantifiers

Refresher : Logical Expressions

The diagram shows the logical formula $\forall x \exists y \text{ chase}(x, y) \rightarrow \text{run}(x) \wedge \text{run}(y)$. The quantifiers $\forall x$ and $\exists y$ are highlighted in green. A bracket under $\forall x \exists y$ indicates they are the quantifiers. A larger bracket under the entire formula indicates its scope. Three curved arrows point from the $\forall x$ and $\exists y$ to the x and y in the $\text{chase}(x, y)$ predicate, showing their binding.

$$\forall x \exists y \text{ chase}(x, y) \rightarrow \text{run}(x) \wedge \text{run}(y)$$

FOPC (non.free)

All the *variables* that occur in the body of the formula, are usually bound with a quantifier at the front!!!

Lambda Expressions

We can turn expressions into functions using lambda expressions

$\lambda x \lambda y$ chase (x, y)

$\left[\lambda x \lambda y \text{ chase (x, y)} \right]$ (farmer, duckling)



chase (farmer, duckling)

Lambda Expressions

We turn expressions into functions by using lambda expressions

$\lambda x \lambda y \text{ chase } (x, y)$

$\left[\lambda x \lambda y \text{ chase } (x, y) \right] (\text{farmer})$



$\lambda y \text{ chase } (\text{farmer}, y)$

Unification

Example: Unification

$$\left(\begin{array}{l} \text{cat: } S \\ \text{subj: } \left(\begin{array}{l} \text{cat: } N \end{array} \right) \\ \text{verb: } \left(\begin{array}{l} \text{cat: } V \end{array} \right) \end{array} \right) \sqsubseteq \left(\begin{array}{l} \text{verb: } \left(\begin{array}{l} \text{cat: } V \\ \text{num: pl} \end{array} \right) \end{array} \right) \\ = \quad ?$$

Example: Unification

$$\left(\begin{array}{l} \text{cat: } S \\ \text{subj: } \left(\begin{array}{l} \text{cat: } N \end{array} \right) \\ \\ \text{verb: } \left(\begin{array}{l} \text{cat: } V \\ \text{num: pl} \end{array} \right) \end{array} \right)$$

Example: Unification

$$\left(\begin{array}{l} \text{cat: } S \\ \text{subj: } \left(\begin{array}{l} \text{cat: } N \\ \text{num: } \#1 \end{array} \right) \\ \text{verb: } \left(\begin{array}{l} \text{cat: } V \\ \text{num: } \#1 \end{array} \right) \end{array} \right) \sqsubseteq \left(\begin{array}{l} \text{verb: } \left(\begin{array}{l} \text{cat: } V \\ \text{num: } \text{pl} \end{array} \right) \end{array} \right) \\ = \quad ?$$

Example: Unification

$$\left[\begin{array}{l} \text{cat: } S \\ \text{subj: } \left[\begin{array}{l} \text{cat: } N \\ \text{num: pl} \end{array} \right] \\ \\ \text{verb: } \left[\begin{array}{l} \text{cat: } V \\ \text{num: pl} \end{array} \right] \end{array} \right]$$

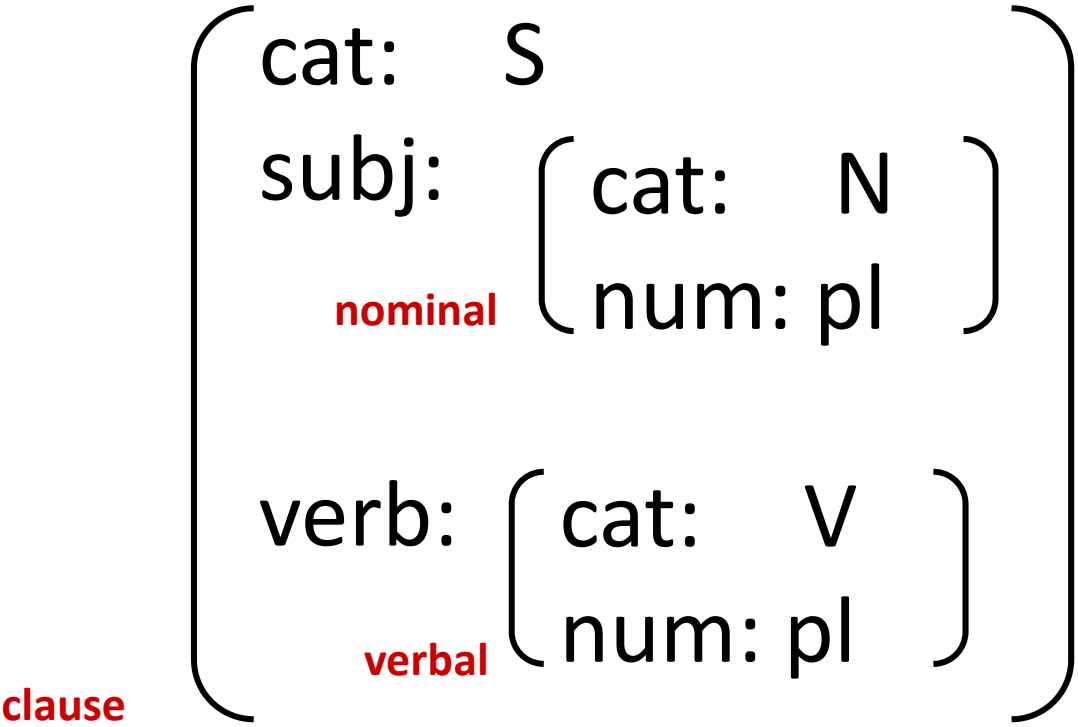
Example: Unification

$$\left(\begin{array}{l} \text{cat: } S \\ \text{subj: } \left(\begin{array}{l} \text{cat: } N \\ \text{num: } \#1 \end{array} \right) \\ \text{verb: } \left(\begin{array}{l} \text{cat: } V \\ \text{num: } \#1 \end{array} \right) \end{array} \right) \sqcup \left(\begin{array}{l} \text{subj: } \left(\begin{array}{l} \text{cat: } N \\ \text{num: } \text{sg} \end{array} \right) \\ \text{verb: } \left(\begin{array}{l} \text{cat: } V \\ \text{num: } \text{pl} \end{array} \right) \end{array} \right) = ?$$

Typed Unification

- Nowadays grammars usually work with **typed unification**
 - distinct information types are defined as having particular combinations of attributes
 - types are organised into inheritance lattices

Example: Typed Unification



Example: Typed Unification

$$\begin{array}{l} \text{clause} \left[\begin{array}{l} \text{cat: S} \\ \text{subj: } \left[\begin{array}{l} \text{cat: N} \\ \text{num: pl} \end{array} \right] \\ \text{verb: } \left[\begin{array}{l} \text{cat: V} \\ \text{num: pl} \end{array} \right] \end{array} \right] \\ \text{nominal} \left[\begin{array}{l} \text{num: sg} \end{array} \right] = ? \end{array}$$

Example: Typed Unification

$$\left(\begin{array}{l} \text{cat: } S \\ \text{subj: } \left(\begin{array}{l} \text{cat: } N \\ \text{num: pl} \end{array} \right) \\ \\ \text{verb: } \left(\begin{array}{l} \text{cat: } V \\ \text{num: pl} \end{array} \right) \end{array} \right)$$

