Accessing and Analyzing Linguistic Field Data

Steven Bird

University of Melbourne, AUSTRALIA

May 1, 2007

Introduction

- variety of data types, e.g.
- **lexicon:** database of words, minimally containing part of speech information and glosses
- **paradigm:** any kind of rational tabulation of words or phrases to illustrate contrasts and systematic variation
- **text:** any larger unit such as a narrative or a conversation
- descriptions: field notes, grammars and analytical papers
- complex inter-relations (new word; lexicon update, paradigm; add to field notes; extend grammar, ...)
- tasks: sorting, tabulating, garnering statistics, searching, verifying transcriptions
- principle challenge is computational

Introduction

- variety of data types, e.g.
- **lexicon:** database of words, minimally containing part of speech information and glosses
- **paradigm:** any kind of rational tabulation of words or phrases to illustrate contrasts and systematic variation
- **text:** any larger unit such as a narrative or a conversation
- descriptions: field notes, grammars and analytical papers
- complex inter-relations (new word; lexicon update, paradigm; add to field notes; extend grammar, ...)
- tasks: sorting, tabulating, garnering statistics, searching, verifying transcriptions
- principle challenge is computational

Introduction

- variety of data types, e.g.
- **lexicon:** database of words, minimally containing part of speech information and glosses
- **paradigm:** any kind of rational tabulation of words or phrases to illustrate contrasts and systematic variation
- **text:** any larger unit such as a narrative or a conversation
- descriptions: field notes, grammars and analytical papers
- complex inter-relations (new word; lexicon update, paradigm; add to field notes; extend grammar, ...)
- tasks: sorting, tabulating, garnering statistics, searching, verifying transcriptions
- principle challenge is computational

Introduction

- variety of data types, e.g.
- **lexicon:** database of words, minimally containing part of speech information and glosses
- **paradigm:** any kind of rational tabulation of words or phrases to illustrate contrasts and systematic variation
- **text:** any larger unit such as a narrative or a conversation
- descriptions: field notes, grammars and analytical papers
- complex inter-relations (new word; lexicon update, paradigm; add to field notes; extend grammar, ...)
- tasks: sorting, tabulating, garnering statistics, searching, verifying transcriptions
- principle challenge is computational

Introduction

- variety of data types, e.g.
- **lexicon:** database of words, minimally containing part of speech information and glosses
- **paradigm:** any kind of rational tabulation of words or phrases to illustrate contrasts and systematic variation
- **text:** any larger unit such as a narrative or a conversation
- descriptions: field notes, grammars and analytical papers
- complex inter-relations (new word; lexicon update, paradigm; add to field notes; extend grammar, ...)
- tasks: sorting, tabulating, garnering statistics, searching, verifying transcriptions
- principle challenge is computational

Introduction

- variety of data types, e.g.
- **lexicon:** database of words, minimally containing part of speech information and glosses
- **paradigm:** any kind of rational tabulation of words or phrases to illustrate contrasts and systematic variation
- **text:** any larger unit such as a narrative or a conversation
- descriptions: field notes, grammars and analytical papers
- complex inter-relations (new word; lexicon update, paradigm; add to field notes; extend grammar, ...)
- tasks: sorting, tabulating, garnering statistics, searching, verifying transcriptions
- principle challenge is computational

Introduction

- variety of data types, e.g.
- **lexicon:** database of words, minimally containing part of speech information and glosses
- **paradigm:** any kind of rational tabulation of words or phrases to illustrate contrasts and systematic variation
- **text:** any larger unit such as a narrative or a conversation
- descriptions: field notes, grammars and analytical papers
- complex inter-relations (new word; lexicon update, paradigm; add to field notes; extend grammar, ...)
- tasks: sorting, tabulating, garnering statistics, searching, verifying transcriptions
- principle challenge is computational

Introduction

- variety of data types, e.g.
- **lexicon:** database of words, minimally containing part of speech information and glosses
- **paradigm:** any kind of rational tabulation of words or phrases to illustrate contrasts and systematic variation
- **text:** any larger unit such as a narrative or a conversation
- descriptions: field notes, grammars and analytical papers
- complex inter-relations (new word; lexicon update, paradigm; add to field notes; extend grammar, ...)
- tasks: sorting, tabulating, garnering statistics, searching, verifying transcriptions
- principle challenge is computational

Tools and Technologies

- General-purpose tools
  - word processors
  - spreadsheets
  - databases
- Special purpose tools
  - Toolbox
- broader question: data management in support of best practices
- Bird, Steven and Gary Simons (2003). Seven Dimensions of Portability for Language Documentation and Description, *Language* 79: 557-582.

