Transfer

Martin Kay

Stanford University and
The University of the Saarland

Spelling Conventions

flap+s → flaps
flash+s → flashes

When *s* has a sibilant and a morpheme boundary to the left and a word boundary to the right, replace it with *es*.

Compile:
s → es / (s | z | c h | s h) + __ #

	after	c	s	z	+	#	h	?
0	*start*	1	2	3	0	0	0	0
1	c	1	2	3	0	0	3	0
2	s	1	2	3	4	0	3	0
3	c h, s h, z	1	2	3	4	0	0	0
4	sib +	1	6	3	0	0	0	0
5	sib + s	1	2	3	0	0	0	0
6	?	1	2	3	0	0	0	0

f l a p + s #
0 0 0 0 0 0 2 0

	after	c	s	z	+	#	h	?
0	*start*	1	2	3	0	0	0	0
1	c	1	2	3	0	0	3	0
2	s	1	2	3	4	0	3	0
3	c h, s h, z	1	2	3	4	0	0	0
4	sib +	1	5	3	0	0	0	0
5	sib + s	1	2	3	0	0	0	0
6	?	1	2	3	0	0	0	0

f l a s h + s #
0 0 0 0 2 3 4 5 ↔ 0

	after	c	s	z	+:ε	+:e	#:ε	h	?
0	***start***	1	2	3	0		0	0	0
1	**c**	1	2	3	0		0	3	0
2	**s**	1	2	3	4	5	0	3	0
3	**c h, s h, z**	1	2	3	4	5	0	0	0
4	**sib +:ε**							0	0
5	**sib +:e**								
6	**sib +:ε s**	1	2	3	4			3	0
7	**sib +:e s**						0		
8	**?**	1	2	3	0		0	0	

Finite-state Transducers

Two languages in parallel

f	l	a	p	+:ε	s	#:ε
0	0	0	0	0	0	0	0

	after	c	s	z	+:ε	+:e	#:ε	h	?
0	***start***	1	2	3	0		0	0	0
1	**c**	1	2	3	0		0	3	0
2	**s**	1	2	3	4	5	0	3	0
3	**c h, s h, z**	1	2	3	4	5	0	0	0
4	**sib +:ε**	1	6	3	0		0	0	0
5	**sib +:e**		7						
6	**sib +:ε s**	1	2	3	4			3	0
7	**sib +:e s**						0		
8	**?**	1	2	3	0		0	0	

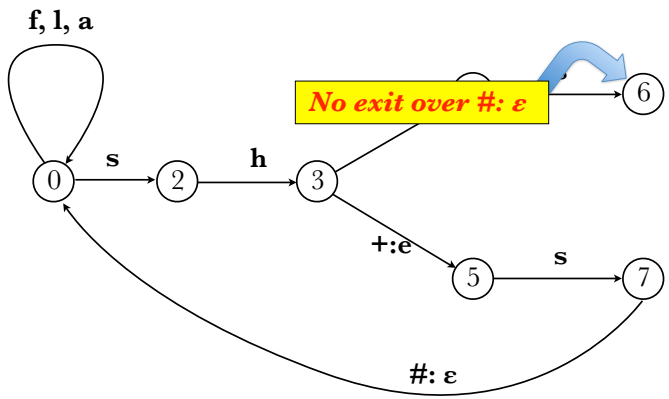
f	l	a	s	h	+:ε	s	#
0	0	0	0	2	3	4	6	✗

	after	c	s	z	+:ε	+:e	#:ε	h	?
0	***start***	1	2	3	0		0	0	0
1	**c**	1	2	3	0		0	3	0
2	**s**	1	2	3	4	5	0	3	0
3	**c h, s h, z**	1	2	3	4	5	0	0	0
4	**sib +:ε**	1	6	3	0		0	0	0
5	**sib +:e**		7						
6	**sib +:ε s**	1	2	3	4		✗	3	0
7	**sib +:e s**						0		
8	**?**	1	2	3	0		0	0	

f	l	a	s	h	+:ε	s	#
0	0	0	0	2	3	4	6	✗

	after	c	s	z	+:ε	+:e	#:ε	h	?
0	***start***	1	2	3	0		0	0	0
1	**c**	1	2	3	0		0	3	0
2	**s**	1	2	3	4	5	0	3	0
3	**c h, s h, z**	1	2	3	4	5	0	0	0
4	**sib +:ε**	1	6	3	0		0	0	0
5	**sib +:e**		7						
6	**sib +:ε s**	1	2	3	4			3	0
7	**sib +:e s**						0		
8	**?**	1	2	3	0		0	0	

f	l	a	s	h	+:e	s	#
0	0	0	0	2	3	5	7	0



A Context-free Relation

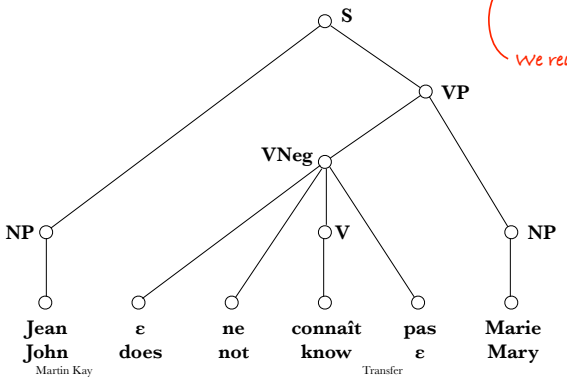
S → <s, ε> <(, ε > NP VP <), ε >
NP → <np, ε> <(, ε > DET N <), ε >
VP → <vp, ε ><(, ε > V NP <), ε >
DET → <det, th
N → <n, dog
N → <n, cat
V → <v, chased>

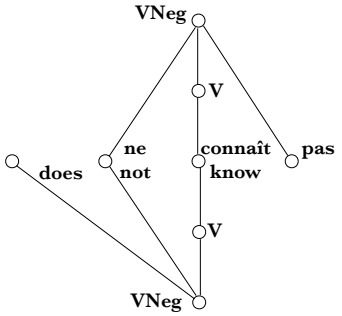
Context-free Transducers
Two languages in parallel

```
s ( np ( det  n ) vp (   v    np ( det  n  )))
        the dog       chased      the cat
```


S → NP_{subj} VP_{pred}, NP_{subj} VP_{pred}
NP → Jean, John
NP → Marie, Mary
VP → $VNeg_n$, $VNeg_n$
$VNeg_n$ → ε ne V_v pas, does not V_v ε
V → connaît, know

