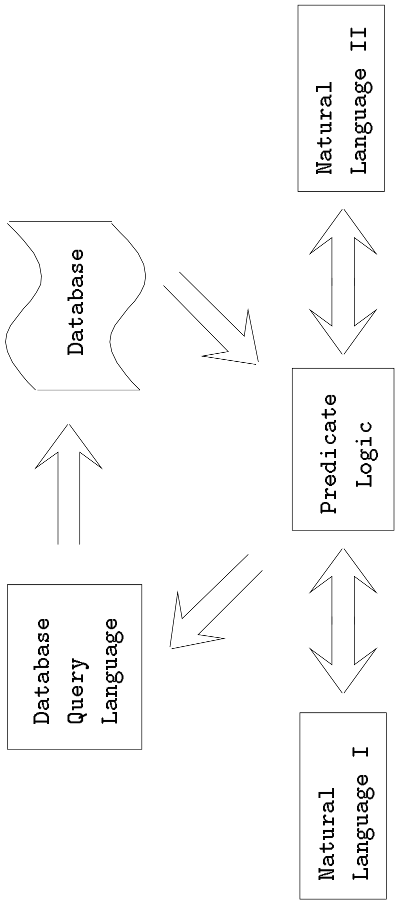
University of Auckland

07.363
Functional and Logic Programming

Christian Collberg
October 7, 1994

Natural Language III

Predicate Logic & Natural Language



Slide 18–1

Predicate Logic & Prolog

John sleeps *(English)*
`sleeps(John)` (Logic)
`sleeps(john)` (Prolog)

John likes Lisa *(English)*
`likes(John, Lisa)` (Logic)
`likes(john, lisa)` (Prolog)

Every bird flies
$\forall b \bullet \texttt{bird}(b) \Rightarrow \texttt{flies}(b)$
`all(X, bird(X) => flies(X))`

Some birds don't fly
$\exists b \bullet \texttt{bird}(b) \Rightarrow \neg\texttt{flies}(b)$
`exists(X, bird(X) => not flies(X))`

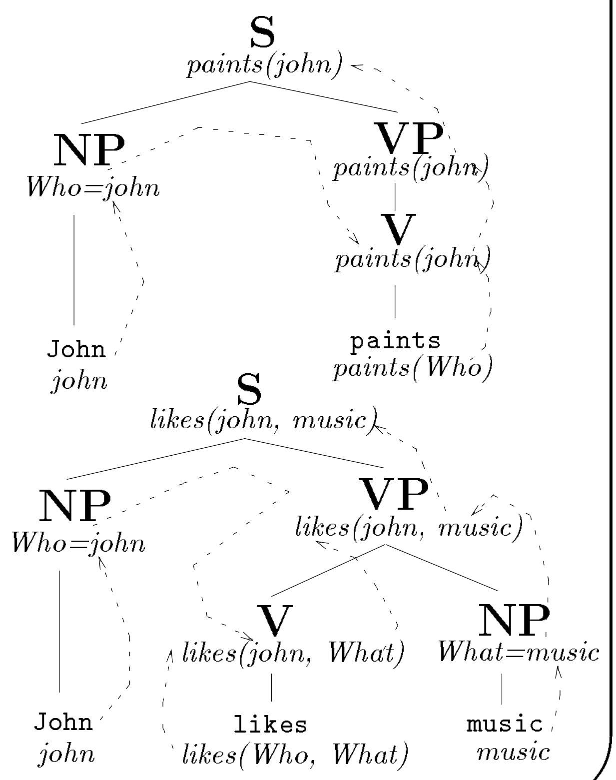
Slide 18–2

English $\Longleftrightarrow$ Logic I

```
s(M) --> np(Who), vp(Who, M).

np(john)-->[john].  np(music)-->[music].

vp(Who, Meaning) -->
   intrans_v(Who, Meaning).
vp(Who, Meaning) -->
   trans_v(Who, What, Meaning), np(What).

iv(Who, paints(Who)) --> [paints].
tv(Who, What, likes(Who, What))-->[likes].

?- s(M, [john, paints], []).
   M = paints(john).
?- s(M, [john, likes, music], []).
   M = likes(john, music)
?- s(likes(john, music), A, []).
   A = [john, likes, music]
```

