Defectiveness: challenges for bottom-up lexical description

Roger Evans
Natural Language Technology Group
University of Brighton

10th April 2008

Overview

- Introduction

- Defectiveness
- Formal/Computational Linguistics and Defectiveness

- Encoding defectiveness – 3 approaches
- Implications for 3 lexical architectures
- Conclusions

Introduction

- Explore defectiveness from the perspective of lexical description systems and architectures
- Aiming for a better understanding of issues and options – not theory building
- Formal/Computational Linguistics orientation

Defectiveness – What is it?

- “When some generative lexical process predicts a form, but the form does not (normally) occur”
- Status of such forms unclear – ungrammatical? (surprisingly) rare? bad style? incomprehensible?
- Defective forms are accessible in some sense, and can sometimes be coerced into use
- Idiosyncratic quality of defectiveness is unclear – you do get regular patterns.

Defectiveness – Where is it?

- Traditionally identified in inflectional paradigms
- Greg Stump's talk this morning distinguished two loci for defectiveness phenomena
- Defective-like behaviour also found in Binyanim (which are more derivational in quality)
- Maybe lurking in other places unnoticed, or under different names (eg valency alternations)?

Defectiveness – How is it encoded?

- Traditionally on the form feature/value itself (eg in a paradigm table cell)
- Again, Greg Stump's talk elaborated this a bit
- Explicit inference systems for describing lexical objects may support other options (as we shall see later)

A Formal/CL Perspective on Defectiveness

Three main activities:

- Natural Language Understanding (big)
- Natural Language Generation (small)
- Natural Language Description (grammars, lexicons and other resources) (small

A Formal/CL Perspective – NLU

- NLU largely ignores defectiveness
  - because it never arises?
  - because it never hurts to analyse it as non-defective?
- (Also largely ignores ungrammaticality – relying on very weak notions of grammar, or partial analysis, to achieve robustness)

A Formal/CL Perspective – NLG

- NLG largely ignores defectiveness
  - relies on semantic control to avoid it?
- BUT avoiding defectiveness would be a significant architectural change for many systems, which currently only look for lexical gaps at the semantic or lexeme level, not in the inflectional phase.
  - Thus revising ‘canning’ to ‘being able to’ could be a significant operation

A Formal/CL Perspective – Description

- Underpins the other two areas (to some extent)
- Limited demand has meant limited attention
- But that's what the rest of this talk is about…

Technical preliminaries – DATR

- Now let's get a bit more technical.
- First, a little bit of DATR …

```
VERB:
   <mor> == "<mor root>"
   <mor prp> == "<mor root>" ing
   <mor form> == <mor "<syn form>">
   <syn cat> == v
   <syn form> == 1sing
.
```

```
Must:
   <> == VERB
   <mor root> == must
.
Musting:
   <> == Must
   <syn form> == prp
.
```

```
VERB:
   <mor> == "<mor root>"
   <mor prp> == "<mor root>" ing
   <mor form> == <mor "<syn form>">
   <syn cat> == v
   <syn form> == 1sing
.
```

```
Must:
   <> == VERB
   <mor root> == must
.
Musting:
   <> == Must
   <syn form> == prp
.
```

```
Must:
   <syn cat> = v
   <syn form> = 1sing
   <mor root> = must
   <mor 1sing> = must
   <mor prp> == must ing
   <mor form> = must
```

```
Musting:
   <syn cat> = v
   <syn form> = prp
   <mor root> = must
   <mor 1sing> = must
   <mor prp> == must ing
   <mor form> = must ing
```

```
VERB:
   <mor> == "<mor root>"
   <mor prp> == "<mor root>" ing
   <mor form> == <mor "<syn form>">
   <syn cat> == v
   <syn form> == 1sing
.
```

```
Must:
   <> == VERB
   <mor root> == must
.
Musting:
   <> == Must
   <syn form> == prp
.
```

```
Must:  syn:  cat:    v
             form:   1sing
       mor:  root:   must
             1sing:  must
             prp:    must ing
             form:   must
```

```
Musting:  syn:  cat:    v
                form:   prp
          mor:  root:   must
                1sing:  must
                prp:    must ing
                form:   must ing
```

Encoding defectiveness – as ‘missing’

- What can be missing?
  - A feature value
  - An AVM
  - A whole lexical entry
- How is it achieved?
  - Failure to specify
  - Withdrawal of specification
  - Overriding of specification
- Where is it done?
  - Anywhere in the derivation of the missing item

Encoding defectiveness – as 'missing'

- What can be missing?
  - A feature value
  - An AVM
  - A whole lexical entry

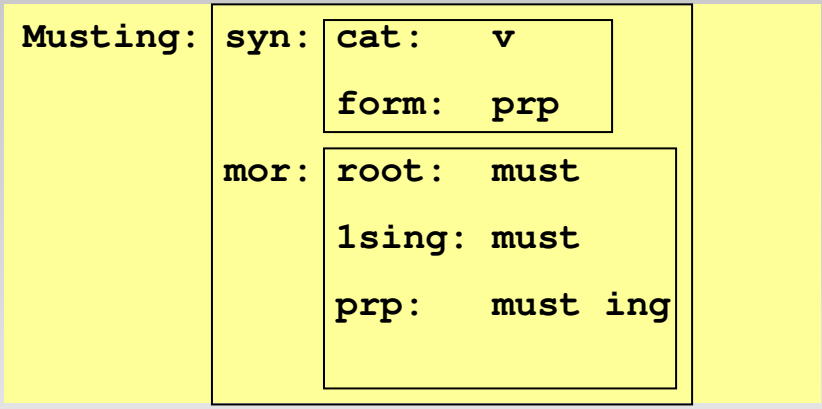
```
Musting:  syn:  cat:   v
                form:  prp
          mor:  root:  must
                1sing: must
                prp:   must ing
                form:  must ing
```

- How is it achieved?
  - Failure to specify
  - Withdrawal of specification
  - Overriding of specification

- Where is it done?
  - Anywhere in the derivation of the missing item

Encoding defectiveness – as 'missing'