We really don't need these





$VNeg_n$ → ne V_v pas, does not V_v

Suppose I always want this one!

der Mann	gab	dem Jungen	den Hund
the man	gave	the boy	the dog
der Mann	gab	den Hund	dem Jungen
the man	gave	the dog	the boy
den Hund	gab	der Mann	dem Jungen
the dog	gave	the man	the boy
dem Jungen	gab	der Mann	den Hund
the boy	gave	the man	the dog

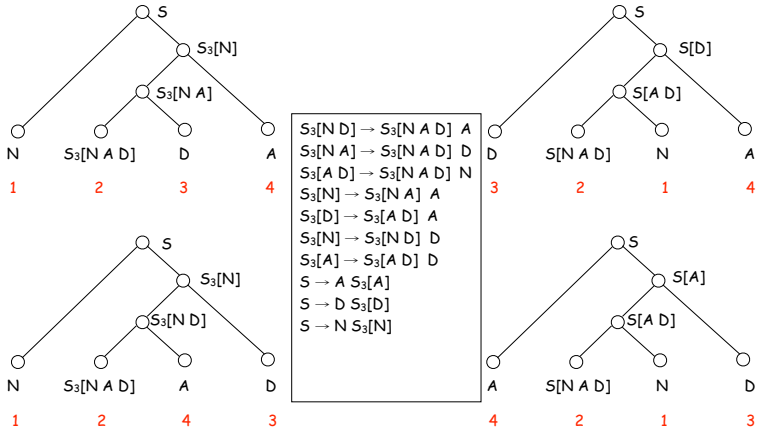
All the same

$S \rightarrow N_n\ V_v\ D_d\ A_a,\ \ N_n\ V_v\ N_d\ N_a$
$S \rightarrow N_n\ V_v\ A_a\ D_g,\ \ N_n\ V_v\ N_d\ N_a$
$S \rightarrow A_a\ V_v\ N_n\ D_g,\ \ N_n\ V_v\ N_d\ N_a$
$S \rightarrow D_g\ V_v\ N_n\ A_a,\ \ N_n\ V_v\ N_d\ N_a$

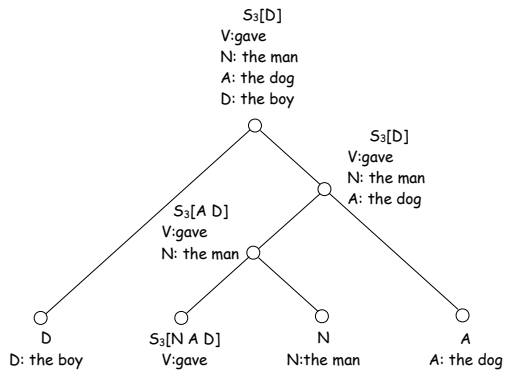
Suppose I also want
"The man gave a dog to the boy"

$S \rightarrow N_n\ V_v\ D_d\ A_a,\ \ N_n\ V_v\ N_d\ N_a$
$S \rightarrow N_n\ V_v\ A_a\ D_g,\ \ N_n\ V_v\ N_d\ N_a$
$S \rightarrow A_a\ V_v\ N_n\ D_g,\ \ N_n\ V_v\ N_d\ N_a$
$S \rightarrow D_g\ V_v\ N_n\ A_a,\ \ N_n\ V_v\ N_d\ N_a$

$S \rightarrow N_n\ V_v\ D_d\ A_a,\ \ N_n\ V_v\ N_a$ **to** N_d
$S \rightarrow N_n\ V_v\ A_a\ D_g,\ \ N_n\ V_v\ N_a$ **to** N_d
$S \rightarrow A_a\ V_v\ N_n\ D_g,\ \ N_n\ V_v\ N_a$ **to** N_d
$S \rightarrow D_g\ V_v\ N_n\ A_a,\ \ N_n\ V_v\ N_a$ **to** N_d

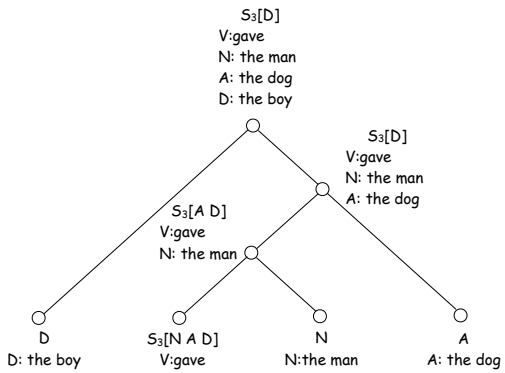


$S_3[N\ D] \rightarrow S_3[N\ A\ D]\ A$
$S_3[N\ A] \rightarrow S_3[N\ A\ D]\ D$
$S_3[A\ D] \rightarrow S_3[N\ A\ D]\ N$
$S_3[N] \rightarrow S_3[N\ A]\ A$
$S_3[D] \rightarrow S_3[A\ D]\ A$
$S_3[N] \rightarrow S_3[N\ D]\ D$
$S_3[A] \rightarrow S_3[A\ D]\ D$
$S \rightarrow A\ S_3[A]$
$S \rightarrow D\ S_3[D]$
$S \rightarrow N\ S_3[N]$

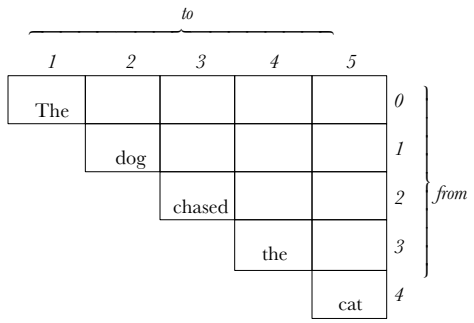


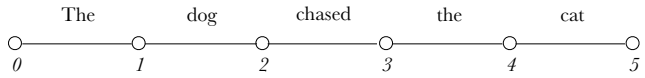
Abstract away from word order

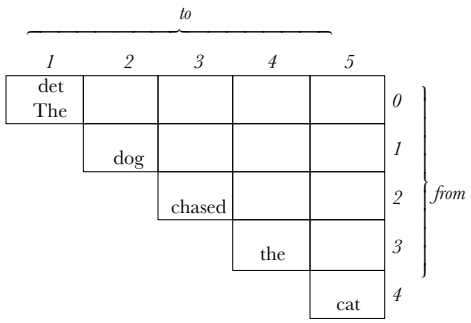
$S_3[N\ D] \rightarrow S_3[N\ A\ D]\ A$
$S_3[N\ A] \rightarrow S_3[N\ A\ D]\ D$
$S_3[A\ D] \rightarrow S_3[N\ A\ D]\ N$
$S_3[N] \rightarrow S_3[N\ A]\ A$
$S_3[D] \rightarrow S_3[A\ D]\ A$
$S_3[N] \rightarrow S_3[N\ D]\ D$
$S_3[A] \rightarrow S_3[A\ D]\ D$
$S \rightarrow A\ S_3[A]$
$S \rightarrow D\ S_3[D]$
$S \rightarrow N\ S_3[N]$