Slide 18–3

English ⟺ Logic II



Slide 18–4

English ⟺ Logic III

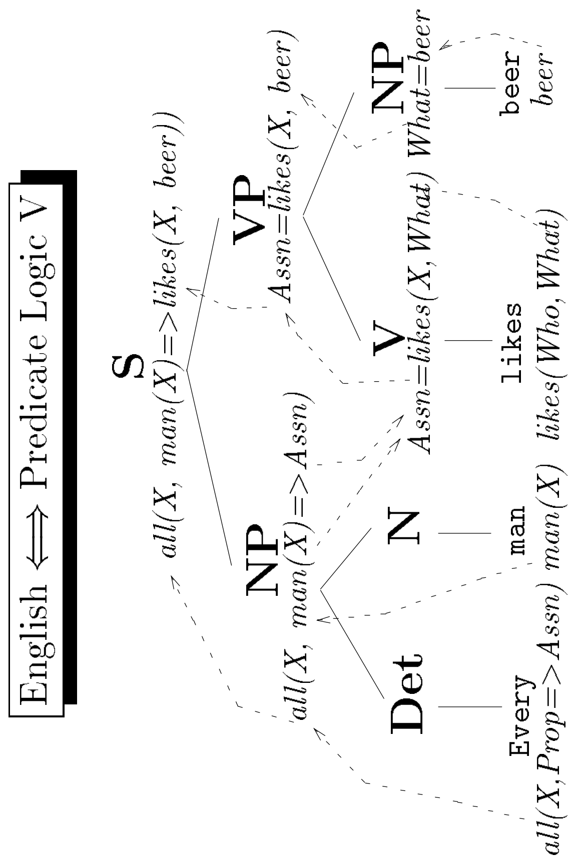
```
:- op(500, xfy, &).  :- op(500, xfy, =>).

s(Meaning) -->
   np(Who, Assn, Meaning), vp(Who, Assn).

det(X, Prop, Assn,
   exists(X, Prop & Assn)) --> [a].
det(X, Prop, Assn,
   all(X, Prop => Assn)) --> [every].

np(X, Assn, Meaning) -->
   noun(X, Meaning).
np(X, Assn, Meaning) -->
   det(X, Prop, Assn, Meaning),
   noun(X, Prop).

noun(X, man(X)) --> [man].
noun(X, music) --> [music].
```

Slide 18–5

English ⟺ Logic IV

```
vp(Who, Meaning) -->
   intrans_v(Who, Meaning).
vp(Who, Meaning) -->
   trans_v(Who, What, Meaning),
   np(_, _, What).

intrans_v(Who, sleeps(Who)) --> [sleeps].
trans_v(Who, What,
   likes(Who, What)) --> [likes].

?- s(M, [a, man, likes, music], []).
   M = exists(_6, man(_6) & likes(_6, m))
?- s(M, [every, man, likes, music], []), .
   M = all(_6, man(_6) => likes(_6, m))
?- s(M, [a, man, sleeps], []), .
   exists(_4, man(_4) & sleeps(_4))
?- s(all(X,man(X)=>likes(X, music)),A,[]).
   A = [every,man,likes,music]
```

Slide 18–6



Slide 18–7

Maori $\Longleftrightarrow$ Predicate Logic I

Ka	*pai*	*te*	*pia*	*ki*	*a*	*Hone*	
tense	good	the	beer	to	pers	John	
VP		NP		PP			

'John likes beer'

Ka	*pai*	*te*	*pia*	*ki*	*teehati*	*tangata*	
tense	good	the	beer	to	a	man	

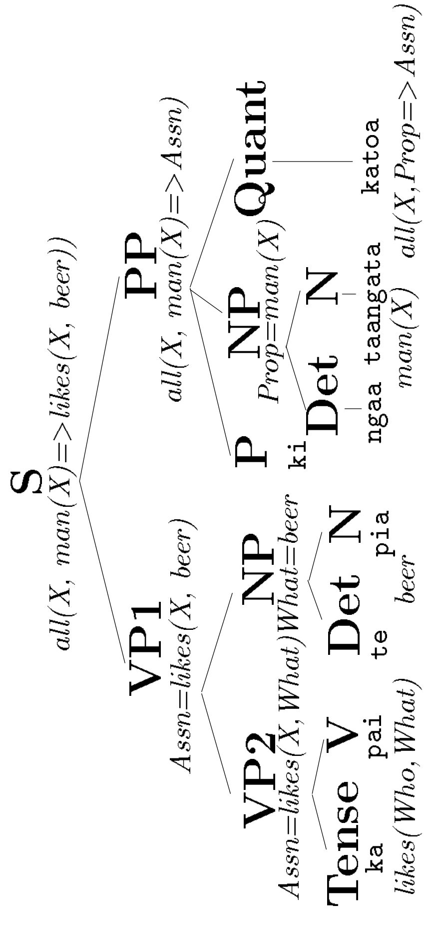
'A man likes beer'

