University of Auckland

07.363

Functional and Logic Programming

Christian Collberg
October 7, 1994

Natural Language I

Computational Linguistics I

- Text Analysis
  - Attribution of authorship: "Did the same person write *The Merchants of Venice* and *The Taming of the Shrew*?"
- Machine Translation
  - Translate between two natural languages.
  - Canada: English ⇒ French weather forecasts: 75% OK.
  - Japan TeleCom (?), project: simultaneous spoken Japanese ⇔ English translation.
- Information Extraction
  - Scan articles for information about some particular subject.

Slide 14–1

Computational Linguistics II

- Man-Machine Interaction
  - Question-answering systems.
  - Explanation systems.
- Speech Processing
  - Speech understanding.
  - Speech generation.
  - Computer dictation systems.
- Teaching
  - Computer-Aided instruction
  - Speech Therapy
  - Foreign language teaching

Slide 14–2

Some NLP TLAs

CL Computational Linguistics

NLP Natural Language Processing

NLU Natural Language Understanding

NLG Natural Language Generation

NLI Natural Language Interfaces

MT Machine Translation

DCG Definite Clause Grammar

Slide 14–3

Definitions

phoneme

- A language's *sound categories* ($\approx 20 - -50$)
- English: /ʃ/ (bu*sh*), /ŋ/ (pi*ng*).

morpheme

- A language's *meaning elements*
- English: in-de-ciper-able; print-s, print-ed, print-ing, im-print.
- Swahili: wa-me-tu-lipa (they-have-us-paid)

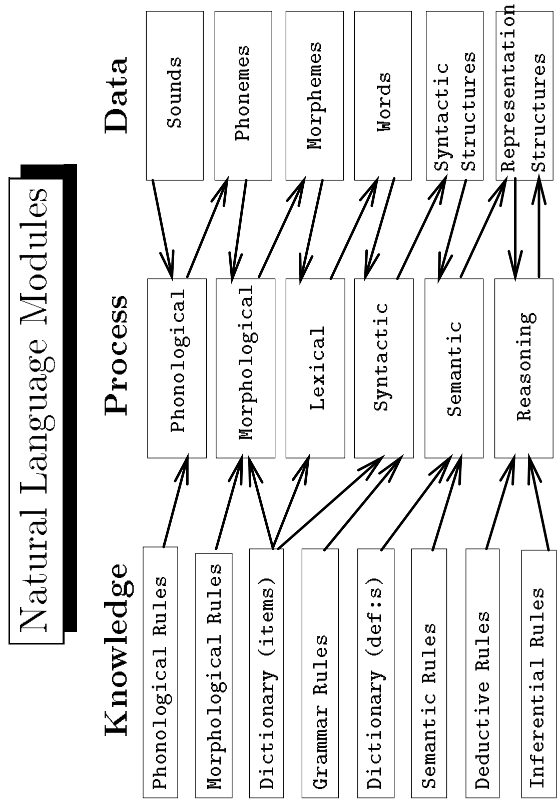
syllable

- Vowel preceeded/followed by consonants.
- English(≈ 60): ham/let, op/er/a.
- Yiddish: /k v/ (*kvetch*).
- Bantu: /m b/ (*Mbutu*).