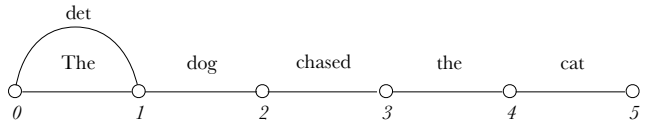


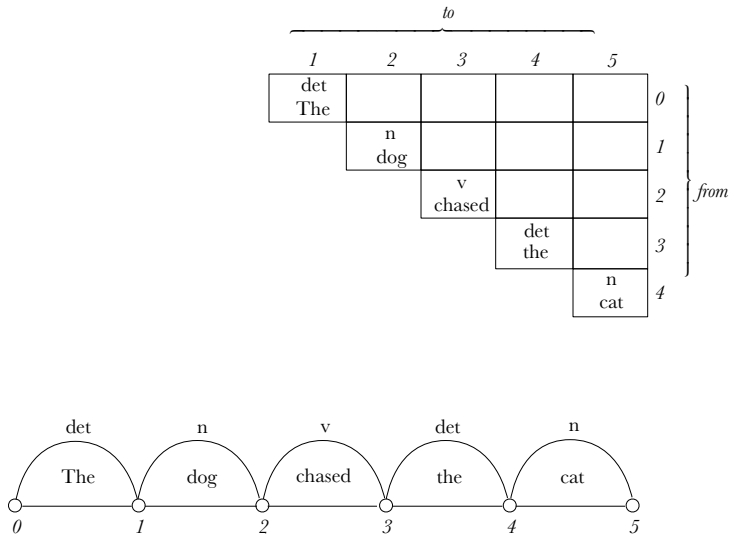
Abstract away from word order
⇒ semantics!