- What can be missing?
  - A feature value
  - An AVM
  - A whole lexical entry

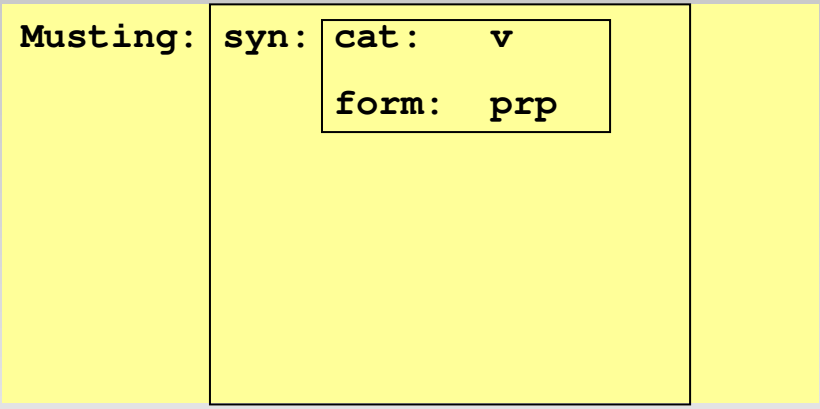


- How is it achieved?
  - Failure to specify
  - Withdrawal of specification
  - Overriding of specification

- Where is it done?
  - Anywhere in the derivation of the missing item

Encoding defectiveness – as 'missing'

- What can be missing?
  - A feature value
  - An AVM
  - A whole lexical entry



- How is it achieved?
  - Failure to specify
  - Withdrawal of specification
  - Overriding of specification

- Where is it done?
  - Anywhere in the derivation of the missing item

Encoding defectiveness – as ‘missing’

- What can be missing?
  - A feature value
  - An AVM
  - A whole lexical entry

- How is it achieved?
  - Failure to specify
  - Withdrawal of specification
  - Overriding of specification

- Where is it done?
  - Anywhere in the derivation of the missing item

Encoding defectiveness – as 'missing'

- What can be missing?
  - A feature value
  - An AVM
  - A whole lexical entry

```
Musting:  syn:  cat:   v
                form:  prp
          mor:  root:  must
                1sing: must
                prp:   must ing
                form:  must ing
```

- How is it achieved?
  - Failure to specify
  - Withdrawal of specification
  - Overriding of specification

- Where is it done?
  - Anywhere in the derivation of the missing item

```
VERB:
   <mor> == "<mor root>"
   <mor prp> == "<mor root>" ing
   <mor form> == <mor "<syn form>">
   <syn cat> == v
   <syn form> == 1sing
.
```

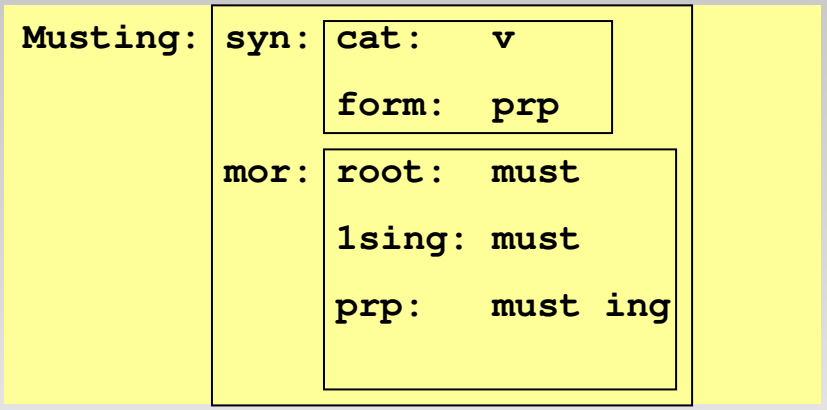
eness – as 'missing'

- An AVM
- A whole lexical entry

```
Must:
   <> == VERB
   <mor root> == must
.
Musting:
   <> == Must
   <syn form> == prp
   <mor form> == UNDEF
.
```

n

- Anywhere in the derivation of the missing item

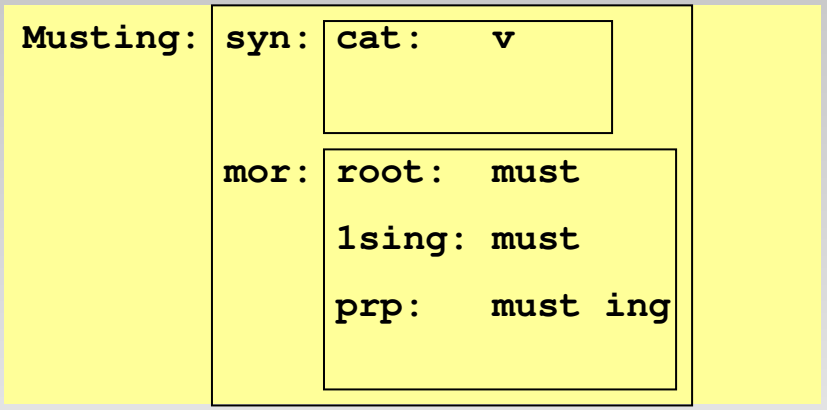


eness – as 'missing'

```
VERB:
   <mor> == "<mor root>"
   <mor prp> == "<mor root>" ing
   <mor form> == <mor "<syn form>">
   <syn cat> == v
   <syn form> == 1sing
.
```

- An AVM
- A whole lexical entry

```
Must:
   <> == VERB
   <mor root> == must
.
Musting:
   <> == Must
   <syn form> == UNDEF
.
```

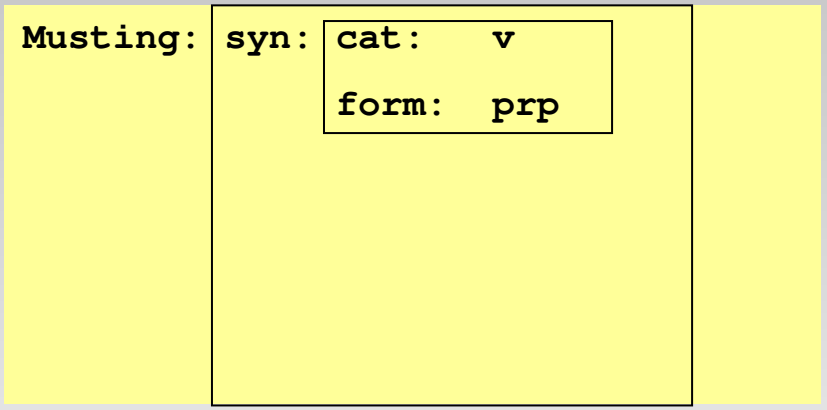


n

- Where is it done?
  - Anywhere in the derivation of the missing item

eness – as 'missing'

```
VERB:
   <mor> == "<mor root>"
   <mor prp> == "<mor root>" ing
   <mor form> == <mor "<syn form>">
   <syn cat> == v
   <syn form> == 1sing
.
```