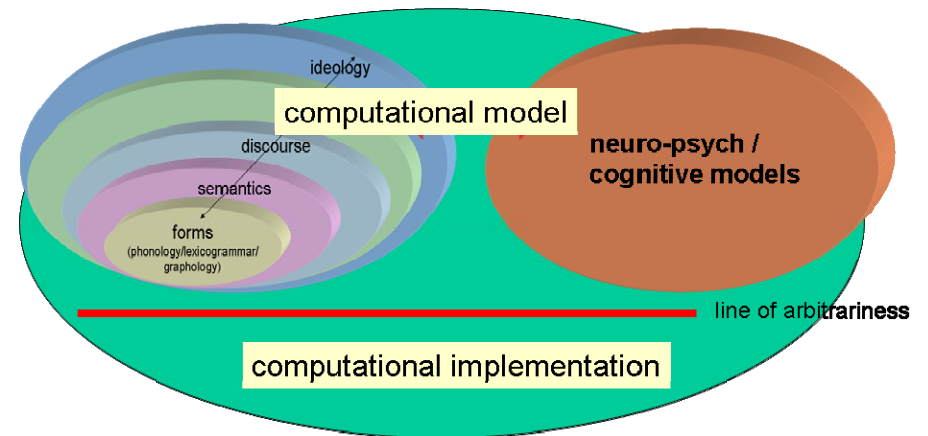
$$\sqcup \left(\begin{array}{l} \text{num: sg} \end{array} \right) = ?$$

Feature Geometries

- Much of modern linguistics is now expressed as **bundles of features**
- and how these are **distributed around** syntactic structures
- Some special kinds of features flow along the `backbone' provided by the tree structures:
head features

Basic computational modelling tools

- basic computational linguistic tools for representing **information**
 - feature structures ✓
 - unification ✓
- basic computational linguistic tools for representing **semantics**
 - event-based semantics
 - ontologies
 - description logics
 - lambda expressions ✓



Grammar and Feature Unification

Problems with CF Phrase Structure Grammars

- Difficult to capture **dependencies** between constituents
 - the boy runs
 - the boys run

 - * the boy run
 - * the boys runs

Problems with CF Phrase Structure Grammars

- Difficult to capture **dependencies** between constituents
 - the boy opens the door
 - *?? the boy opens

 - the boy sits
 - *?? the boy sits the table

CF Solution

- exploding the number of rules is one way to provide a solution...

S → NP_{sing} VP_{sing}
S → NP_{plural} VP_{plural}
NP → NP_{sing}
NP → NP_{plural}
NP_{sing} → Det_{sing} N_{sing}
NP_{plural} → Det_{plural} N_{plural}

VP_{sing} → V_{sing}
VP_{plural} → V_{plural}

VP_{sing} → V_{tr}_{sing} NP
VP_{plural} → V_{tr}_{plural} NP

grammar

Det_{sing} → {a, this, the}
Det_{plural} → {some, these, the}

N_{sing} → {boy, girl, ...}
N_{plural} → {boys, girls, ...}

V_{sing} → {sits, ...}
V_{plural} → {sit, ...}

V_{tr}_{sing} → {opens, ...}
V_{tr}_{plural} → {open, ...}

lexicon

A better solution...

- What we really want to say is that some constituents *share* properties

The boy runs

the 'Subject' and the 'Verb' **agree** in number

- i.e., they share the same value for their **number feature**

Phrase structure rules with features

S → NP VP
+singular +singular

VP → V (NP)
+singular +singular


Phrase structure rules with features

S → NP VP
+plural +plural


VP → V (NP)
+plural +plural

Phrase structure rules with features

S → NP VP
[number: sing] [number: sing]




VP → V (NP)
[number: sing] [number: sing]




Phrase structure rules with features

S → NP VP
[number: pl] [number: pl]



VP → V (NP)
[number: pl] [number: pl]



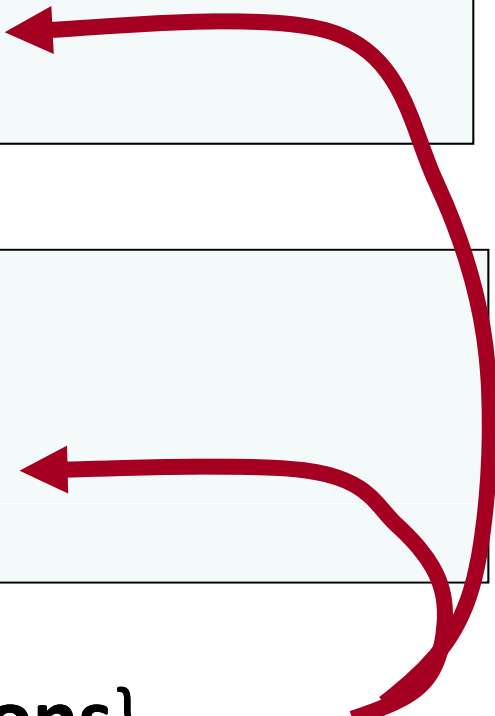
Generalised Rules

S → NP VP
[number: **x**] [number: **x**]

VP → V (NP)
[number: **x**] [number: **x**]

Generalised Rules (PATR formalism)

$S \rightarrow NP \quad VP$
 $\langle NP \text{ number} \rangle = \langle VP \text{ number} \rangle$



$VP \rightarrow V \quad (NP)$
 $\langle VP \text{ number} \rangle = \langle V \text{ number} \rangle$

Grammar = {PS-rules + **path equations**}

Features → Feature structures

Attribute-value matrices (AVMs)

+singular

[number: singular]

+ing-form

[verb: ing-form]

+masc

+sing

(gender: masc
number: sing)