Ka	*pai*	*te*	*pia*	*ki*	*ngaa*	*taangata*	*kotoa*
tense	good	the	beer	to	the-Logic	men	all

'Every man likes beer'

Slide 18–8

Maori $\Longleftrightarrow$ Predicate Logic II



Slide 18–9

Maori $\Longleftrightarrow$ Predicate Logic III

```
s(maori, Meaning) -->
   trans_vp(maori, Who, Assn),
   pp(maori, Who, Assn, Meaning).

det(maori) --> [a].  % pers
det(maori) --> [te].  % the
det(maori) --> [ngaa].  % the-Logic

quant(maori, Who, Prop, Assn, % a
   exists(Who, Prop & Assn))-->[teetahi].
quant(maori, Who, Prop, Assn, % every
   all(Who, Prop => Assn)) --> [kotoa].

np(maori, Who, Meaning, Meaning) -->
   det(maori), noun(maori, Who, Who).
np(maori, Who, Assn, Meaning) -->
   quant(maori, Who, Prop, Assn, Meaning),
   noun(maori, Who, Prop).
```

Slide 18–10

Maori $\Longleftrightarrow$ Predicate Logic IV

```
np(maori, Who, Assn, Meaning) -->
   det(maori),
   noun(maori, Who, Prop),
   quant(maori, Who, Prop, Assn, Meaning).

pp(maori, Who, Assn, Meaning) -->
   [ki], np(maori, Who, Assn, Meaning).

noun(maori, Who, man(Who)) --> [tangata].
noun(maori, Who, man(Who)) --> [tangaata].
noun(maori, beer, beer) --> [pia].
noun(maori, john, john) --> [hone].
intrans_v(maori, Who,
      sleeps(Who)) --> [sleeps].
trans_vp(maori, Who, Assn) -->
   tense(maori),
   trans_v(maori, Who, What, Assn),
   np(maori, What, Assn, Assn).
```

Slide 18–11

Maori ⟺ Predicate Logic VI

```
tense(maori) --> [ka].
trans_v(maori, Who, What,
   likes(Who, What)) --> [pai].

% A man likes beer.
?- s(maori, M, [ka, pai, te, pia, ki,
         teetahi, tangata], []).
   M=exists(_1, man(_1) & likes(_1, beer))

% Every man likes beer.
?- s(maori, M, [ka, pai, te, pia, ki,
         ngaa, tangata, kotoa], []).
   M=all(_3, man(_3)=>likes(_3, beer))

% John likes beer.
?- s(maori, M, [ka, pai, te, pia,
         ki, a, hone], []).
   M = likes(john, beer)
```

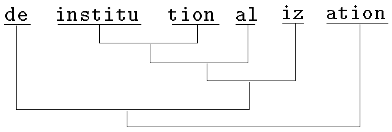
Slide 18–12

Machine Translation I

```
e2m(E, M) :-
   s(english, Logic, E, []),
   s(maori, Logic, M, []).

?- e2m([a, man, likes, beer], M).
   M = [ka,pai,a,pia,ki,teetahi,tangata]
?- e2m(E, [ka, pai, te, pia, ki,
         teetahi, tangata]).
   E = [a, man, likes, beer]

?- e2m([every, man, likes, beer], M).
   M=[ka,pai,a,pia,ki,kotoa,tangata] %Wrong
?- e2m(E, [ka, pai, te, pia, ki,
         ngaa, tangata, kotoa]).
   M = [every, man, likes, beer]

?- e2m([john, likes, beer], M), write(M), nl.
   M = [ka, pai, te, pia, ki, a, hone]
?- e2m(E, [ka, pai, te, pia, ki, a, hone]).
   M = [john, likes, beer]
```

Slide 18–13

Morphology I



morpheme Smallest unit of meaning.

stem Basic form of a word.

affix Morpheme added on to a stem.

prefix Affix added to beginning of word (English: ≈ 75, *un-*, *de-*).

suffix Affix added to end of word (English: ≈ 80, *-ing*, *-ed*).

infix Affix inserted within word. (Arabic, Languages of SE Asia).

stemming Morph. analysis of a word.