- An AVM
- A whole lexical entry

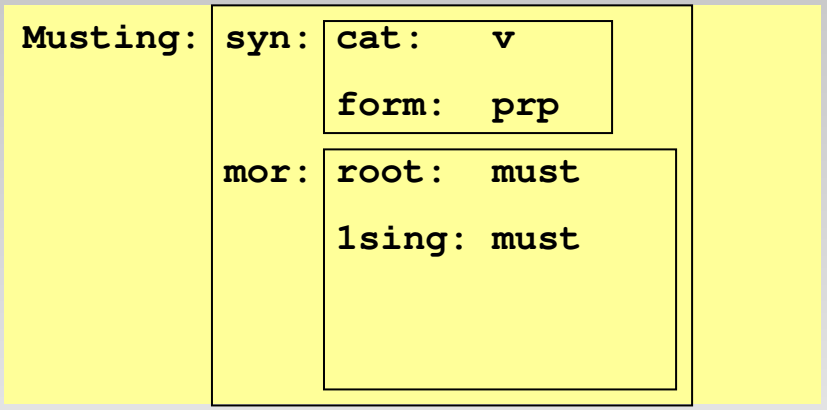
```
Must:
   <> == VERB
   <mor root> == must
.
Musting:
   <> == Must
   <syn form> == prp
   <mor> == UNDEF
.
```

n

- Anywhere in the derivation of the missing item

eness – as 'missing'

```
VERB:
   <mor> == "<mor root>"
   <mor prp> == "<mor root>" ing
   <mor form> == <mor "<syn form>">
   <syn cat> == v
   <syn form> == 1sing
.
```



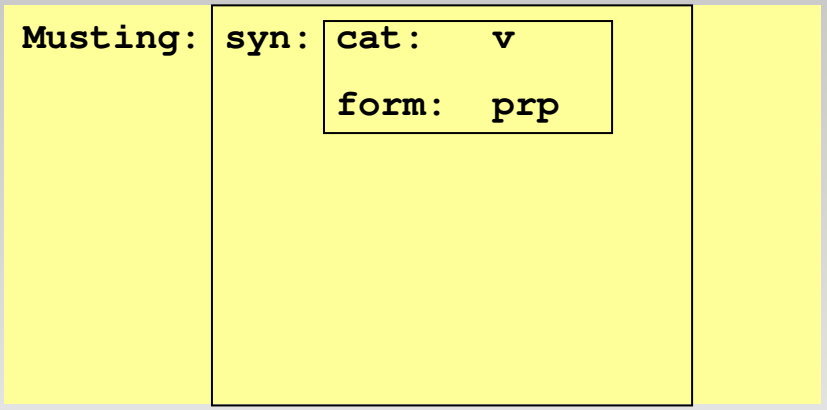
- An AVM
- A whole lexical entry

```
Must:
   <> == VERB
   <mor root> == must
.
Musting:
   <> == Must
   <syn form> == prp
   <mor prp> == UNDEF
.
```

- Anywhere in the derivation of the missing item

eness – as 'missing'

```
VERB:
   <mor> == "<mor root>"
   <mor prp> == "<mor root>" ing
   <mor form> == <mor "<syn form>">
   <syn cat> == v
   <syn form> == 1sing
.
```



- An AVM
- A whole lexical entry

```
Must:
   <> == VERB
   <mor root> == must
.
Musting:
   <> == Must
   <syn form> == prp
   <mor root> == UNDEF
.
```

n

- Anywhere in the derivation of the missing item

Encoding defectiveness – as ‘missing’

- Implications for defectiveness:
  - Flexible approach – a range of ways to achieve required result
  - Missing information really is missing – it can’t really be recovered (without ‘inside knowledge’ of the descriptions)
  - Defectiveness marked like this definitely presents as ‘ungrammatical’

Encoding defectiveness – with a feature

- What can be marked?
  - An AVM
  - A whole lexical entry
- How is it achieved?
  - By adding a <defective> == true statement at an appropriate point
- Where is it done?
  - At the point where the value is specified (or above it in an inheritance heirachy)

```
VERB:
   <mor> == "<mor root>"
   <mor prp> == "<mor root>" ing
   <mor form> == <mor "<syn form>">
   <syn cat> == v
   <syn form> == 1sing
.
```

eness – with a feature

?

– A whole lexical entry

```
Must:
   <> == VERB
   <mor root> == must
.
Musting:
   <> == Must
   <syn form> == prp
   <defective> == true
.
```

```
Musting: syn:  cat:    v
               form:   prp
         mor:  root:   must
               1sing:  must
               prp:    must ing
               form:   must ing
         defective: true
```

e> == true statement at an

e value is specified (or above it in an inheritance hierarchy)

```
VERB:
   <mor> == "<mor root>"
   <mor prp> == "<mor root>" ing
   <mor form> == <mor "<syn form>">
   <syn cat> == v
   <syn form> == 1sing
.
```

eness – with a feature

?