Compatibility

Information

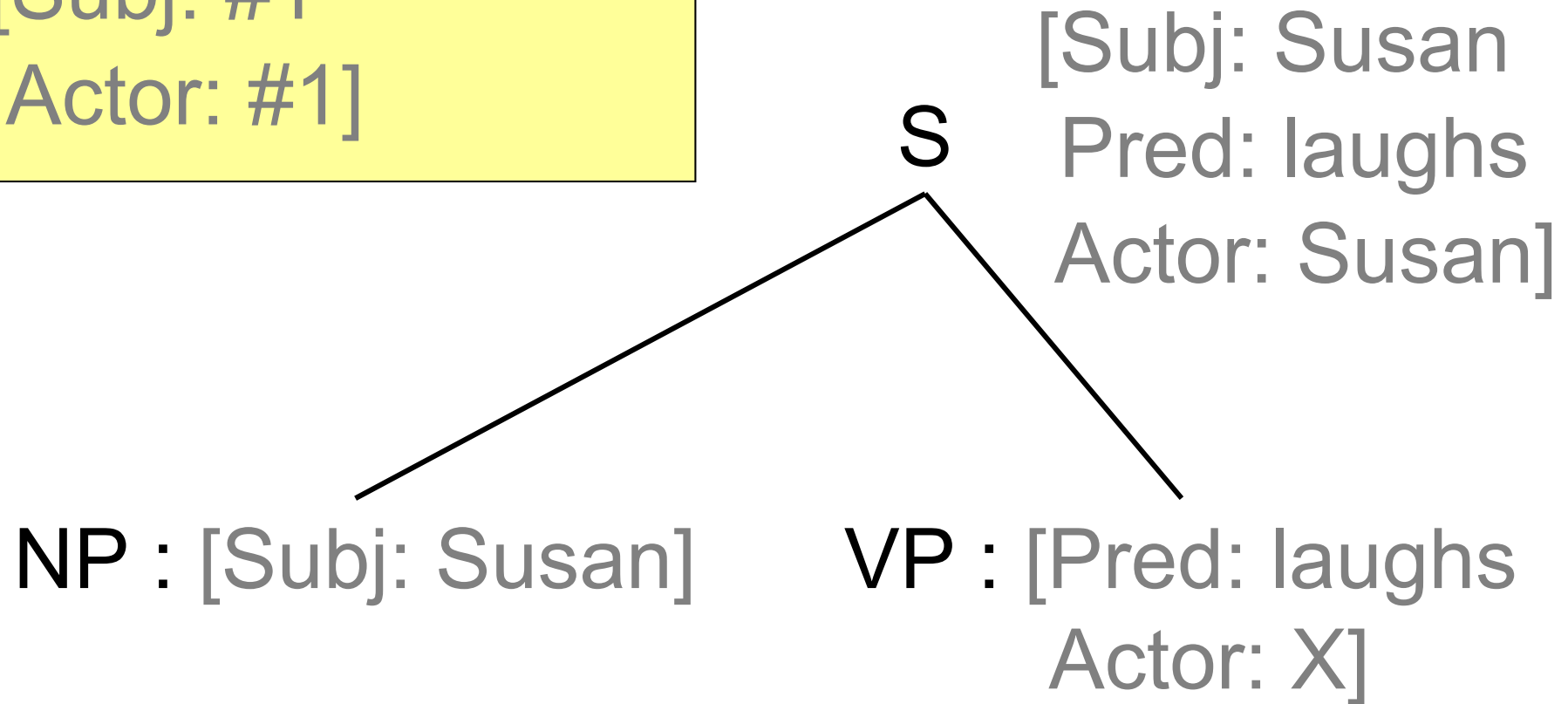
Feature structures

Unification

grammar, phonology, ...

Susan laughs

S → **NP** **VP**
[Subj: #1
Actor: #1]

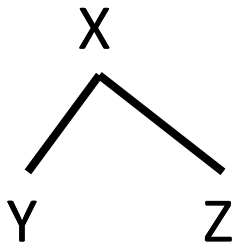


Feature Representation

- Syntactic tree becomes a more complex structure
- Each node in the tree is in fact a bundle of features
- Particular rules (specified in the grammar) specify what conditions hold on the feature structures
- Usually: local – i.e., conditions hold over a dominating node and its children

Final move...

- *All* information is moved into the feature structure – even the tree structure...
 - HPSG
(Head-driven Phrase Structure Grammar)



[Head: X
Daughters: <Y Z ...>
Cat: C]

Summary: Recap

- This is really the current state of the art in computational linguistics: all linguistic information is represented as feature bundles
- This is called a 'information-based' paradigm

Example Unification Grammars

PATR-II : WinPatr System ...

Example Grammar

Parameter start symbol is S

Rule {Satz}
S -> NP VP.

Rule {NP-Name}
NP -> Name.

Rule {NP}
NP -> Det N.

Rule {VP intransitiv}
VP -> Vi.

Rule {VP transitiv}
VP -> Vt NP.

Rule {VP transitiv mit PP}
VP -> Vt NP PP.

Rule {VP ditransitiv}
VP -> Vt2 NP NP.

Rule {VP mit PP-Objekt}
VP -> Vpo PP.

Rule {einfache PP}
PP -> P NP.

PATR-formalism

Example Lexicon

\w cried
\c Vi

\w girl
\c N

\w the
\c Det

\w saw
\c Vt

\w student
\c N

\w a
\c Det

\w gave
\c Vt2

\w book
\c N

\w in
\c P

\w on
\c P

PATR-formalism



f03b.grm
f02.lex

Example Lexicon (with features)

\w cried
\c Vi

\w saw
\c Vt

\w gave
\c Vt2

\w sings
\c Vi
\f 3sg

\w girl
\c N
\f sg

\w student
\c N

\f sg

\w book
\c N
\f sg

\w girls
\c N
\f pl

\w students
\c N
\f pl

\w the
\c Det

\w a
\c Det

\w in
\c P

\w on
\c P

Example Lexicon (with features)

\w cried
\c Vi

\w girl
\c N
lf sg

\w the
\c Det

\w saw
\c Vt

\w student
\c N

\w a
\c Det

\w gave
\c Vt2

lf sg

\w in
\c P

[num: sg].

Let sg be [nu]

\w sings
\c Vi
lf 3sg

\w book
\c N
lf sg

\w on
\c P

[num: sg
pers: 3].

Let 3sg be sg [p

\w girls
\c N
lf pl

\w students
\c N
lf pl

Example Lexicon (with features)

\w cried
\c Vi

\w girl
\c N
lf sg

\w the
\c Det

\w saw
\c Vt

\w student
\c N

\w a
\c Det

[lex: student]
[cat: N].

\w gave
\c Vt2

lf sg

\w in
\c P

\w sings
\c Vi
lf 3sg

\w book
\c N
lf sg

\w on
\c P

[lex: sings]
[cat: V].

\w girls
\c N
lf pl

\w students
\c N
lf pl

Head features are usually
'passed up' to the dominating node

S → NP VP

<NP head num> = <VP head num>

<NP head pers> = <VP head pers>

Head features are usually 'passed up' to the dominating node

Rule {VP intransitiv}
 $VP \rightarrow V:$
<VP head> = <V>
<V subcat> = i.

$\backslash w$ sings \longrightarrow [lex: sings]
 $\backslash c$ Vi [cat: V].
 $\backslash f$ 3sg

Let V be [cat: V].
Let Vi be V [subcat: i].

[cat: V
subcat: i]

exercise with the example grammar...

what structure (both tree structure and feature structure) does the grammar produce for:

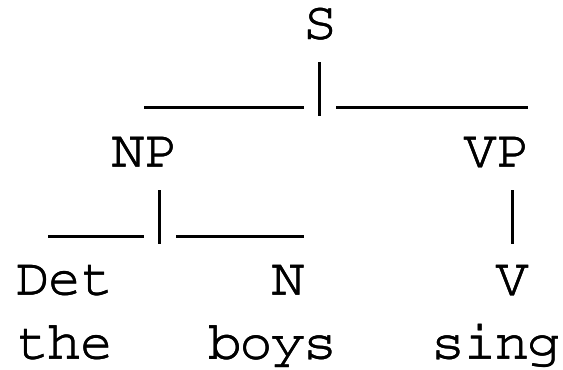
“the boys sing”

and what would it do for

“the boy sing”

Feature-based Parsing: “the boys sing”

1:



```
S:
[ cat: S
  subj: [ cat: NP
          spec: [ cat: Det
                  lex: the
                  num: pl ]
          head: [ cat: N
                  lex: boys
                  num: pl
                  pers: 3 ] ]
  pred: [ cat: VP
          head: [ cat: V
                  subcat:i
                  lex: sing
                  num: pl
                  pers: 3 ] ] ] ]
```