Tools and Technologies

- General-purpose tools
  - word processors
  - spreadsheets
  - databases
- Special purpose tools
  - Toolbox
- broader question: data management in support of best practices
- Bird, Steven and Gary Simons (2003). Seven Dimensions of Portability for Language Documentation and Description, *Language* 79: 557-582.

Tools and Technologies

- General-purpose tools
  - word processors
  - spreadsheets
  - databases
- Special purpose tools
  - Toolbox
- broader question: data management in support of best practices
- Bird, Steven and Gary Simons (2003). Seven Dimensions of Portability for Language Documentation and Description, *Language* 79: 557-582.

Tools and Technologies

- General-purpose tools
  - word processors
  - spreadsheets
  - databases
- Special purpose tools
  - Toolbox
- broader question: data management in support of best practices
- Bird, Steven and Gary Simons (2003). Seven Dimensions of Portability for Language Documentation and Description, *Language* 79: 557-582.

Tools and Technologies

- General-purpose tools
  - word processors
  - spreadsheets
  - databases
- Special purpose tools
  - Toolbox
- broader question: data management in support of best practices
- Bird, Steven and Gary Simons (2003). Seven Dimensions of Portability for Language Documentation and Description, *Language* 79: 557-582.

Tools and Technologies

- General-purpose tools
  - word processors
  - spreadsheets
  - databases
- Special purpose tools
  - Toolbox
- broader question: data management in support of best practices
- Bird, Steven and Gary Simons (2003). Seven Dimensions of Portability for Language Documentation and Description, *Language* 79: 557-582.

Tools and Technologies

- General-purpose tools
  - word processors
  - spreadsheets
  - databases
- Special purpose tools
  - Toolbox
- broader question: data management in support of best practices
- Bird, Steven and Gary Simons (2003). Seven Dimensions of Portability for Language Documentation and Description, *Language* 79: 557-582.

Tools and Technologies

- General-purpose tools
  - word processors
  - spreadsheets
  - databases
- Special purpose tools
  - Toolbox
- broader question: data management in support of best practices
- Bird, Steven and Gary Simons (2003). Seven Dimensions of Portability for Language Documentation and Description, *Language* 79: 557-582.

General Purpose Tools: Word Processors

- office software widely used in computer-based language documentation work
- familiar, ready availability (though may be too expensive?)
- Word processors for managing dictionaries:
  - consistency of content and format — time consuming
  - how would we check the following constraint:
    each entry must have a part-of-speech field, drawn from a set of 20 possibilities, displayed after the pronunciation field, and rendered in 11-point bold
  - generally a bad idea!
- save to a non-proprietary format (e.g. RTF, HTML, or XML)
- write programs to do this checking automatically

General Purpose Tools: Word Processors

- office software widely used in computer-based language documentation work
- familiar, ready availability (though may be too expensive?)
- Word processors for managing dictionaries:
  - consistency of content and format — time consuming
  - how would we check the following constraint:
    each entry must have a part-of-speech field, drawn from a set of 20 possibilities, displayed after the pronunciation field, and rendered in 11-point bold
  - generally a bad idea!
- save to a non-proprietary format (e.g. RTF, HTML, or XML)
- write programs to do this checking automatically

General Purpose Tools: Word Processors

- office software widely used in computer-based language documentation work
- familiar, ready availability (though may be too expensive?)
- Word processors for managing dictionaries:
  - consistency of content and format — time consuming
  - how would we check the following constraint:
    each entry must have a part-of-speech field, drawn from a set of 20 possibilities, displayed after the pronunciation field, and rendered in 11-point bold
  - generally a bad idea!
- save to a non-proprietary format (e.g. RTF, HTML, or XML)
- write programs to do this checking automatically

General Purpose Tools: Word Processors

- office software widely used in computer-based language documentation work
- familiar, ready availability (though may be too expensive?)
- Word processors for managing dictionaries:
  - consistency of content and format — time consuming
  - how would we check the following constraint:
    each entry must have a part-of-speech field, drawn from a set of 20 possibilities, displayed after the pronunciation field, and rendered in 11-point bold
  - generally a bad idea!
- save to a non-proprietary format (e.g. RTF, HTML, or XML)
- write programs to do this checking automatically