– A whole lexical entry

```
Must:
   <> == VERB
   <mor root> == must
.
Musting:
   <> == Must
   <syn form> == prp
   <mor defective> == true
.
```

```
Musting:  syn:  cat:    v
                form:   prp
          mor:  root:   must
                1sing:  must
                prp:    must ing
                form:   must ing
                defective: true
```

e> == true statement at an

e value is specified (or above it in an inheritance hierarchy)

Encoding defectiveness – with a feature

- Implications for defectiveness:
  - Introduces additional features ‘in parallel with’ defective forms – a bit cumbersome, and dependent on appropriate external interpretation
  - Side-steps ungrammaticality problem – you can choose whether to take account of this feature or not
  - Mixes up words and nonwords in the lexicon

Encoding defectiveness – with a marked value

- What can be marked?
  - A value
- How is it achieved?
  - By allowing values to take a +defective property (as part of their internal structure)
  - By propagating the +defective property through value derivation sequences (cf tainted variables in Perl)
- Where is it done?
  - Anywhere in the derivation of the missing item

```
VERB:
  <mor> == "<mor root>"
  <mor prp> == "<mor root>" ing
  <mor form> == <mor "<syn form>">
  <syn cat> == v
  <syn form> == 1sing
.
```

ess – with a marked value

```
Musting: syn: cat:    v
              form:   prp
         mor: root:   must
              1sing:  must
              prp:    must ing
              form:   must ing
```

- How is it achieved?

```
Must:
  <> == VERB
  <mor root> == must
.
Musting:
  <> == Must
  <syn form> == prp
.
```

ake a +defective property (as part
re)

efective property through value
cf tainted variables in Perl)

- Where is it done?
  - Anywhere in the derivation of the missing item

```
VERB:
  <mor> == "<mor root>"
  <mor prp> == "<mor root>" ing
  <mor form> == <mor "<syn form>">
  <syn cat> == v
  <syn form> == 1sing
.
```

ess – with a marked value

```
Musting: syn: cat:   v
              form:  prp
         mor: root:  must
              1sing: must
              prp:   must ing
              form:  must ing
```

- How is it achieved?

```
Must:
  <> == VERB
  <mor root> == must
.
Musting:
  <> == Must
  <syn form> == prp
  <mor root> == must
.
```

ake a +defective property (as part
re)

efective property through value
cf tainted variables in Perl)

  - Anywhere in the derivation of the missing item

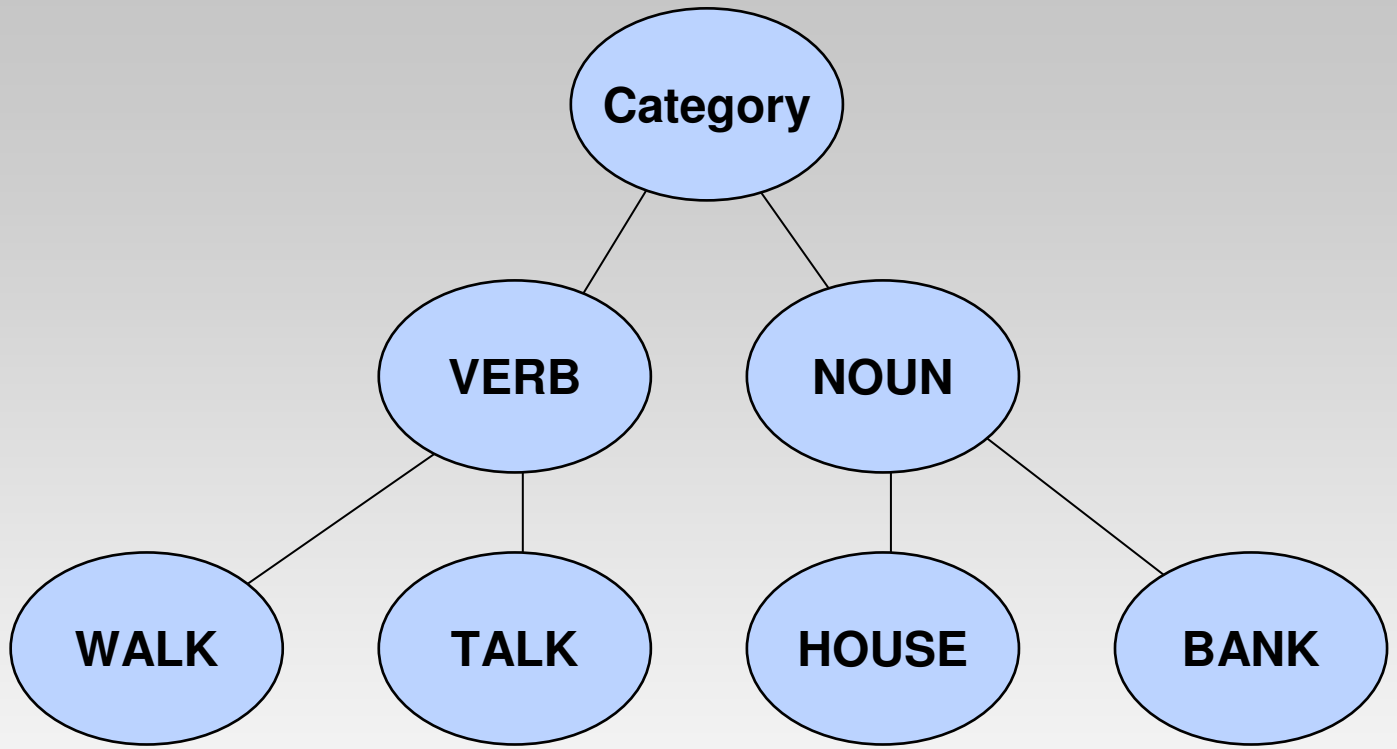
Encoding defectiveness – with a marked value

- Implications for defectiveness:
  - A compromise between the other two approaches
  - Requires an extension to the framework (but not unlike adding probabilities to values)
  - Delivers a flexible approach
  - Side-steps ungrammaticality problem – you can choose whether to take account of this feature or not
  - Mixes up words and nonwords in the lexicon