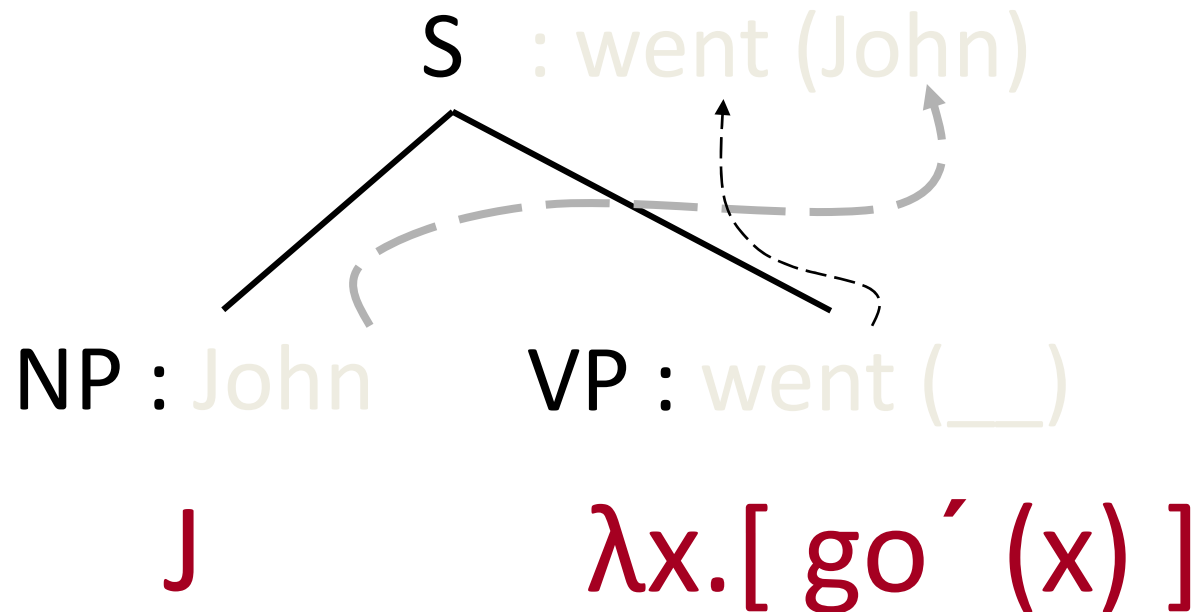
1 parse found

Problem of surface interpretation

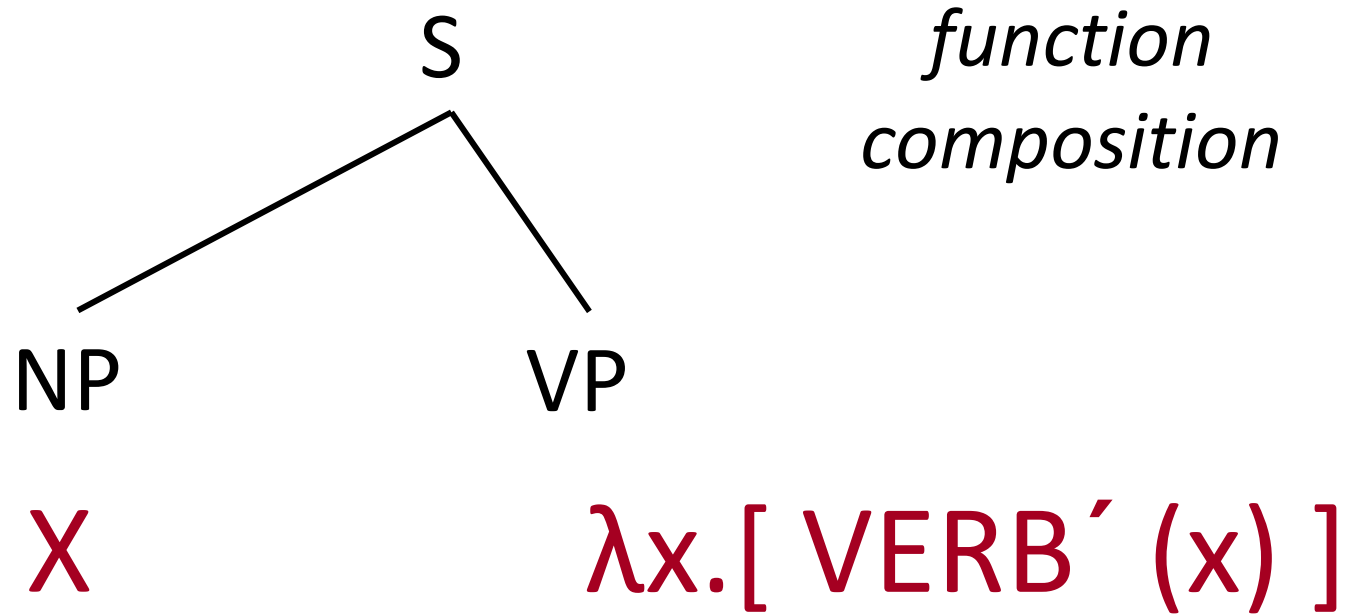
- If we interpret a syntactic tree as a set of instructions for forming a semantic interpretation, then we do not need to rely on a 'deep' structure for meaning...

How to interpret the 'surface' structure?:
Compositional Semantics as Function Application

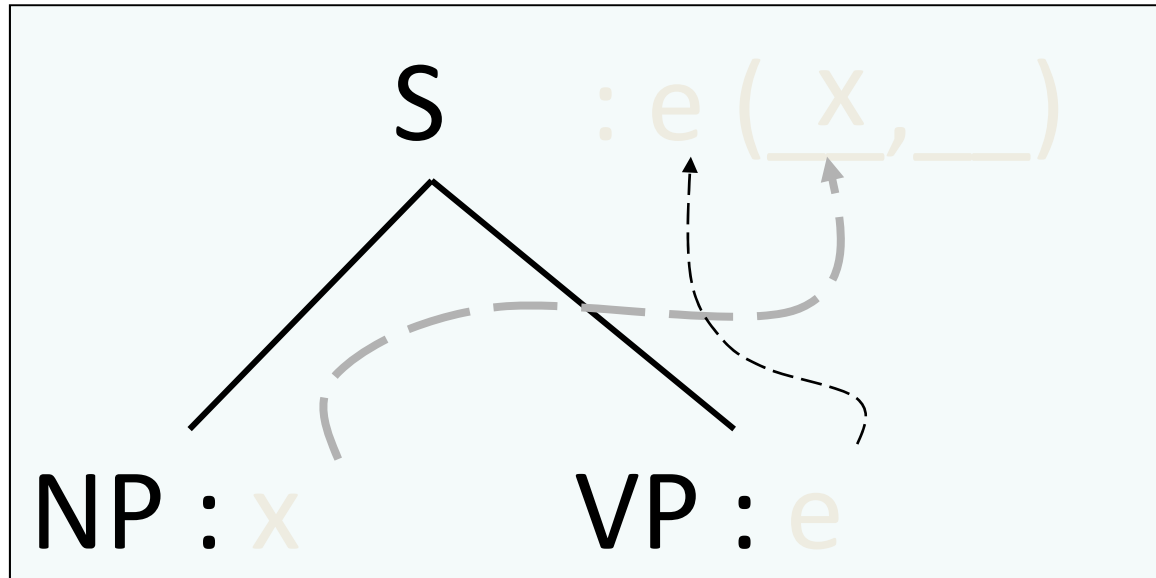
“John went”