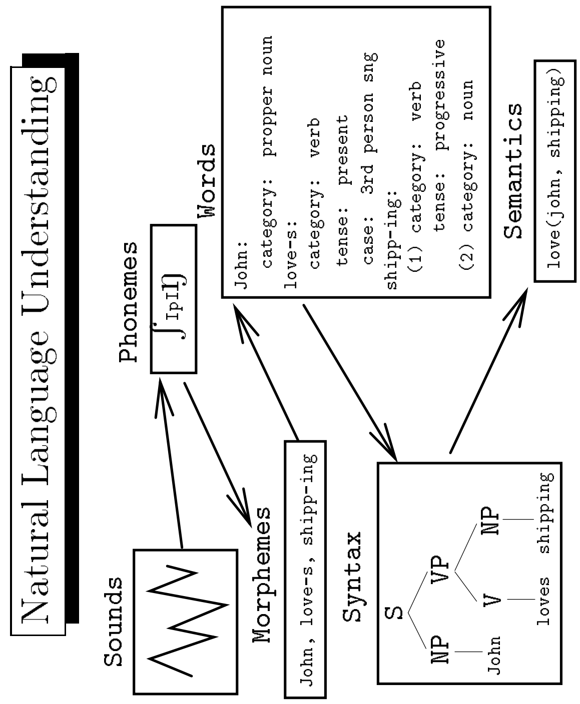
syntax word order, sentence construction

semantics meaning of sentences

Slide 14–4



Slide 14–5



Slide 14–6

Syntactic Categories

S [Sentence]

- ``J. likes S.'s black hair''

N [Noun]

- ``John'', ``hair''
- The _ is/are good.

V [Verb]

- ``eating'', ``sat''
- You _ well.

Adj [Adjective]

- ``black'', ``long''
- The _ one is.

Det [Determiner]

- ``the'', ``a'', ``every''

NP [Noun Phrase]

- ``S.'s long black hair''

VP [Verb Phrase]

- ``eating apples''

Slide 14–7

The Lexicon

```
Word [the]:
   <cat>     = Det
Word [book]:
   <cat>     = N
Word [little]:
   <cat>     = A

Word [died]:
   <cat>     = V
   <tense>   = past
   <arg1>    = ∅

Word [paid]:
   <cat>     = V
   <tense>   = past
   <arg1>    = NP
```

Slide 14–8

Natural Language Syntax I

- Syntax = word order
- **S**=Subject, **V**=Verb, **O**=Object
- Different languages have different basic word order:

  SVO English

  V_1 (verb-first, VSO or VOS) Irish, Māori

  V_2 (verb-second, SVO or OVS) German, Swedish

  SOV Japanese, Turkish

- Languages have *fixed* or *variable* word order:

  fixed Use word order

  variable Use inflection (word-suffixes)

  to indicate *constituents.*

Slide 14–9

Natural Language Syntax II

English:

Johnny	*kissed*	*Lisa*
S	**V**	**O**

The teacher	*gave*	*a book*	*to*
S	**V**	**DO**	

the student
IO

German:

Ich	*mag*	*das Buch.*
S	**V**	**O**
I	like	the book.

Das Buch	*mag*	*ich.*
O	**V**	**S**
the book	like	I

'I like the book'

Slide 14–10

Natural Language Syntax III

Japanese:

Taroo-ga	*ringo-o*	*tabeta*
S	**O**	**V**
Taro	apple	ate

'Taro ate an apple'

Sensei-ga	*seito-ni*	*hon-o*	*ageta*
S	**IO**	**DO**	**V**
Teacher	student	book	give

'The teacher gave a book to the student'

Maori:

I	*kai*	*a Anaru*	*I te aaporo*
	V	**S**	**O**
past	eat	Andrew	the apple

'Andrew ate the apple'

I	*whakakii*	*raatou*	*i te taraka*	*ki*	*te p.*
	fill	they	truck	with	the p.
	V	**S**	**DO**		**IO**

'They filled the truck with petrol'

Slide 14–11

Phrase Structure Rules

LHS → *RHS*

- LHS ("the left hand side") can be composed of RHS ("the right hand side")

$S \rightarrow NP\ VP$

- A sentence (S) can be composed of a noun phrase (NP) followed by a verb phrase (VP)

$NP \rightarrow Det\ A\ N$

- A noun phrase can be composed of a determiner (*a, the*) followed by an adjective (*pretty, blue*), followed by a noun.