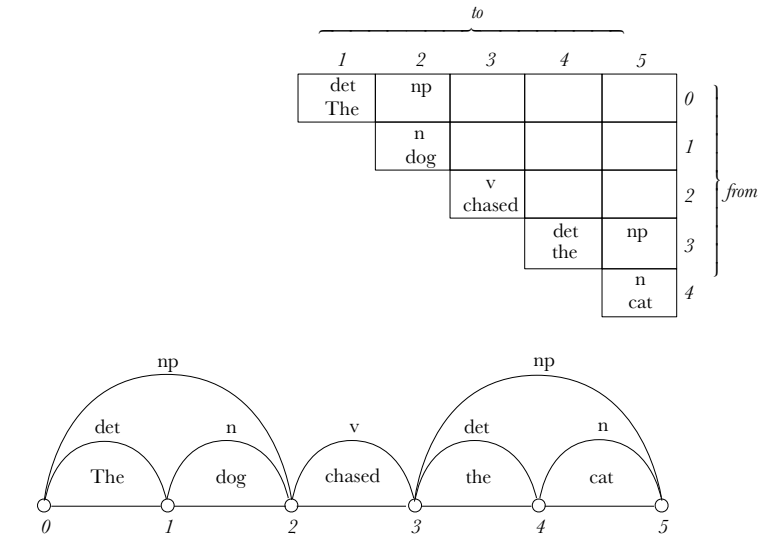


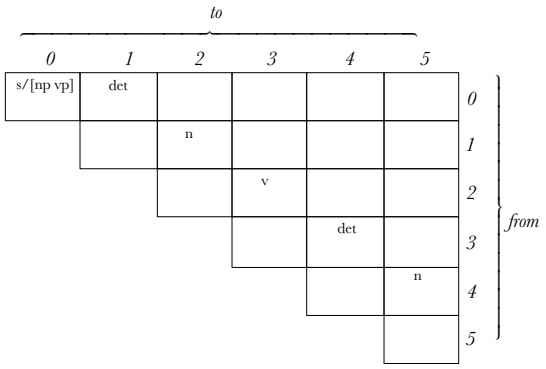






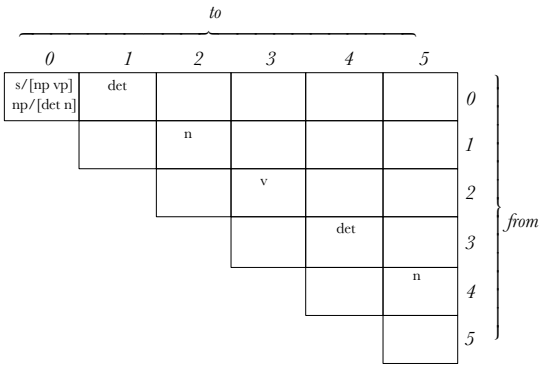


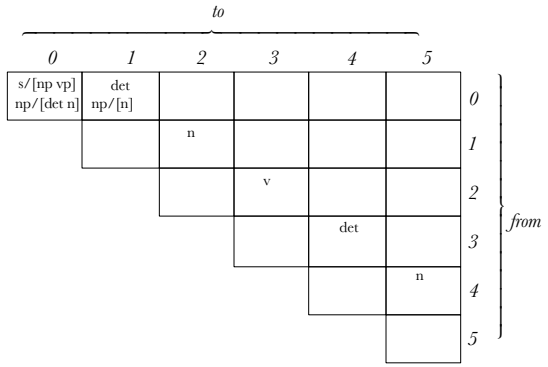


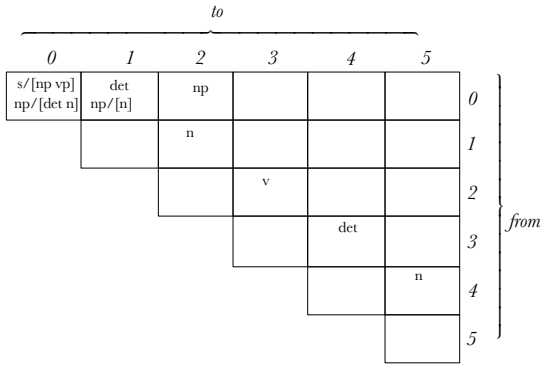


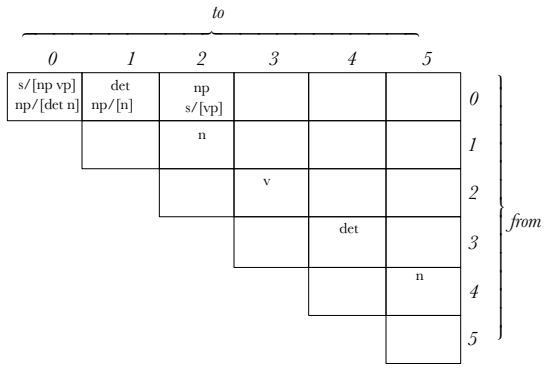
Zero-length items!