Semantic Interpretation Rule

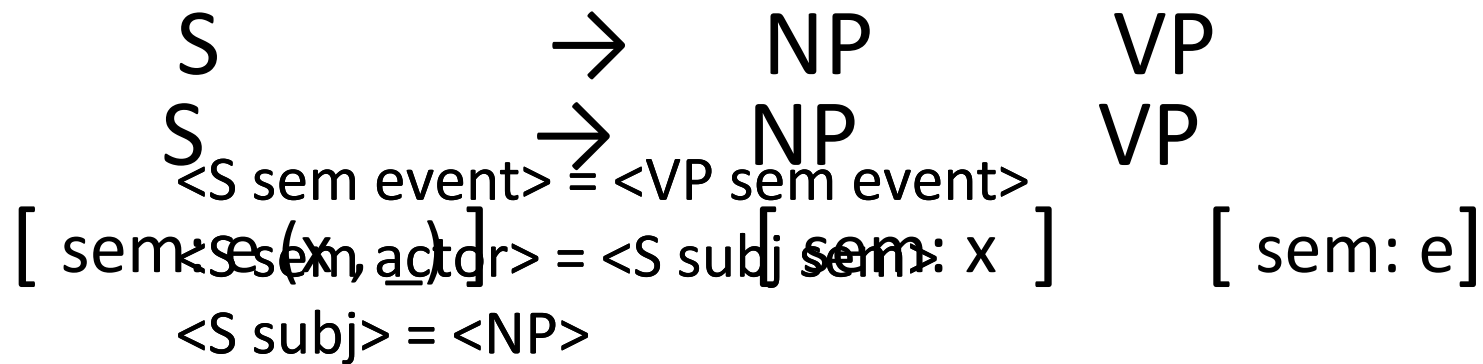


Semantic composition represented i.t.o. Semantic features



S \rightarrow **NP** **VP**
[sem: e (x , _)] [sem: x] [sem: e]

Semantic features



Semantic features

- *as informal feature values*

S → NP VP
[sem: e (x , _)] [sem: x] [sem: e]

- *as path equations*

S → NP VP

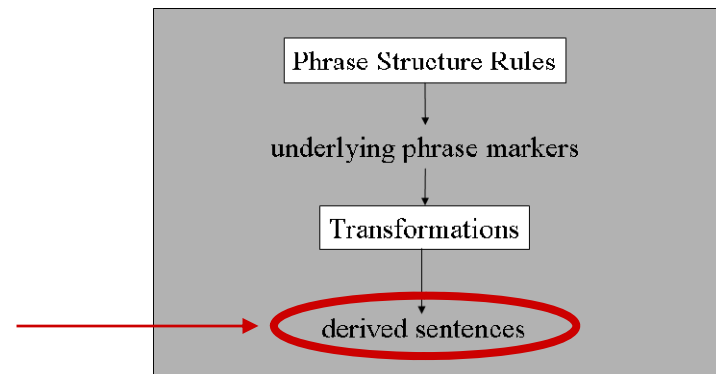
<S sem event> = <VP sem event>

<S sem actor> = <S subj sem>

<S subj> = <NP>

Surface Structure

- If we had a surface structure, we could interpret it...
- But we don't...
- How can we produce the structures that natural human languages use without invoking far too powerful mechanisms?



Remaining problems...

- Transformations are in general Turing equivalent and hence unlikely for a language model
- Seems implausible that our language facility evolved as a mechanism with that kind of power
- Unification is also potentially a **very** expensive operation
- Need strong constraint for just which structures are to be unified with others
- **One way of achieving this is by making the syntactic backbone do more work...**

A different style of grammar

- Combinatory Categorical Grammar (CCG)
 - generalisation of Categorical Grammars
 - Ajdukiewicz (1935)

“Die syntaktische Konnexität”. *Studia Philosophica*, 1:1-27.
 - Bar-Hillel (1953)

“A quasi-arithmetic notation for syntactic description”.
Language 29:47-58.

promoted particularly for computational linguistics by Mark Steedman

Combinatory Categorical Grammar

CCG

- In a CCG all lexical items are given categories that state their potential for combination with other lexical (and non-lexical) units
- There are only general rules on how to combine categories, no particular 'grammar' rules.

Analysis with CCG

Example: “leave the room”

collect the lexical items and their categories (given in the lexicon):

leave :- s/np

the :- np/n

room :- n

Analysis with CCG

Example: “leave the room”

leave :- s/np

 the :- np/n

room :- n

‘the’ is a unit looking for an ‘n’
on its right to combine with to
give an ‘np’

Analysis with CCG

leave :- s/np

the :- np/**n**

room :- **n**

Analysis with CCG

leave :- s/np

the :- np/n

room :- n



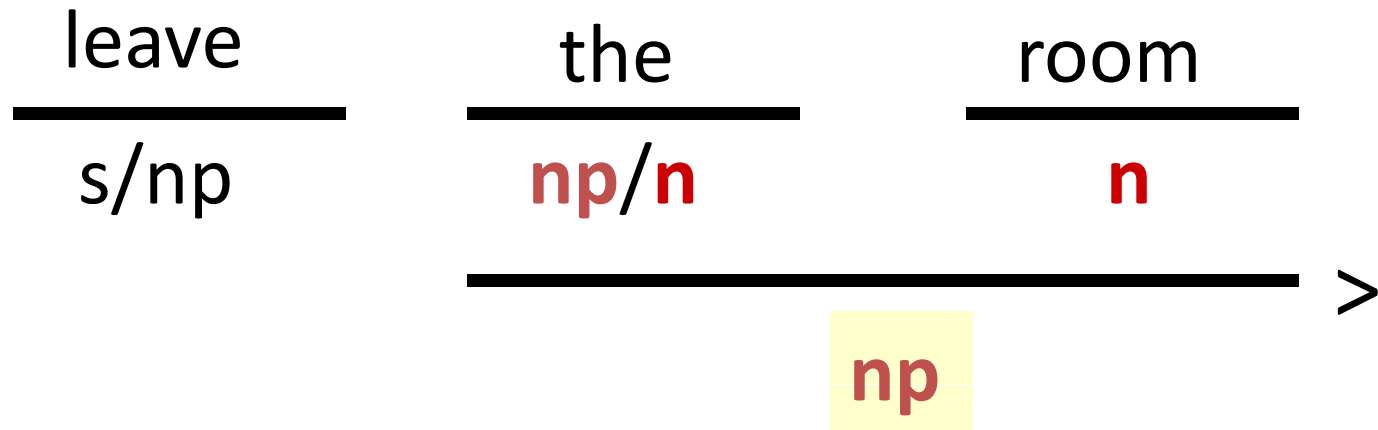
the room :- np

“forward application”