Slide 14–12

Grammar G_1 I

- Context-Free Phrase Structure Grammar (CF-PSG)

$S \rightarrow NP\ VP$	`% Simple sentence`
$VP \rightarrow V\ NP$	`% Transitive verb`
$VP \rightarrow V$	`% Intransitive verb`
$NP \rightarrow N$	`% Noun phrase`
$N \rightarrow John$	
$N \rightarrow Lisa$	
$N \rightarrow house$	
$V \rightarrow died$	
$V \rightarrow kissed$	

Slide 14–13

Uses of CF-PSG Grammars

1. Define grammatical/ungrammatical sentences of the language.
2. Generate grammatical sentences.
   (a) Sentence := start symbol (**S**).
   (b) Select a rule $X \rightarrow Y\ Z$.
   (c) Replace an occurence of **X** in the sentence with **Y Z**. Repeat from (b).
   (d) Stop when the sentence = just words.
3. Test if a sentence is grammatical (parsing).
   (a) Select a rule $X \rightarrow Y\ Z$.
   (b) Replace an occurence of **Y Z** in the sentence with **X**. Repeat from (a).
   (c) Stop when sentence = start symbol.
4. Produce the syntactic *structure(s)* of a sentence.
   - Usually a tree.

Slide 14–14

Grammar G_1 — Generation — III

`S ⇒ NP VP`	`% Rule:`	$S \rightarrow NP\ VP$
`⇒ N VP`	`% Rule:`	$NP \rightarrow N$
`⇒ John VP`	`% Rule:`	$N \rightarrow John$
`⇒ John V NP`	`% Rule:`	$VP \rightarrow V\ NP$
`⇒ John kissed NP`	`% Rule:`	$V \rightarrow kissed$
`⇒ John kissed N`	`% Rule:`	$NP \rightarrow N$
`⇒ John kissed Lisa`	`% Rule:`	$N \rightarrow Lisa$

Slide 14–15

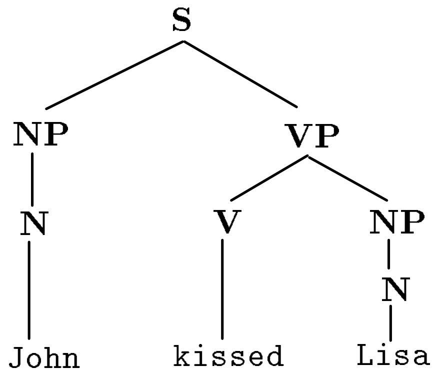
Grammar G_1 — Recognition (Parsing) — IV

```
John kissed Lisa
⇑ N kissed Lisa      Rule: N → John
⇑ NP kissed Lisa     Rule: NP → N
⇑ NP V Lisa          Rule: V → Kissed
⇑ NP V N             Rule: N → Lisa
⇑ NP V NP            Rule: NP → N
⇑ NP VP              Rule: VP → V NP
⇑ S                  Rule: S → NP VP
```

Slide 14–16

Grammar G_1 — Structure — V

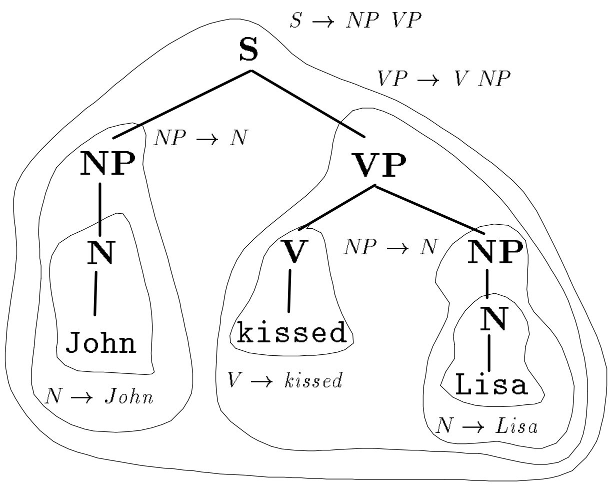
- The grammar assigns a *phrase-structure tree* (or *parse tree*) to each grammatical sentence.
- The tree shows how different phrases are syntactically combined to form a sentence.
- The tree forms the basis for a semantic interpretation.