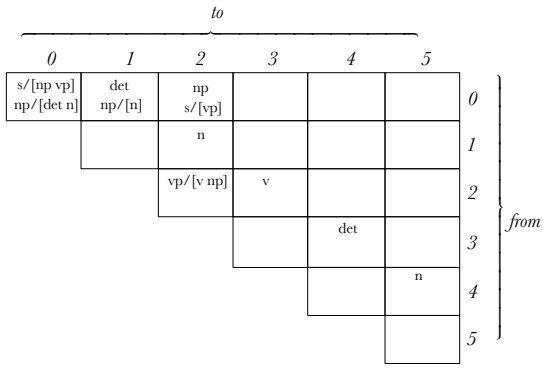
Active Edges





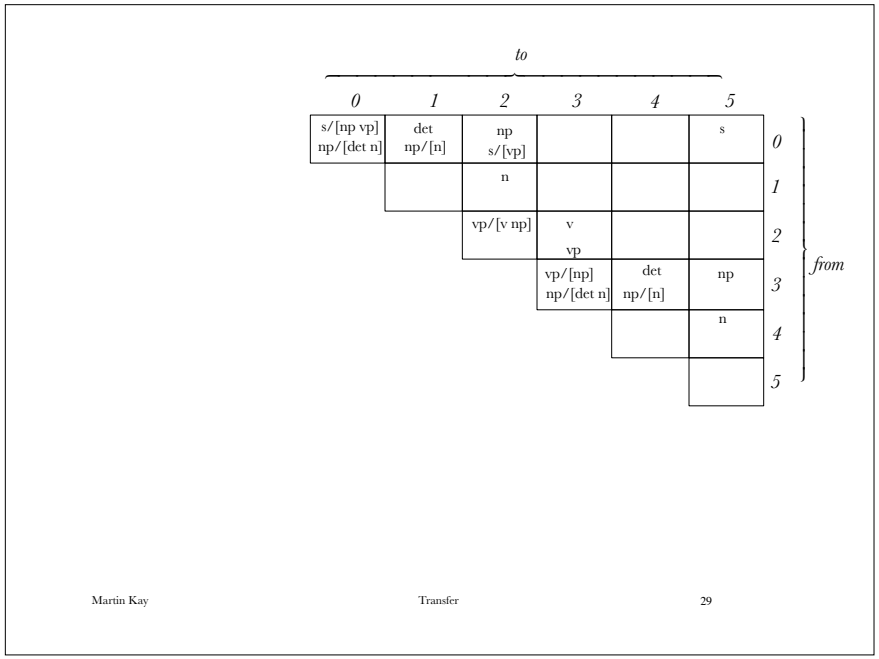


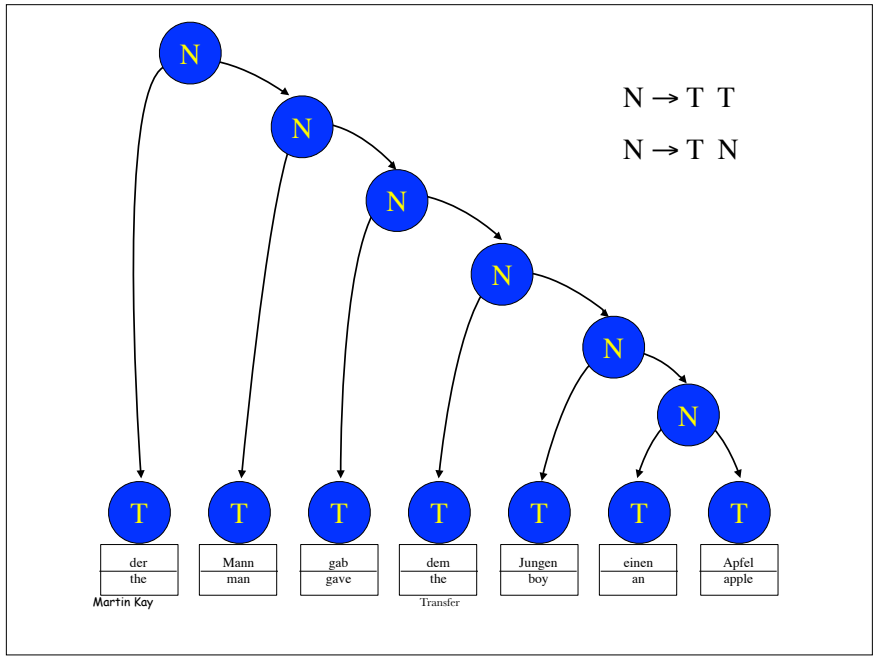


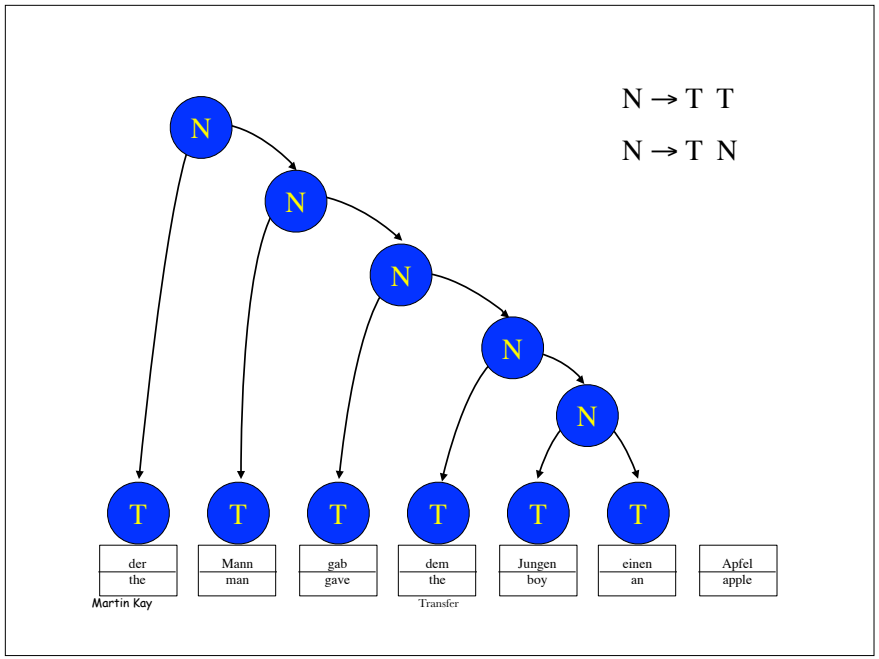


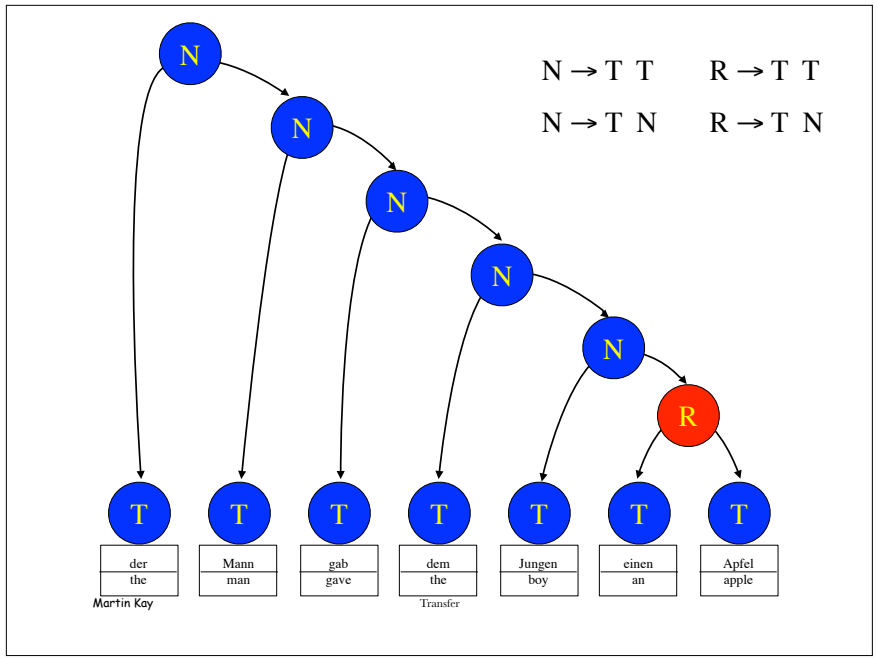
Advantages

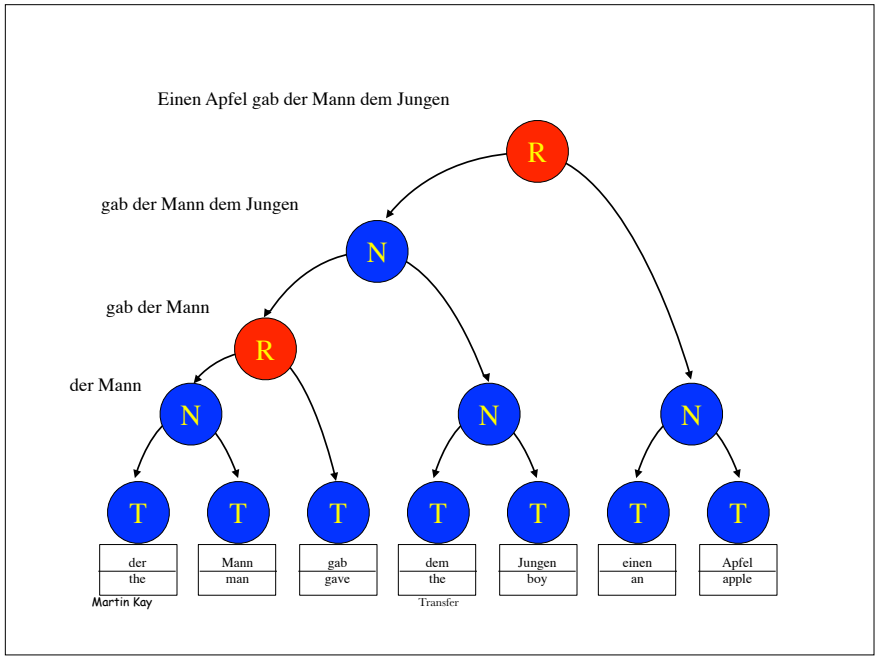
- Rules with right sides of arbitrary length
- Top-down guidance


to
0
1
2
3
4
5
from
s/[np vp]
np/[det n]
det
np/[n]
np
s/[vp]
s
n
vp/[v np]
v
vp
vp/[np]
np/[det n]
det
np/[n]
np
n
Martin Kay
Transfer
29

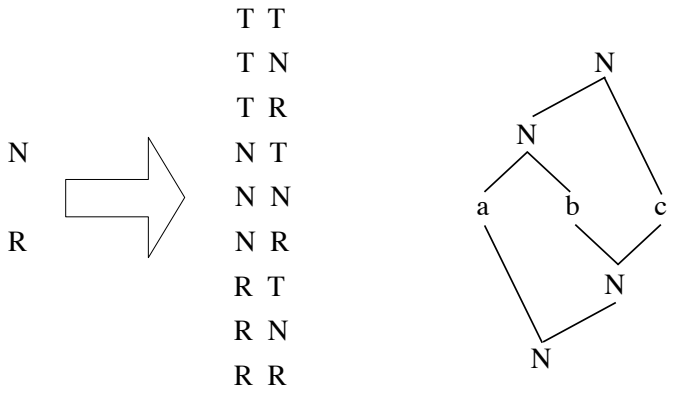

N → T T
N → T N
der
the
Mann
man
gab
gave
dem
the
Jungen
boy
einen
an
Apfel
apple
Martin Kay
Transfer