- (>) the room :- np

Analysis with CCG



Analysis with CCG

leave :- s/**np**

the room :- **np**

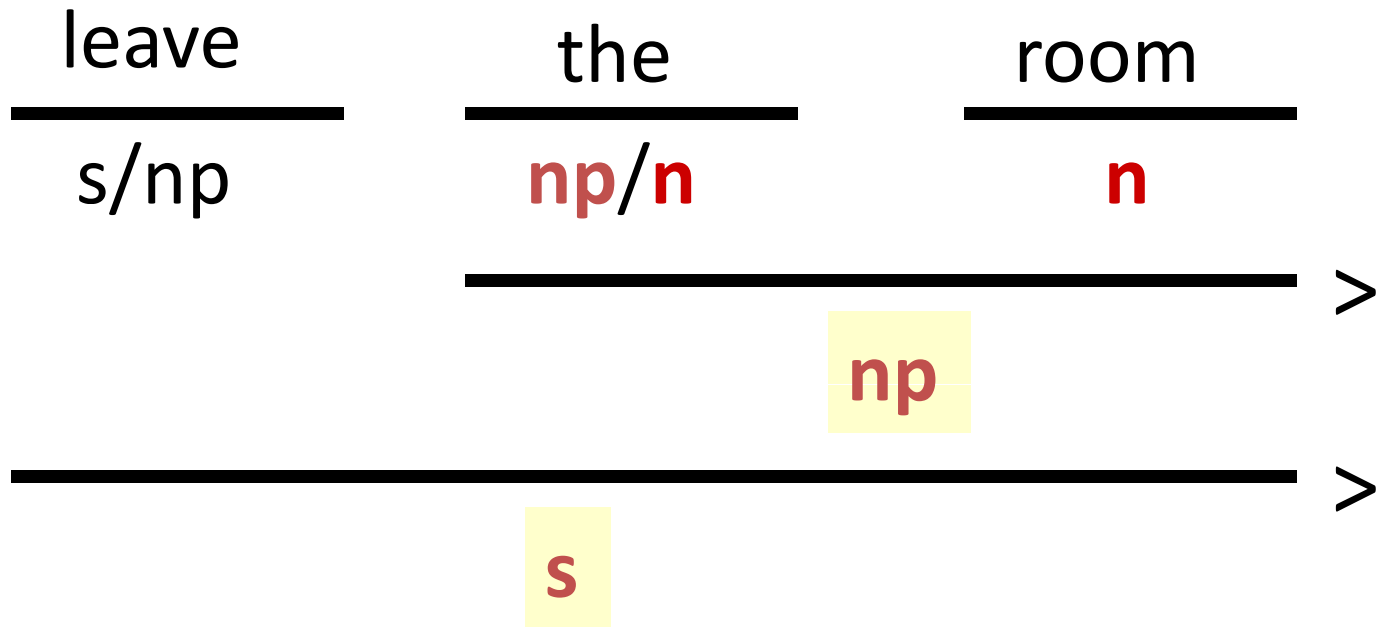
Analysis with CCG

leave :- s/np

the room :- np

- leave the room :- s

Analysis with CCG



Most basic combination rules

- *Functional Application*

$$-X/Y \ Y \Rightarrow \ X$$

$$-Y \ X \backslash Y \Rightarrow \ X$$

forward application (>)

backward application (<)

CCG: analysis

“Keats eats apples”