Slide 18–14

Morphology II – Stemming

un-	*bend*	*-ing*
prefix	stem	suffix

Stemming compound words:

- In German we must also break up compound words:

schädlings	*-bekämpfungs*	*-mittel*
insect	combat	means

"pesticide"

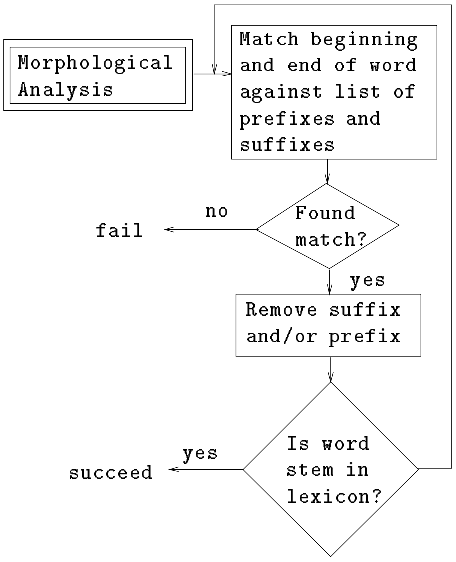
Applications:

- Speeds up searching.
- Reduces storage requirements for the lexicon.

Slide 18–15

Morphology III – Stemming

- Spelling changes: *easy* ↔ *easily*.
- Many affixes: *institution-al-iz-ed*
- Wrong affixes within affixes: $\boxed{co}$*unter-*. Longer affixes first.



Slide 18–16

Morphology IV – Adverbs

- The suffix *-ly* turns an adjective into an adverb: *sweetly, softly*.

Stemming cases:

1. The suffix can be removed without changing the stem:
   - *mainly* ↔ *main*.
2. A final i is changed to a y:
   - *easily* ↔ *easy*.
3. A final *-le* must be added:
   - *ably* ↔ *ab* ↔ *able*
4. *-ly* is not an affix:
   - *rely, belly*.

Slide 18–17

Morphology V – Adverbs

```
IF the word ends in -ly THEN
   remove the -ly;
   IF string matches a stem THEN
      succeed
   ELSIF the string ends in -i THEN
      change -i to y;
      IF string matches a stem
      THEN succeed ELSE fail END
   ELSE
      add -le;
      IF string matches a stem
      THEN succeed
      ELSE fail END
   END
ELSE
   fail
END
```

Slide 18–18

Morphology VI – Adverbs

```
adj(main).  adj(easy).  adj(able).
adv(fast).  adv(yesterday).

adv(X) :-
   name(X, XL),
   append(SL, "ly", XL),
   is_adv(SL).

% main -> mainly
is_adv(SL) :- name(S, SL), adj(S).

% easy -> easily
is_adv(SL) :-
   append(S1L, "i", SL),
   append(S1L, "y", S2L),
   name(S, S2L), adj(S).
```

Slide 18–19

Morphology VII – Adverbs

```
% ably -> able
is_adv(SL) :-
   append(SL, "le", S1L),
   name(S, S1L), adj(S).

?- adv(mainly).
   yes

?- adv(easily).
   yes

?- adv(ably).
   yes

?- adv(rely).
   no
```

Slide 18–20

Morphology VIII – Verbs

```
reg_verb(talk).  reg_verb(look).
reg_verb(hum).  reg_verb(rub).

ireg_verb([see, saw, seen]).
ireg_verb([go, went, gone]).

verb(V) :- ireg_verb(X), member(V, X).

verb(V) :- reg_verb(V).
verb(V) :-
   name(V, VL),
   append(V1L, "ed", VL),
   is_reg_verb(V1L).

% look -> looked
is_reg_verb(VL) :-
   name(V, VL), reg_verb(V).
```

Slide 18–21

Morphology IX – Verbs

```
% hum -> hummed; rub -> rubbed
is_reg_verb(VL) :-
   append(_, [X, X], VL),
   append(V1L, [_], VL),
   name(V1, V1L), reg_verb(V1).

?- verb(saw).
   yes
?- verb(look).
   yes
?- verb(looked).
   yes
?- verb(rubbed).
   yes
?- verb(sawed).
   no
```

Slide 18–22