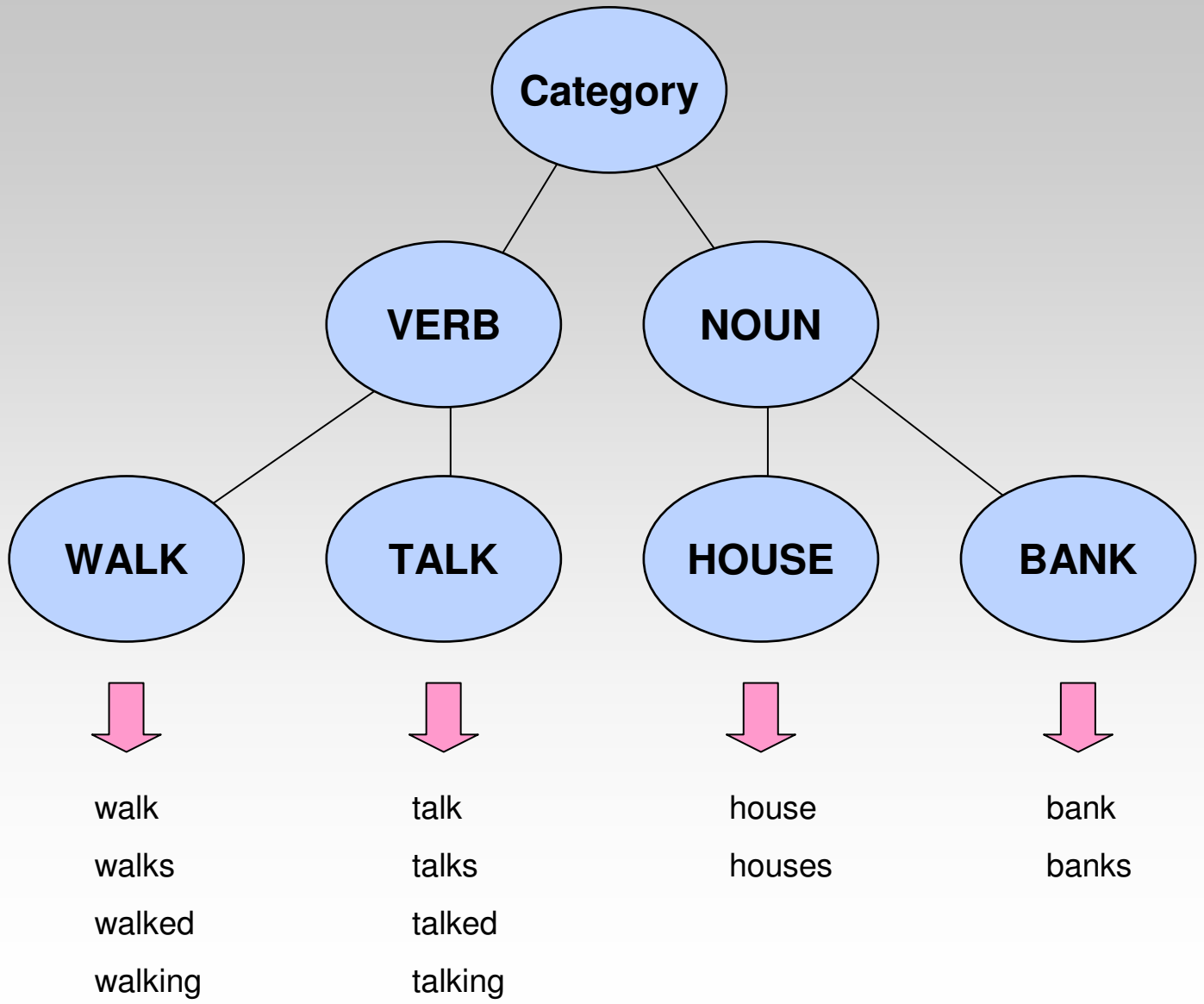
Lexical architectures – lexeme-based

- Objects described are lexemes
- Each contains a paradigm table of forms
- Description may also include abstract nodes capturing generalisations

Lexeme-based lexicon



Lexeme-based lexicon



Lexical architectures – lexeme-based

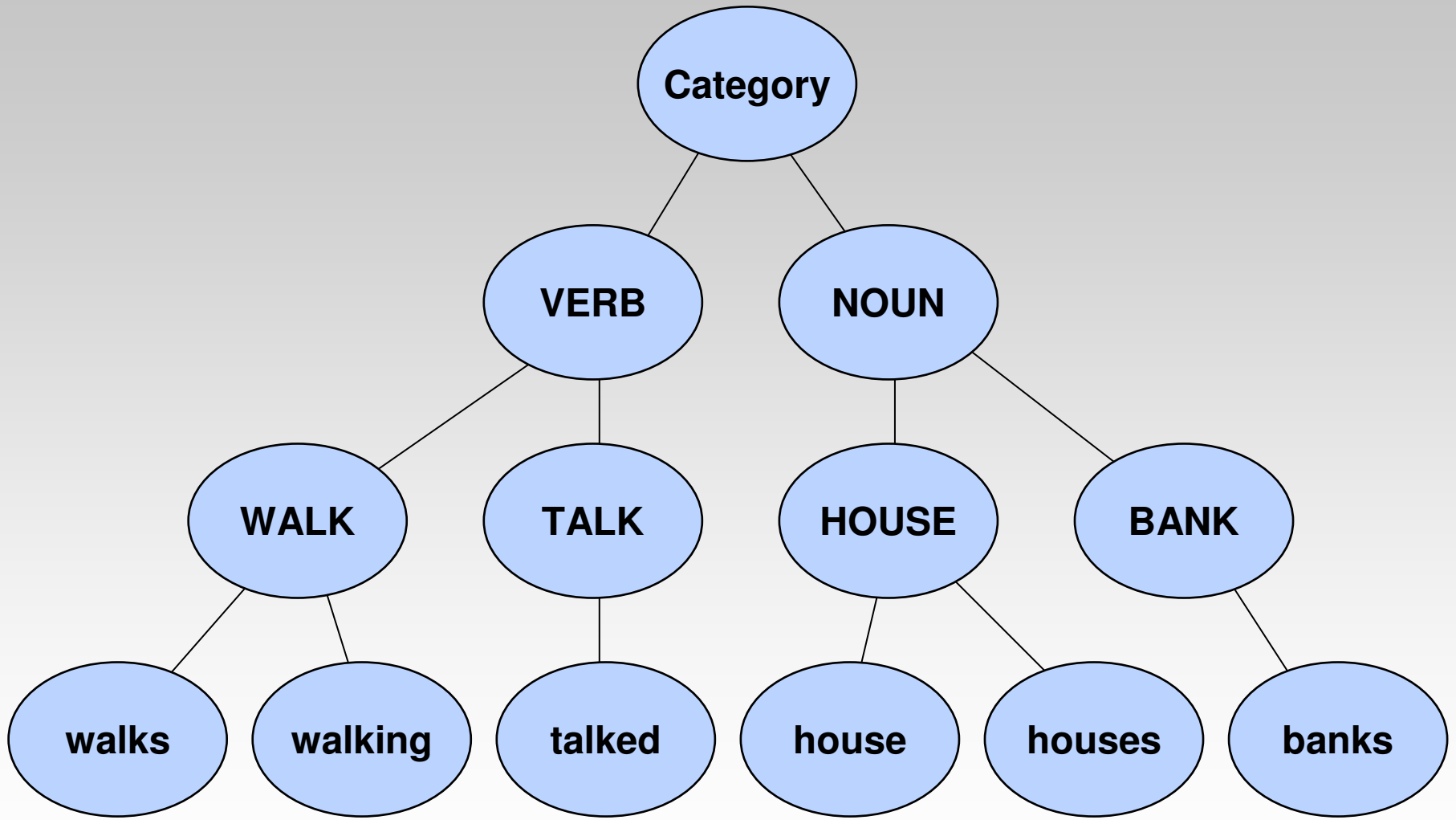
- Objects described are lexemes
- Each contains a paradigm table of forms
- Description may also include abstract nodes capturing generalisations