Keats: N

apples: N

eats: $(S \setminus N) / N$

CCG: analysis

Keats: NP
apples: NP

eats:(S\NP)/NP

Keats eats apples

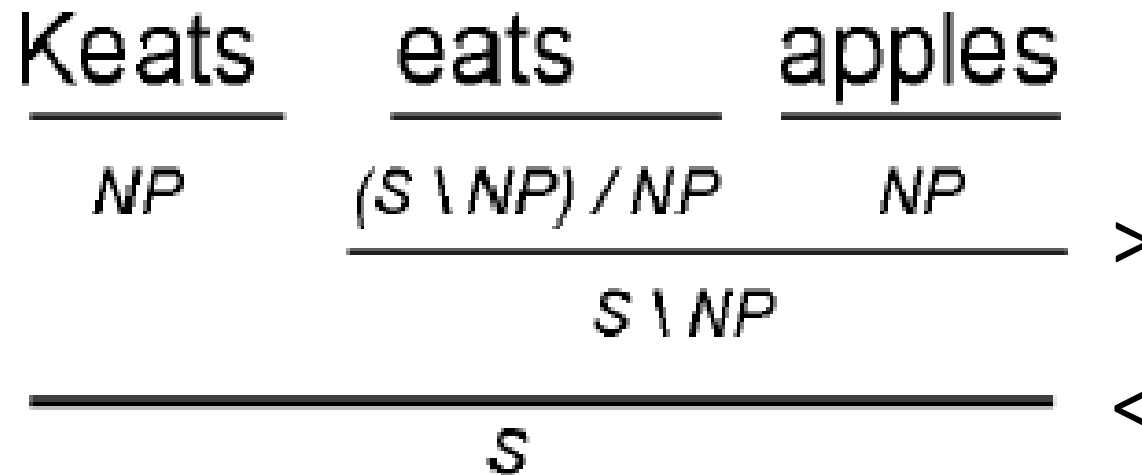
>

<

CCG: analysis

Keats: NP
apples: NP

eats:(S\NP)/NP



More combination rules

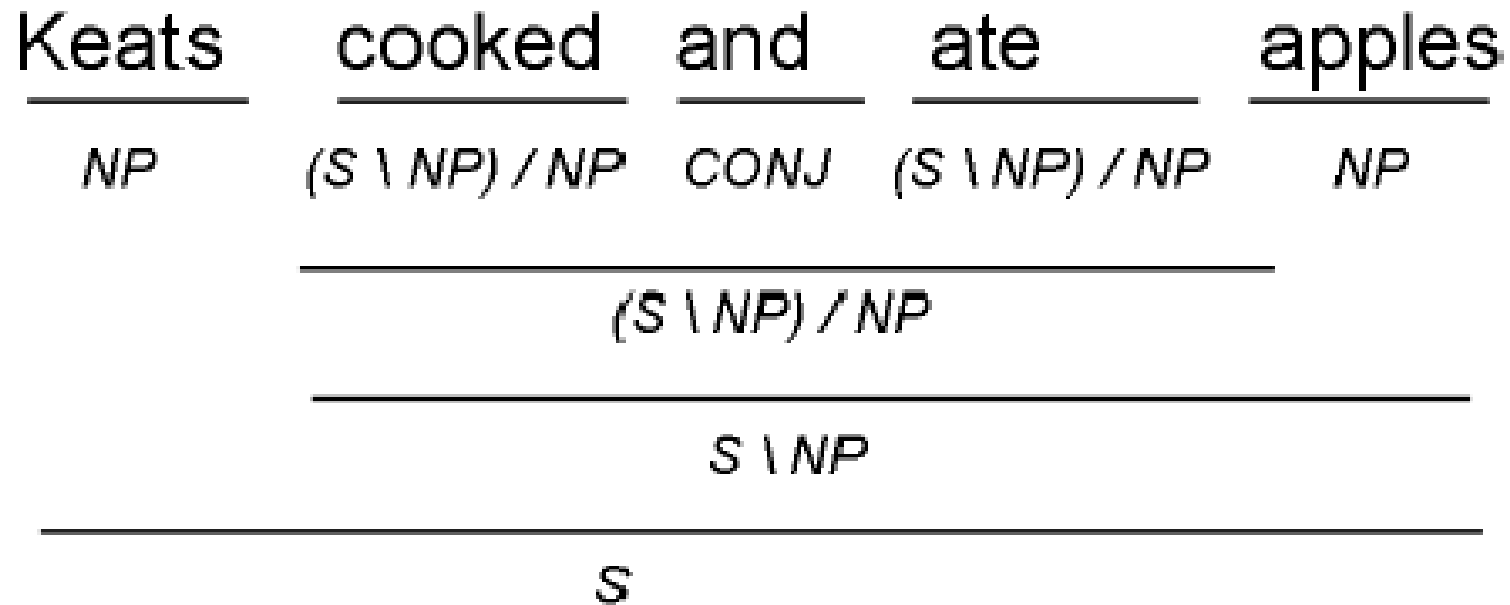
- *Coordination*

$$-X \text{ CONJ } X \Rightarrow X$$

CCG: analysis

Keats cooked and ate apples

CCG: analysis



Important features of CCG

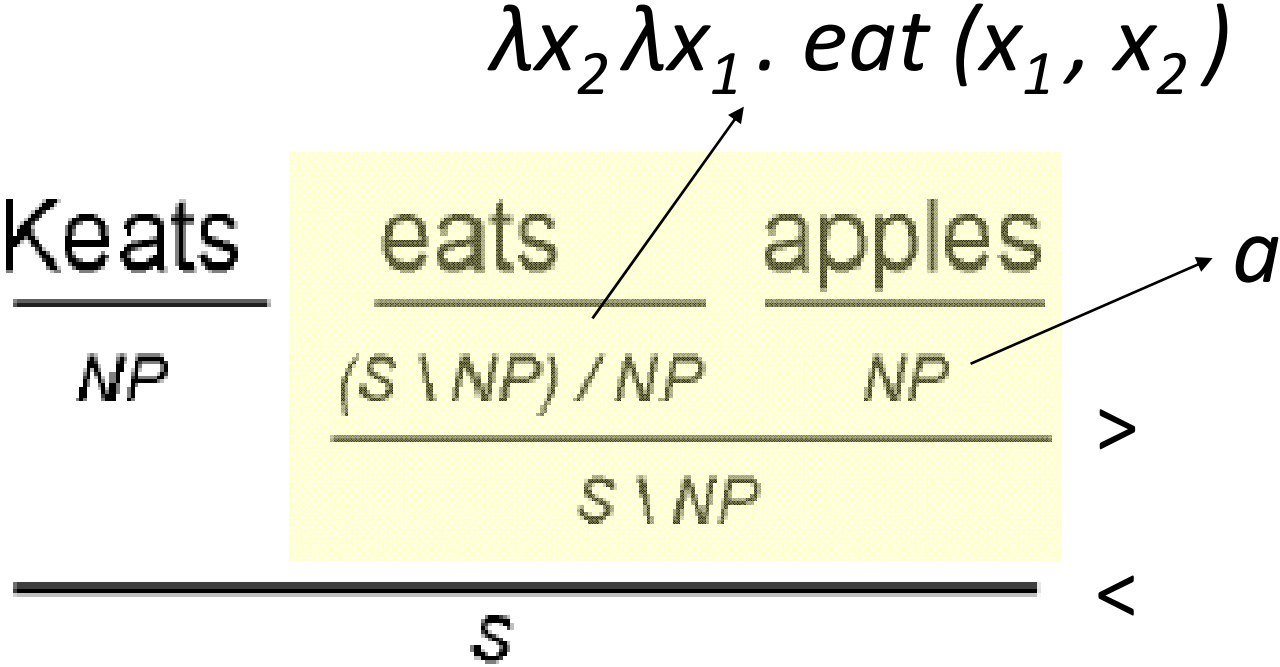
- Highly lexicalized grammars
 - Only lexical items in CCG
 - Analysis is strongly restricted
- Construct syntactic and semantic representations *synchronously*

Semantics (traditional)

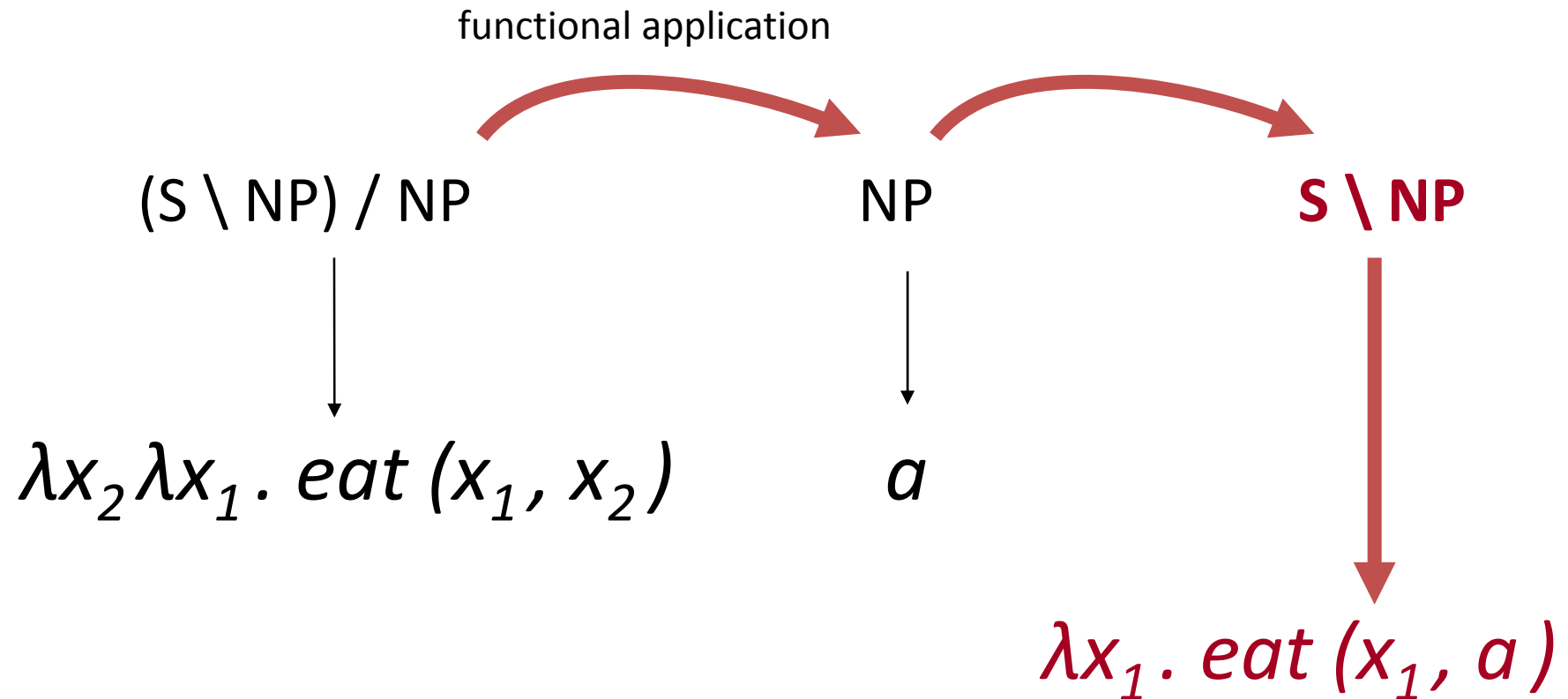
- $eat := (s / np_{nom}) \setminus np_{acc}$
- $\lambda x_2 \lambda x_1 . eat(x_1, x_2)$