Slide 14–17

Grammar G_1 — Structure — VI

- Each phrase structure rule generates some part of the tree.
- A rule $X \rightarrow Y\ Z$ generates a node **X** with daughters **Y** & **Z**.



Slide 14–18

Prolog Grammar Rules I

```
s     --> np, vp.
vp    --> v, np.
vp    --> v.
np    --> n.
n     --> [john].
n     --> [lisa].
n     --> [house].
v     --> [died].
v     --> [kissed].

?- s([john, kissed, lisa], []).
   yes
?- s([lisa, died], []).
   yes
?- s([kissed, john, lisa], []).
   no
```

Slide 14–19

Prolog Grammar Rules II

```
?- s(A, []).
   A = [john,died,john] ;
   A = [john,died,lisa] ;
   A = [john,died,house] ;
   A = [john,kissed,john] ;
   A = [john,kissed,lisa] ;
   A = [john,kissed,house] ;
   A = [john,died] ;
   A = [john,kissed] ;
   A = [lisa,died,john] ;
   A = [lisa,died,lisa] ;
   A = [lisa,died,house] ;
   A = [lisa,kissed,john] ;
   A = [lisa,kissed,lisa] ;
   A = [lisa,kissed,house] ;
   A = [lisa,died] ;
   A = [lisa,kissed] ;
         ......
```

Slide 14–20

Prolog Grammar Rules III

```
s(Z) :- np(X), vp(Y), append(X,Y,Z).
np(Z) :- n(Z).
vp(Z) :- v(X), np(Y), append(X,Y,Z).
vp(Z) :- v(Z).

n([john]).
n([lisa]).
n([house]).
v([died]).
v([kissed]).

?- s([john,kissed,lisa]).
   yes
?- s(S).
   S = [john,died,john] ;
   S = [john,died,lisa] ;
         ......
```

Slide 14–21

Prolog Grammar Rules IV

```
s(A,B)   :- np(A,C), vp(C,B).
np(A,B)  :- n(A,B).
vp(A,B)  :- v(A,C), np(C,B).
vp(A,B)  :- v(A,B).

n([john|R],R).
n([lisa|R],R).
n([house|R],R).
v([died|R],R).
v([kissed|R],R).

?- s([john,kissed,lisa], []).
   yes
?- s([john,kissed|R], []).
   R = [john] ;
   R = [lisa] ;
         ......
```

Slide 14–22

Generating Parse Trees

```
s(s(NP,VP))      --> np(NP), vp(VP).
vp(vp(V, NP))    --> v(V), np(NP).
vp(vp(V))        --> v(V).
np(np(N))        --> n(N).
n(n(john))       --> [john].
n(n(lisa))       --> [lisa].
n(n(house))      --> [house].
v(n(died))       --> [died].
v(n(kissed))     --> [kissed].

?- s(S, [john, kissed, lisa], []).
S=s(np(n(john)),vp(n(kissed),np(n(lisa))))
?- s(S, [lisa, died], []).
S=s(np(n(lisa)),vp(n(died)))
?- s(S, [john, died, lisa], []).
S=s(np(n(john)),vp(n(died),np(n(lisa))))
```

Slide 14–23

Ambiguity I

Lexical ambiguity:

homographic

- spelled the same
- *bat* (wooden stick/animal)
- *import* (noun/verb)

polysemous

- different but related meanings
- *neck* (part of body/part of bottle/narrow strip of land)

homophones

- sound the same
- to/too/two

Syntactic ambiguity:

- More than one parse.
- Many missiles have many war-heads.