N → T T
N → T N
der
the
Mann
man
gab
gave
dem
the
Jungen
boy
einen
an
Apfel
apple
Martin Kay
Transfer

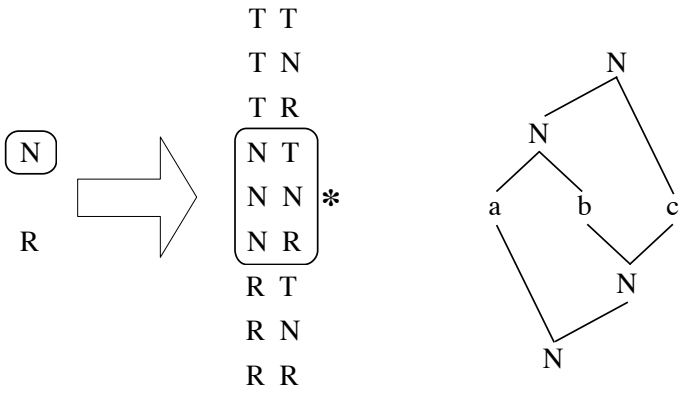

N → T T
N → T N
R → T T
R → T N
der
the
Mann
man
gab
gave
dem
the
Jungen
boy
einen
an
Apfel
apple
Martin Kay
Transfer



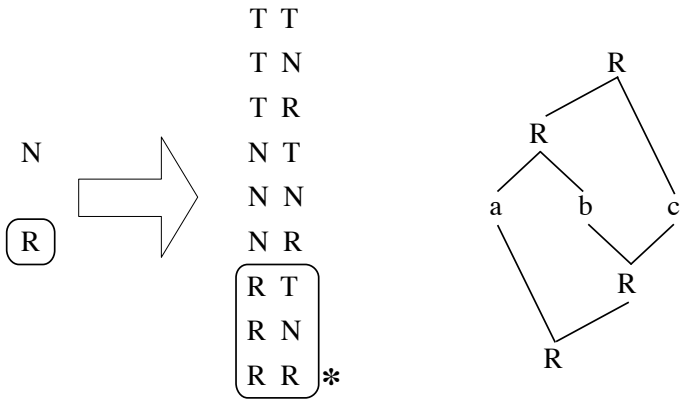
Redundant Structures



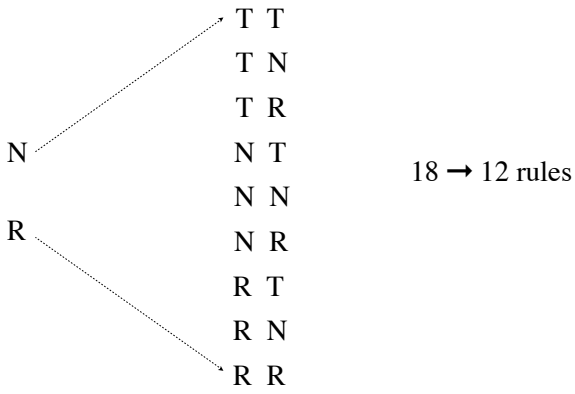
Redundant Structures



Redundant Structures



Redundant Structures



18 → 12 rules

Dekai Wu: ***Inversion Transduction Grammars***

Not all reorderings are possible

Den Hund	hat	der Mann	dem Jungen	gegeben
1		2	3	4
The man	**gave**	**the dog**	**to the boy**	
2	4	1	3	

1	2	3	4	
Den Hund	**hat**	**der Mann**	**dem Jungen**	**gegeben**
3		1	4	2

Center Embedding

This is the house that Jack built.

The house the malt lay in

built.

This is the rat that ate the malt

That lay in the house that Jack built.

This is the cat that killed the rat

That ate the malt that lay in the house that Jack built.

Center Embedding

This is the house that Jack built.

This is the malt that lay in the house that Jack built.

The house the malt the rat ate lay in

That lay in the house that Jack built.

This is the cat that killed the rat

That ate the malt that lay in the house that Jack built.

Center Embedding

This is the house that Jack built.

This is the malt that lay in the house that Jack built.

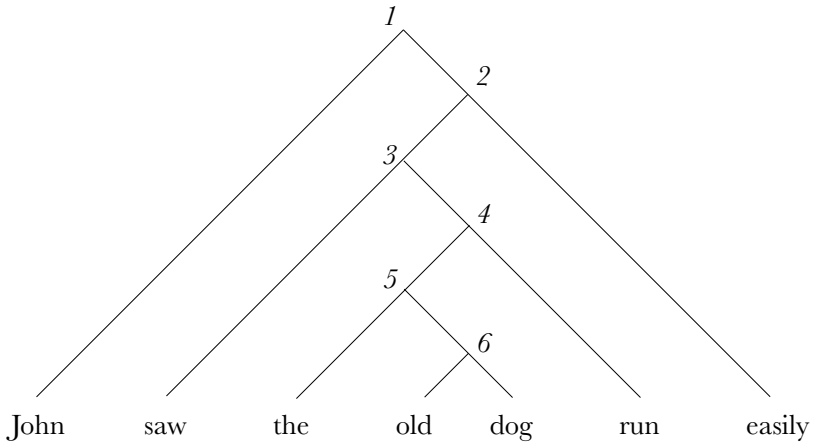
This is the rat that ate the malt

That lay in the house that Jack built.

The house the malt the rat the cat killed ate lay in

ck built.