- Suitable for encoding defectiveness (all models)
- But not great for non-morphological extension (eg syntax) that cares more about wordforms

Lexical architectures – wordform-based

- Objects described are wordforms
- Generalisations percolate up from wordform nodes to abstract nodes
- Abstract nodes ***may*** look like lexemes
- Abstract nodes ***may*** contain paradigmatic generalisations

Wordform-based lexicon



Lexical architectures – wordform-based

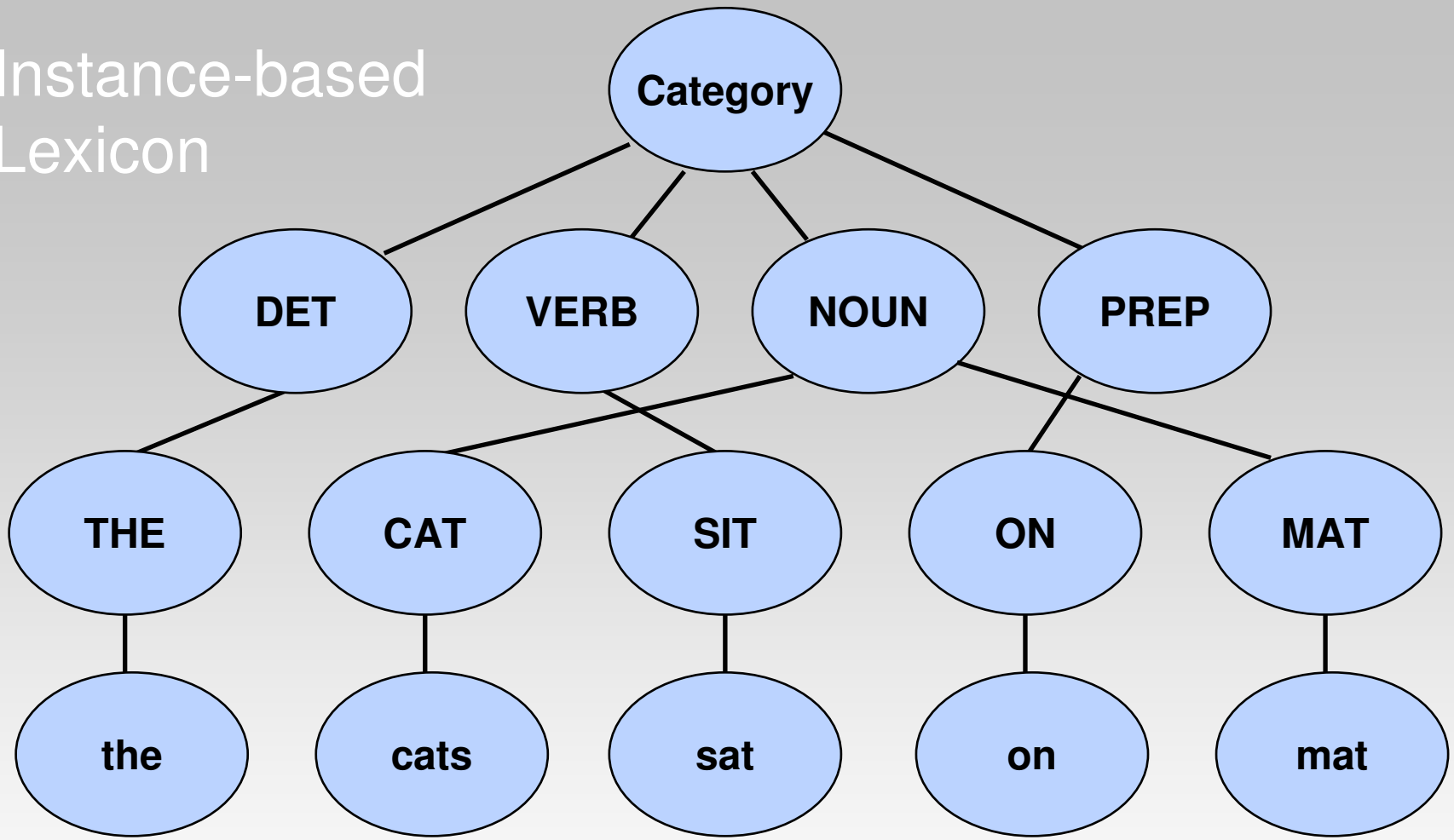
- Objects described are wordforms
- Generalisations percolate up from wordform nodes to abstract nodes
- Abstract nodes ***may*** look like lexemes
- Abstract nodes ***may*** contain paradigmatic generalisations

- Bottom-up approach better for wordform-oriented specification
- Indexed on wordform string, so represents both words and nonwords