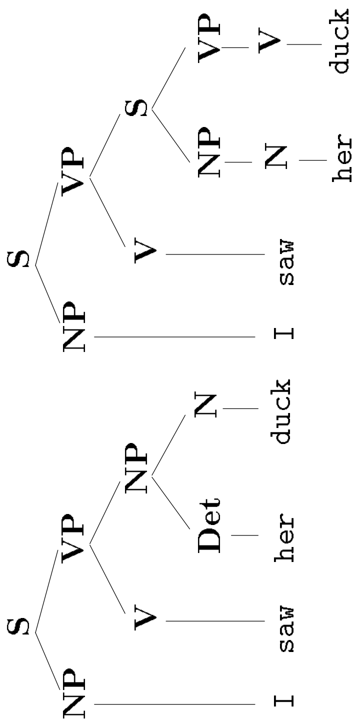
Slide 14–24

Ambiguity II

```
s(s(NP,VP)) --> np(NP), vp(VP).
vp(vp(V, NP)) --> v(V), np(NP).
vp(vp(V, S)) --> v(V), s(S).
vp(vp(V)) --> v(V).
np(np(Det,N)) --> det(Det), n(N).
np(np(N)) --> n(N).
n(n(i)) --> [i].
n(n(duck)) --> [duck].
v(v(duck)) --> [duck].
v(v(saw)) --> [saw].
n(n(saw)) --> [saw].
n(n(her)) --> [her].
det(det(her)) --> [her].

?- s(S, [i, saw, her, duck], []).
```

Slide 14–25

Ambiguity III




```
S = s(np(n(i)),vp(v(saw),np(det(her),n(duck))))
S = s(np(n(i)),vp(v(saw),s(np(n(her)),vp(v(duck)))))
```

Slide 14–26

Pascal Declarations I

```
?- decl([const, a, =, 5, ;,
        var, x, :, 'INTEGER', ;], []).
   yes
?- decl([const, a, =, a, ;,
        var, x, :, 'INTEGER', ;], []).
   no

decl --> const_decl, type_decl,
        var_decl, proc_decl.

% Constant declarations
const_decl --> [ ].
const_decl -->
   [const], const_def, [;], const_defs.

const_defs --> [ ].
const_defs --> const_def, [;], const_defs.
const_def --> identifier, [=], constant.

identifier --> [X], {atom(X)}.
constant --> [X], {(integer(X); float(X))}.
```

Slide 14–27

Pascal Declarationss II

```
% Type declarations
type_decl --> [ ].
type_decl --> [type], type_def, [;], type_defs.

type_defs --> [ ].
type_defs --> type_def, [;], type_defs.
type_def --> identifier, [=], type.

type --> ['INTEGER']. type --> ['REAL'].
type --> ['BOOLEAN']. type --> ['CHAR'].

% Variable decleclarations
var_decl --> [ ].
var_decl --> [var], var_def, [;], var_defs.

var_defs --> [ ].
var_defs --> var_def, [;], var_defs.
var_def --> id_list, [:], type.

id_list --> identifier.
id_list --> identifier, [','], id_list.
```

Slide 14–28

Pascal Declarations III

```
% Procedure declarations
proc_decl --> [ ].
proc_decl --> proc_heading, [;], block.

proc_heading -->
   [procedure], identifier, formal_param_part.

formal_param_part --> [ ].
formal_param_part -->
   ['('], formal_param_section, [')'].

formal_param_section --> formal_params.
formal_param_section -->
   formal_params, [;], formal_param_section.

formal_params --> value_params.
formal_params --> variable_params.
value_params --> var_def.
variable_params --> [var], var_def.
```