CCG: analysis with semantics

Keats: NP
 apples: NP
 eats:(S\NP)/NP



CCG: analysis with semantics



An interesting point

- The syntactic structure is not itself important
- It is just a record of the instructions that were carried out to build the correct semantics

Important features of CCG

- Highly lexicalized grammars
 - Only lexical items in CCG
 - Analysis is strongly restricted
- Construct syntactic and semantic representations *synchronously*
- Highly constrained computational complexity

Complexity classes

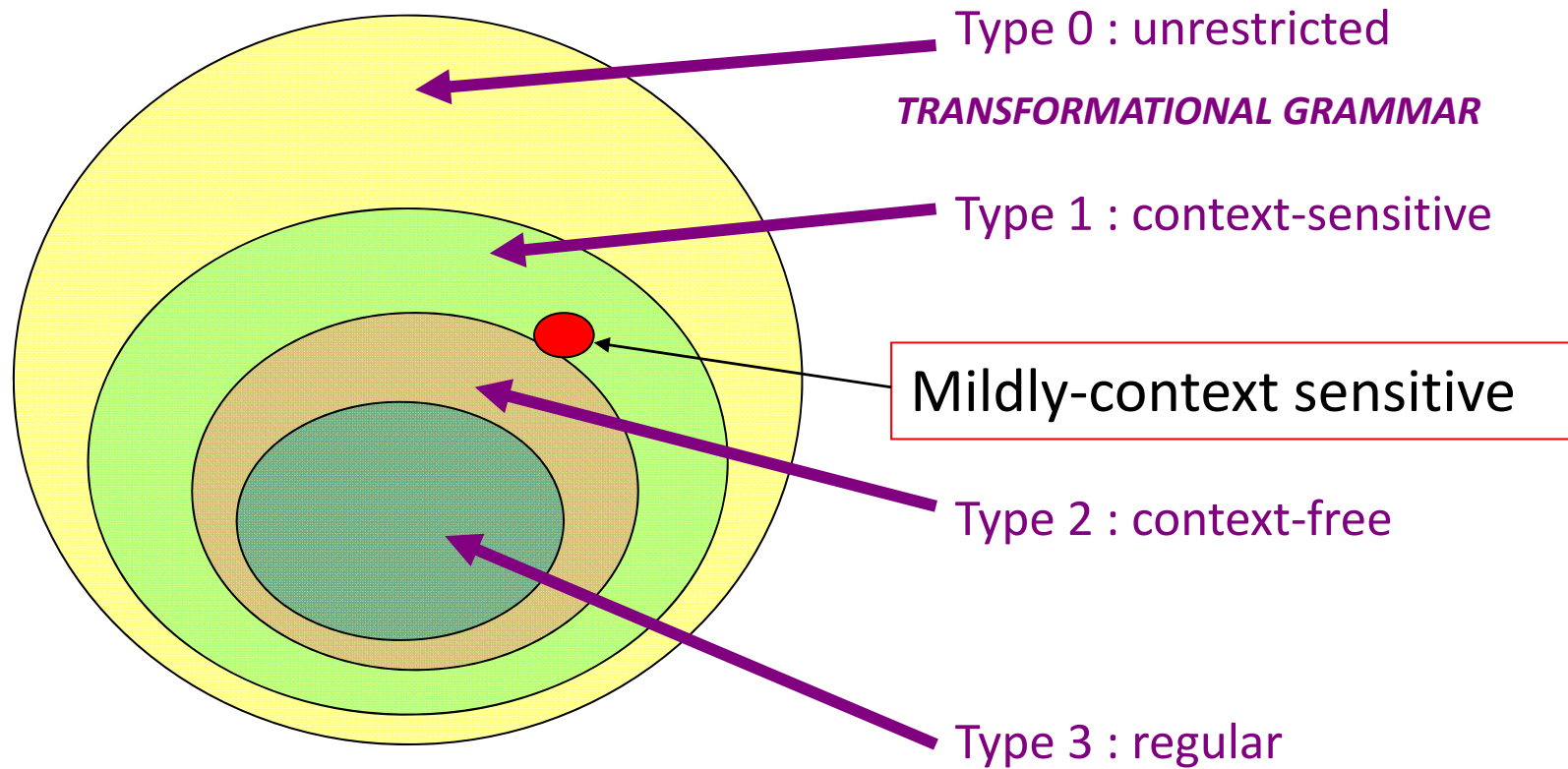
- **Subclasses of polynomial time**
 - Linear time: $O(n)$
 - Quadratic time: $O(n^2)$
 - Cubic time: $O(n^3)$
- **Exponential time: $O(k^n)$**
- **Non-deterministic polynomial time : NP**

CCG: $O(n^6)$

n: length of input

k: difficulty of problem

'Chomsky' Hierarchy



Features of CCG: summary

- CCG is one of the most quickly developing grammatical formalisms for linguistic processing developing at this time
- Its flexibility appears well matched to real language use, including sentence fragments and difficult combinations
- Its computational complexity is low enough for real processing
- Semantic interpretation is provided at the same time as syntactic analysis.

Computational Use of CCG

- OpenCCG
 - White, Baldridge, Kruijff, ...
 - freely accessible
- Grammars now using a particular kind of semantics:
 - Hybrid-Logic Dependency Semantics

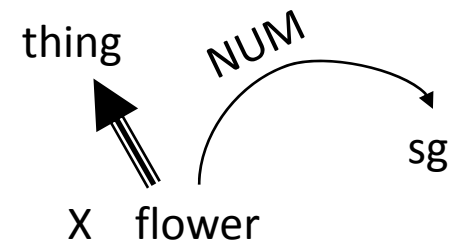
Hybrid-Logic Dependency Semantics

$flower \vdash n\langle 2 \rangle sg, X:thing : @X:thing (flower \wedge \langle NUM \rangle sg)$

↑
complex category
definition

↑
HLDS expression

*combining categories, feature
structures, and semantic
constraints*



Similarity between HLDS and feature structures

Baldrige / Kruijff

(8) $\langle \text{SUBJ} \rangle (i \wedge \langle \text{AGR} \rangle \textit{singular} \wedge \langle \text{PRED} \rangle \textit{dog})$
 $\wedge \langle \text{COMP} \rangle \langle \text{SUBJ} \rangle i$

(9)
$$\left[\begin{array}{l} \text{SUBJ} \quad \boxed{1} \quad \left[\begin{array}{l} \text{AGR singular} \\ \text{PRED dog} \end{array} \right] \\ \text{COMP} \quad \left[\begin{array}{l} \text{SUBJ} \quad \boxed{1} \end{array} \right] \end{array} \right]$$

Basic computational modelling tools

- basic computational linguistic tools for representing **semantics**
 - logic
 - event-based semantics
 - ontologies
 - **lambda expressions** ✓
- basic computational linguistic tools for representing **information**
 - **feature structures** ✓
 - **unification** ✓

- generation
- analysis