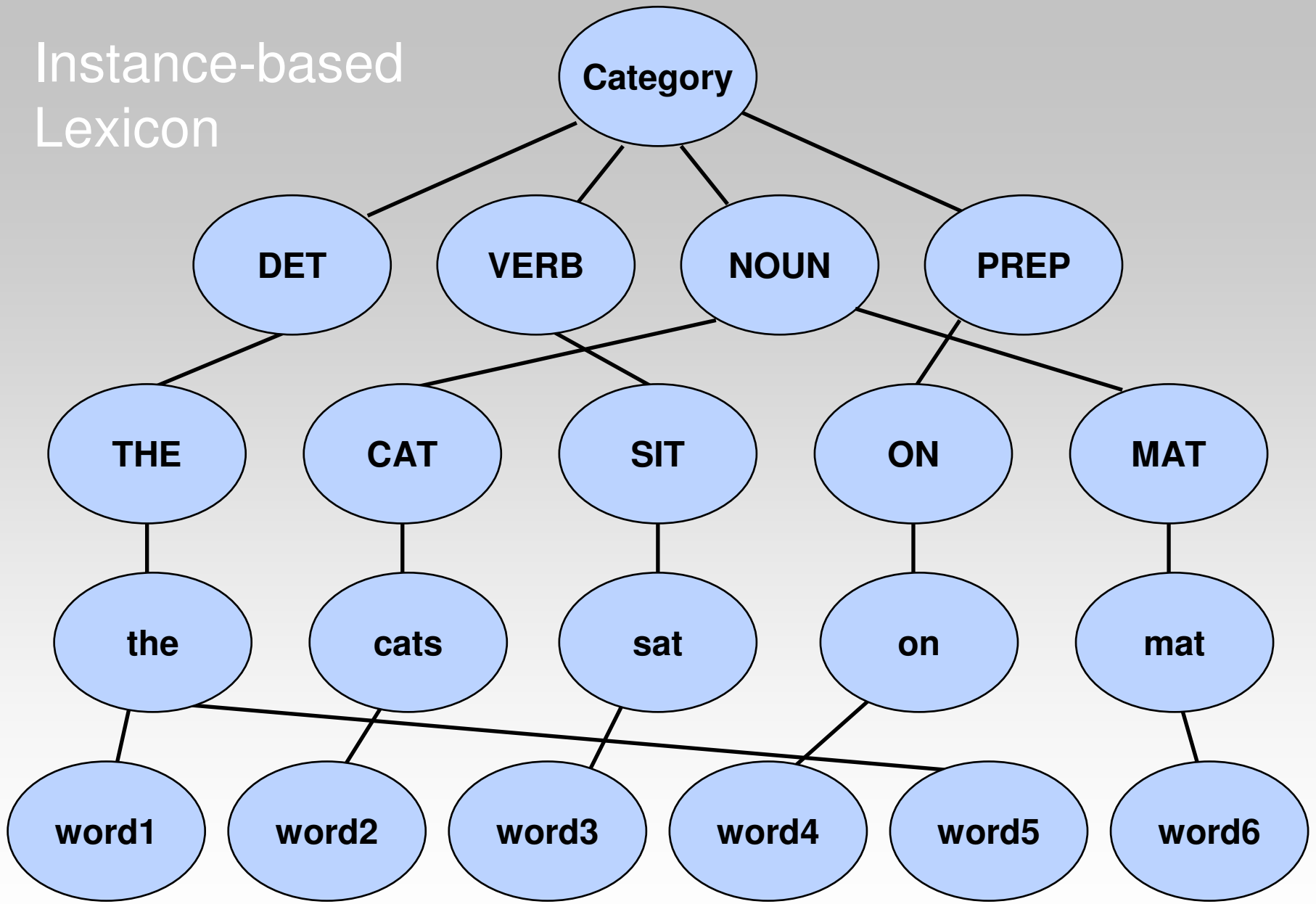
Lexical architectures – instance-based

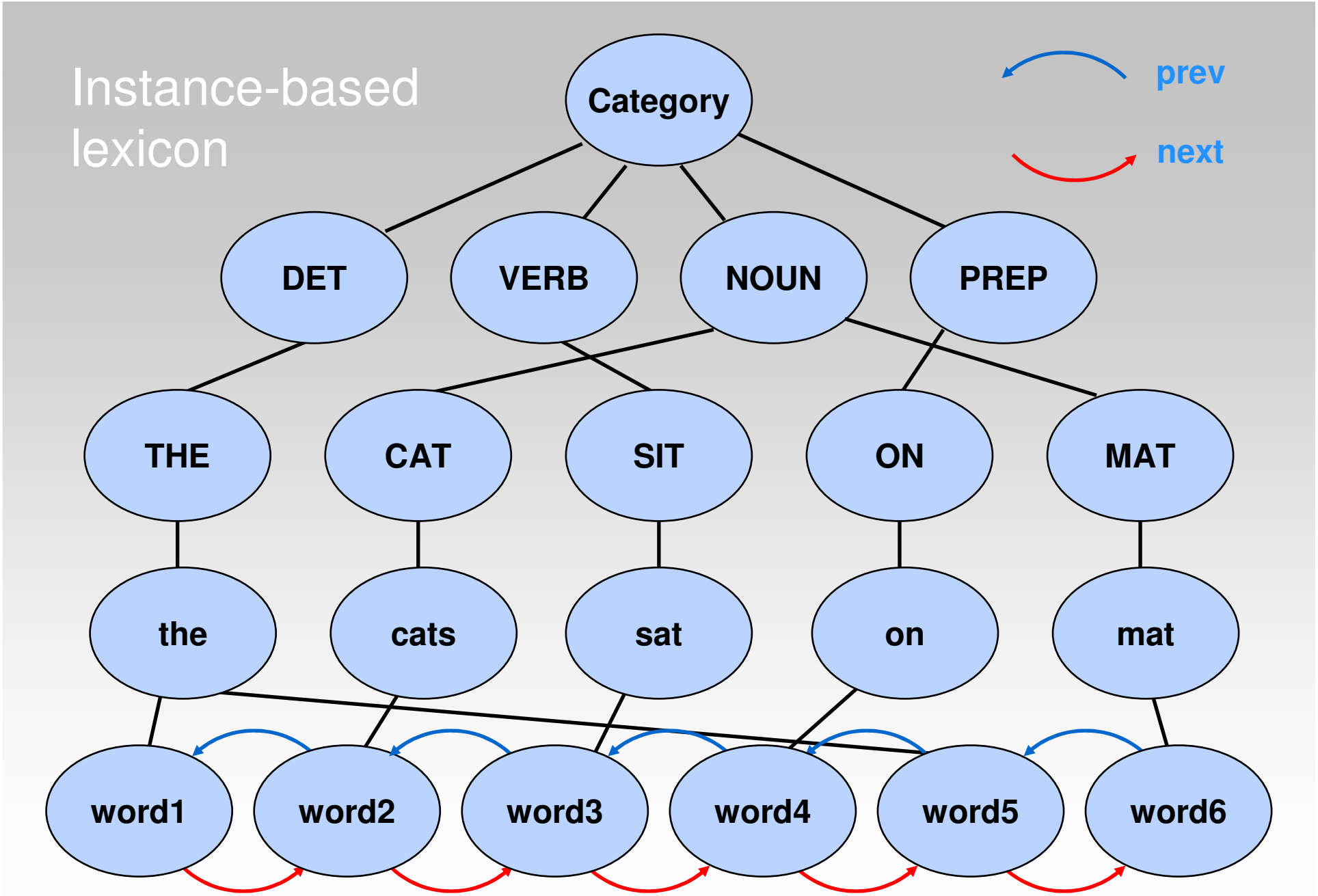
- Objects described are wordform instances
- Instances know about the words around them
- Can use this knowledge to select part-of-speech in context (for example)