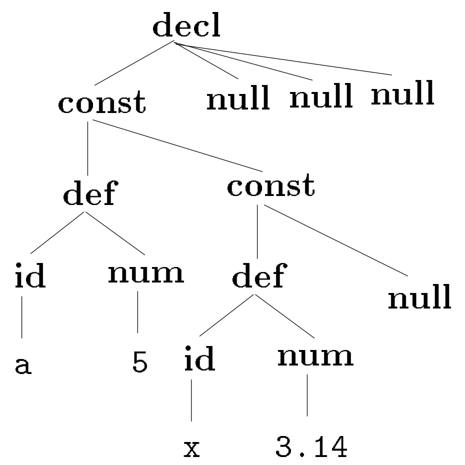
Slide 14–29

Pascal Declarations IV

```
decl(decl(C, T, V, P)) -->
   const_decl(C), type_decl(T),
   var_decl(V), proc_declaration(P).

const_decl(const(null)) --> [ ].
const_decl(const(D, Ds)) -->
   [const], const_def(D),
   [;], const_defs(Ds).

const_defs(null) --> [ ].
const_defs(const(D, Ds)) -->
   const_def(D), [;], const_defs(Ds).

const_def(def(I, C)) -->
   ident(I), [=], const(C).

ident(id(X)) --> [X],
   {atom(X)}.
const(num(X)) --> [X],
   {(integer(X); float(X))}.
```

Slide 14–30

Pascal Declarations IV




```
?- decl(S, [const, a, =, 5, ;,
          x, =, 3.14, ;], []).
   S = decl(
         const(def(id(a),num(5)),
            const(def(id(x),num(3.14)),
                  null)),
         null,null,null)
```

Slide 14–31

Number Conversion I

```
?- number(V, [sixty, three], []).
   V = 63
?- number(V,[one,hundred,and,fourteen],[]).
   V = 114
?- number(V,[nine,hundred,and,ninety,nine],[]).
   V = 999
?- number(V, [fifty, ten], [])
   no
number(0) --> [zero].
number(N) --> xxx(N).

xxx(N) --> digit(D), [hundred], rest_xxx(N1),
           {N is D * 100+N1}.
xxx(N) --> xx(N).

rest_xxx(0) --> [ ].
rest_xxx(N) --> [and], xx(N).

xx(N) --> digit(N).
xx(N) --> teen(N).
xx(N) --> tens(T), rest_xx(N1), {N is T+N1}.
```

Slide 14–32

Number Conversion II

```
rest_xx(0) --> [ ].
rest_xx(N) --> digit(N).

digit(1) --> [one].  teen(10) --> [ten].
digit(2) --> [two].  teen(11) --> [eleven].
digit(3) --> [three].teen(12) --> [twelve].
digit(4) --> [four]. teen(13) --> [thirteen].
digit(5) --> [five]. teen(14) --> [fourteen].
digit(6) --> [six].  teen(15) --> [fifteen].
digit(7) --> [seven].teen(16) --> [sixteen].
digit(8) --> [eight].teen(17) --> [seventeen].
digit(9) --> [nine]. teen(18) --> [eighteen].
                     teen(19) --> [nineteen].
tens(20) --> [twenty].
tens(30) --> [thirty].
tens(40) --> [forty].
tens(50) --> [fifty].
tens(60) --> [sixty].
tens(70) --> [seventy].
tens(80) --> [eighty].
tens(90) --> [ninety].
```

Slide 14–33

Expression Evaluation I

```
?- expr(X, "234+345*456", []).
   X = 157554

expr(Z) --> term(X), "+", expr(Y),
        {Z is X + Y}.
expr(Z) --> term(X), "-", expr(Y),
        {Z is X - Y}.
expr(Z) --> term(Z).

term(Z) --> num(X), "*", term(Y),
        {Z is X * Y}.
term(Z) --> num(X), "/", term(Y),
        {Z is X /Y }.
term(Z) --> num(Z).
```

Slide 14–34

Expression Evaluation II

```
num(C) --> "+", num(C).
num(C) --> "-", num(X), {C is -X}.
num(X) --> int(0, X).

int(L, V) --> digit(C),
        {V is L * 10 +C}.
int(L, X) --> digit(C),
        {V is L* 10 +C},
        int(V, X).

digit(X) --> [C],
        {"0" =< C, C =< "9",
        X is C-"0"}.
```

Slide 14–35