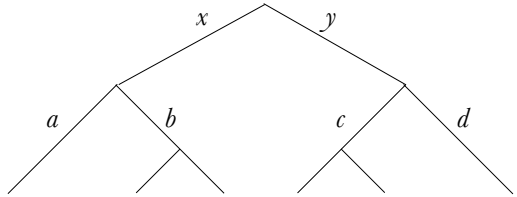
Center Embedding



Computing Embeddings

$$X_{x,y} \rightarrow Y_{a,b}\, Z_{c,d}$$

$$x = \max(a, b+1)$$
$$y = \max(c+1, d)$$



A Touch more Syntax

- **Introduce real grammar rules opportunistically.**
- **With a higher priority than simple ordering rules.**
- **Robustness is not affected.**

Seamantics as Sets of Predicates

The dog saw the cat

dog(d), def(d), saw(s), past(s), cat(c), def(c),
argl (s, d), arg2(s, c)

Grammar:

s(x) → np(y) vp(x, y)

vp(x,y) → v(x,y, z) np(z)

np(x) → det(x) n(x)

Lexicon

Word	Cat	Semantics
cat	n(*x*)	*x*: cat(*x*)
saw	v(*x*, *y*, *z*)	see(*x*), past(*x*), arg1(*x*, *y*), arg2(*x*, *z*)
dog	n(*x*)	*x*: dog(*x*)
the	det(*x*)	*x*: def(*x*)

Initial Agenda

Vertex	Word	Cat	Semantics
d	the	det(d)	d: def(d)
	dog	n(d)	d: dog(d)
s	saw	v(s, d, c)	s: see(s), past(s), arg1(s, d), arg2(s, c)
c	the	det(c)	d: def(d)
	cat	n(c)	d: cat(d)

Initial Agenda

Vertex	Word	Cat	Semantics
d	the	det(d)	d: def(d)
	the	np(d)/n(d)	d: def(d)
	dog	n(d)	d: dog(d)
s	saw	v(s, d, c)	s: see(s), past(s), arg1(s, d), arg2(s, c)
c	saw	v(s, d)/np(c)	s: see(s), past(s), arg1(s, d)
	the	det(c)	d: def(d)
	the	np(c)/(n(c)	d: def(d)
	cat	n(c)	d: cat(d)

Vertex	Word	Cat	Semantics
d	the	det(d)	d: def(d)
	the	np(d)/n(d)	d: def(d)
	dog	n(d)	d: dog(d)
s			
	the dog	np(d)	d: dog(d), def(d)
c	saw	v(s, d)/np(c)	s: see(s), past(s), arg1(s, d)
	the	det(c)	d: def(d)
	the	np(c)/(n(c)	d: def(d)
	cat	n(c)	d: cat(d)
	the cat	np(c)	d: cat(c), def(c)

Vertex	Word	Cat	Semantics
d	the	det(d)	d: def(d)
	the	np(d)/n(d)	d: def(d)
	saw the cat	v(s, d)/np(d)	s: see(s), past(s), arg1(s, d), arg2(s, c), cat(c), def(c)
			g(d)
			st(s), arg1(s, d), arg2(s, c)
c	saw	v(s, d)/np(c)	s: see(s), past(s), arg1(s, d)
	the	det(c)	d: def(d)
	the	np(c)/(n(c)	d: def(d)
	cat	n(c)	d: cat(d)
	the cat	np(c)	c: def(c), cat(c)

Vertex	Word	Cat	Semantics
d	the	det(d)	d: def(d)
	the	np(d)/n(d)	d: def(d)
	dog	n(d)	d: dog(d)
	the dog	np(d)	d: def(d), dog(d)
	saw the cat	v(s, d)/np(d)	s: see(s), past(s), arg1(s, d), arg2(s, c), cat(c), def(c)
s	saw	v(s, d, c)	s: see(s), past(s), arg1(s, d), arg2(s, c)
c	saw	v(s, d)/np(c)	s: see(s), past(s), arg1(s, d)
	the	det(c)	d: def(d)
	the	np(c)/(n(c)	d: def(d)
	cat	n(c)	d: cat(d)
	the cat	np(c)	c: def(c), cat(c)

s: see(s), past(s), arg1(s, d), arg2(s, c)

Vertex	Word	Cat	Semantics
d	the	det(d)	d: def(d)
	the	np(d)/n(d)	d: def(d)
	dog	n(d)	d: dog(d)
	the dog	np(d)	d: def(d), dog(d)
s	saw	v(s, d, c)	s: see(s), past(s), arg1(s, d), arg2(s, c)
	saw the cat	v(s, d)/np(d)	s: see(s), past(s), arg1(s, d), arg2(s, c), cat(c), def(c)
c	saw	v(s, d)/np(c)	s: see(s), past(s), arg1(s, d)
	the	det(c)	d: def(d)
	the	np(c)/(n(c)	d: def(d)
	cat	n(c)	d: cat(d)
	the cat	np(c)	c: def(c), cat(c)

Rewriting rules

String rewriting

$\alpha \rightarrow \beta \ / \ \gamma \_ \ \delta$

$\alpha, \gamma \ \delta$ are strings / regular expressions allowing variable assignment

β is a string

Set rewriting

$\alpha \rightarrow \beta \ / \ \gamma$

α, β are sets allowing variable assignment

γ is a set

The PARC Transfer System
(circa 2000)

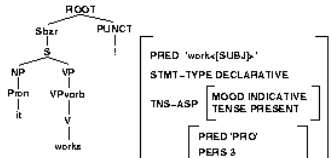
Transfer from LFG F-Structure to F-Structure through the intermediary of a set of Prolog predicates.

Not necessarily limited to F-Structure. Later extended to C-Structure

Characteristics

- **A configuration consists of <u>one</u> <u>set</u> of predicates. Not one for each language.**
- **Completely deterministic.**
- **Ordered rules: Each rule operates on the configuration left by the preceding one.**
- **Anything that is not explicitly transferred to the target language remains in the source language**

Flat Structure



PRED 'work<[SUBJ]>'
STMT-TYPE DECLARATIVE
TNS-ASP [MOOD INDICATIVE, TENSE PRESENT]
SUBJ [PRED 'PRO', PERS 3, ANIM -, GEND NEUT, CASE NOM, PRON-TYPE PERS, NUM SG]

```
pred (0, word)
smt_type(0, declarative)
tns_asp(0, 1)
mood(1, indicative)
tense(1, present)
subj(0, 2)
pred(2, pro)
pers(2, 3)
anm(2, '-')
gend(2, neut)
case(2, nom)
pron_type(2, pers)
nom(2, sg)
```


Typically the grammar begins with a bunch of deletion rules. These rules primarily delete language specific terms which are not supposed to affect the translation relation.

```
aux_select(_,_) ==> 0.
status(_,_)   ==> 0.
gend(_,_)     ==> 0.
prog(_,_)     ==> 0.
anim(_,_)     ==> 0.
ntype(_,_)    ==> 0.
grain(_,_)    ==> 0.
proper(_,_)   ==> 0.
```


Simple rules to transfer specific words or attributes, look like:

```
vtype(X,main) ==> vtype(X,unspec).
pred(X, be)   ==> pred(X, être).
```