Instance-based Lexicon

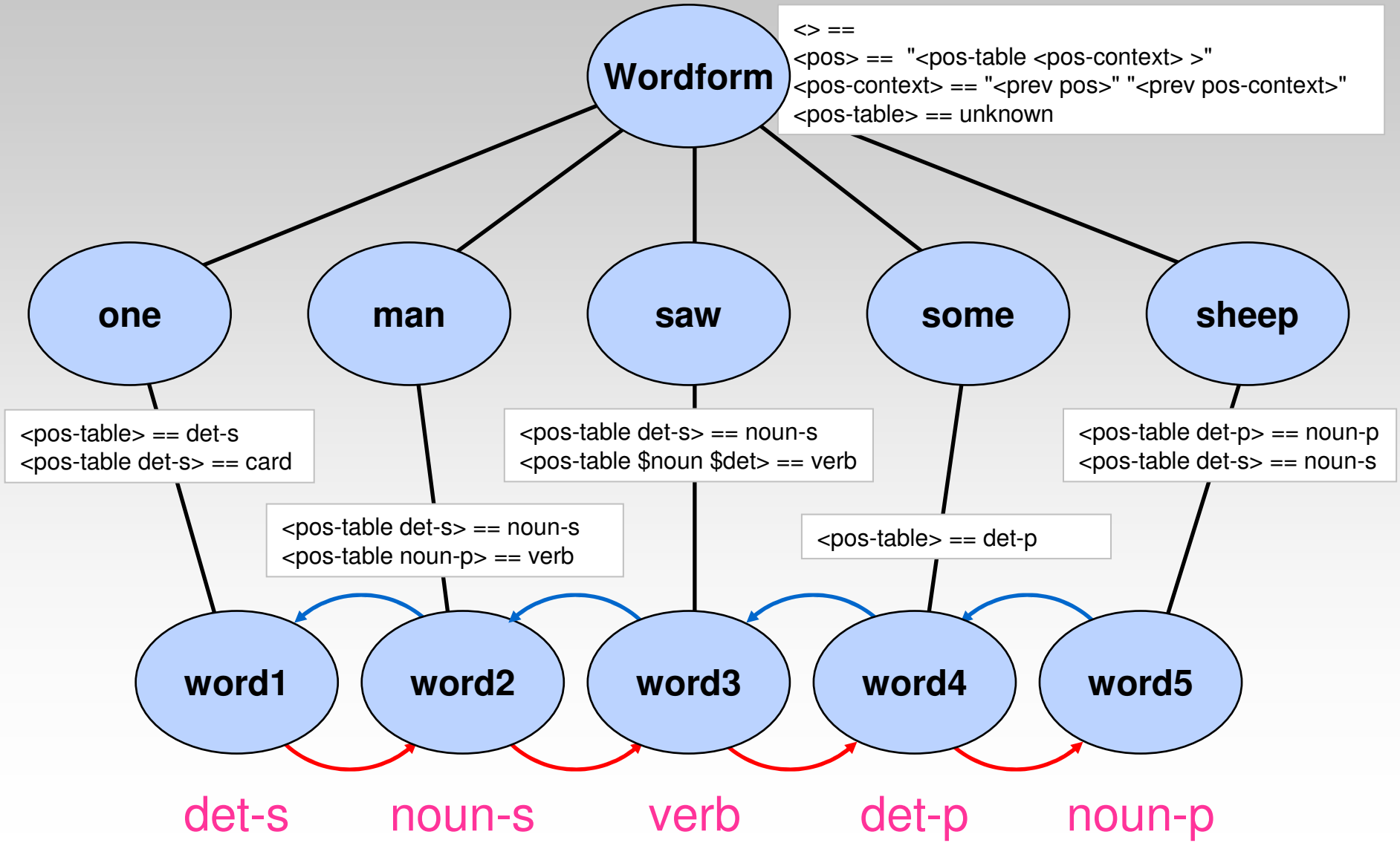


Instance-based Lexicon




Instance-based lexicon
Category
prev
next
DET
VERB
NOUN
PREP
THE
CAT
SIT
ON
MAT
the
cats
sat
on
mat
word1
word2
word3
word4
word5
word6

POS tagging – “one man saw some sheep”



Lexical architectures – instance-based

- Objects described are wordform instances
- Instances know about the words around them
- Can use this knowledge to select part-of-speech in context (for example)

- Fine-grained control over defectiveness behaviour in context
- Can support all defectiveness forms at the same time – selected by the sentential context

Conclusions

- Defectiveness is quite vague, but methods discussed here are quite general
- Interaction with ungrammaticality and word/nonword issues complicates things – and this is not a well-studied area in F/CL either.
- Looked at three approaches to representing defectiveness, each with pros and cons
- Looked at implications for three lexical architectures
- An instance-based marked value approach is perhaps the most interesting to pursue