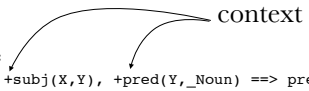

Repetitive transfer rules can be encoded using templates. A template can be defined as any prolog term, including any of the operators provided by prolog (such as '->' or '-'). Templates can call other templates and also combine with other templates and rules.

```
p2p(Source, Target) ::
   pred(X,Source) ==> pred(X,Target).

Source -> Target ::
   p2p(Source, Target).

dog    -> chien.
cat    -> chat.
saw    -> voyait.

sleep  -> dormir.
run    -> courir.
come   -> venir.
walk   -> marcher.
```

```
proper_name(Source, Target) ::
   pred(X,Source) ==> pred(X,Target).

proper_name('John', 'Jean').
proper_name('Mary', 'Marie').
```


Templates can also be redefined, which is especially useful for using the convenient and mnemonic operators.

```
Source -> Target ::
   spec_form(X,Source) ==> spec_form(X,Target).

the    -> le.

Source -> Target ::
   pron_form(X,Source) ==> pron_form(X,Target).

'I'    -> je.

Source -> Target ::
   pred(X,Source), +subj(X,Y), +pred(Y,_Noun) ==> pred(X,Target).

good   -> bon.
young  -> jeune.
left   -> gauche.
right  -> droit.
```



In addition to templates, the system provides a facility for referring to a group of recurring terms together, using macros.

```
add_adjunct(X,Z)      := adjunct(X,Y), in_set(Z,Y).
adjective(X,Adj,Subj) := pred(X,Adj), arg(X,1,Subj).

pred(X,girl) ==>
  pred(X,fille),            % girl -> jeune fille
  add_adjunct(X,J),
  adjective(J,jeune,X).
```

Semantics is a set of predicates

man(m), dog(d), boy(b), def(m), def(d), see(s),
past(s), agent(s, m), patient(s, d), recipient(s, b)

b, d, m, s are arbitrary constants.

Grammar:

s(s) → np(subj), vp(s, subj)

vp(v,subj) → v(v, subj, obj), np(obj)

vp(subj, obj2) → v(v, subj, obj1, obj2), np(obj1)

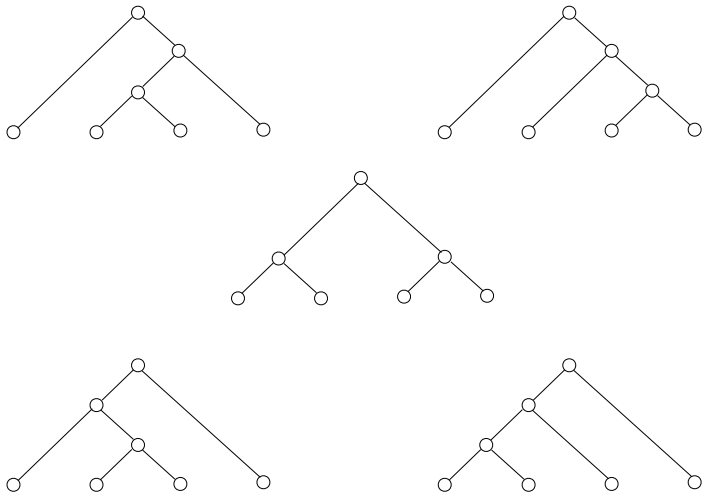
np(x) → det(x), n(x)

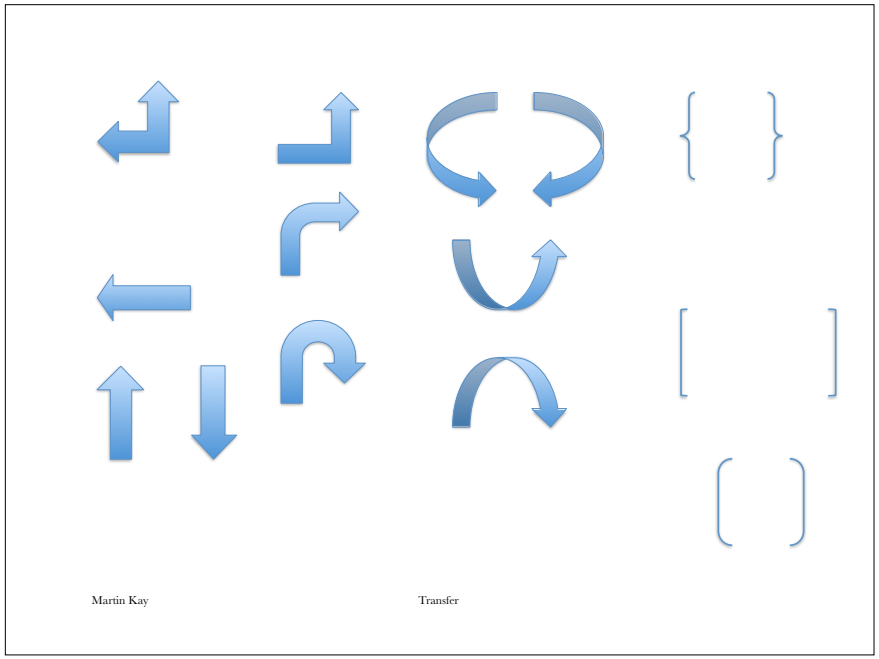
Word	Cat	Semantics
man	n(*x*)	*x*: man(*x*)
boy	v(*x*, *y*, *z*)	*x*: boy(*x*)
dog	n(*x*)	*x*: dog(*x*)
the	det(*x*)	*x*: def(*x*)
gave	v(*v*, *x*, *y*, *z*)	give(*v*), past(*v*), agent(*v*, *x*), patient(*v*, *y*), recipient(*v*, *z*)

- **Semantics is a set of predicates.**
- **Predicates are related by variables**

Word	Cat	Semantics
man	n(*x*)	*x*: man(*x*)
boy	v(*x*, *y*, *z*)	*x*: boy(*x*)
dog	n(*x*)	*x*: dog(*x*)
the	det(*x*)	*x*: def(*x*)
gave	v(*v*, *x*, *y*, *z*)	give(*v*), past(*v*), agent(*v*, *x*), patient(*v*, *y*), recipient(*v*, *z*)

- **Variables connect syntax to semantics**





Martin Kay

Transfer