General Purpose Tools: Word Processors

- office software widely used in computer-based language documentation work
- familiar, ready availability (though may be too expensive?)
- Word processors for managing dictionaries:
  - consistency of content and format — time consuming
  - how would we check the following constraint:
    each entry must have a part-of-speech field, drawn from a set of 20 possibilities, displayed after the pronunciation field, and rendered in 11-point bold
  - generally a bad idea!
- save to a non-proprietary format (e.g. RTF, HTML, or XML)
- write programs to do this checking automatically

General Purpose Tools: Word Processors

- office software widely used in computer-based language documentation work
- familiar, ready availability (though may be too expensive?)
- Word processors for managing dictionaries:
  - consistency of content and format — time consuming
  - how would we check the following constraint:
    each entry must have a part-of-speech field, drawn from a set of 20 possibilities, displayed after the pronunciation field, and rendered in 11-point bold
  - generally a bad idea!
- save to a non-proprietary format (e.g. RTF, HTML, or XML)
- write programs to do this checking automatically

General Purpose Tools: Word Processors

- office software widely used in computer-based language documentation work
- familiar, ready availability (though may be too expensive?)
- Word processors for managing dictionaries:
  - consistency of content and format — time consuming
  - how would we check the following constraint:
    each entry must have a part-of-speech field, drawn from a set of 20 possibilities, displayed after the pronunciation field, and rendered in 11-point bold
  - generally a bad idea!
- save to a non-proprietary format (e.g. RTF, HTML, or XML)
- write programs to do this checking automatically

General Purpose Tools: Word Processors

- office software widely used in computer-based language documentation work
- familiar, ready availability (though may be too expensive?)
- Word processors for managing dictionaries:
  - consistency of content and format — time consuming
  - how would we check the following constraint:
    each entry must have a part-of-speech field, drawn from a set of 20 possibilities, displayed after the pronunciation field, and rendered in 11-point bold
  - generally a bad idea!
- save to a non-proprietary format (e.g. RTF, HTML, or XML)
- write programs to do this checking automatically

Example: Microsoft Word

- sleep [sli:p] **vi** *condition of body and mind...*
- We can enter this in MSWord, then "Save as Web Page"
- resulting HTML:

```
<p class=MsoNormal>
  sleep
  <span style='mso-spacerun:yes'> </span>
  [<span class=SpellE>sli:p</span>]
  <span style='mso-spacerun:yes'> </span>
  <b><span style='font-size:11.0pt'>vi</span></b>
  <span style='mso-spacerun:yes'> </span>
  <i>a condition of body and mind ...</i>
</p>
```

- Observations:
  - entry: HTML paragraph using the <p> element

Example: Validating Dictionaries stored in MSWord

```
>>> import re
>>> legal_pos = set(['n', 'v.t.', 'v.i.', 'adj', 'det'])
>>> pattern = re.compile(r"'font-size:11.0pt'>([a-z.]+)<")
>>> document = open("dict.htm").read()
>>> used_pos = set(re.findall(pattern, document))
>>> illegal_pos = used_pos.difference(legal_pos)
>>> print list(illegal_pos)
['v.intr', 'v.i', 'intrans']
```

Example: Converting MSWord dictionary to Excel

```
>>> import re
>>> document = open("dict.htm").read()
>>> document = re.sub("[\r\n]", "", document)
>>> word_pattern = re.compile(r">([\w]+)")
>>> pron_pattern = re.compile(r"\[.*>([a-z:]+)<.*\]")
>>> for entry in document.split("<p"):
...     word_match = word_pattern.search(entry)
...     pron_match = pron_pattern.search(entry)
...     if word_match and pron_match:
...         lex = word_match.group(1)
...         pos = pron_match.group(1)
...         print '"%s","%s"' % (lex, pos)
"sleep","sli:p"
"walk","wo:k"
"wake","weik"
```

Spreadsheets

- wordlists, paradigms
- e.g. comparative wordlist (`http://www.rosettaproject.org/`):
  - row for each cognate set
  - column for each language
- export spreadsheets in *CSV format*
- read them using Python's `csv` module
- e.g. find cognates having an edit-distance of at least three from each other

Spreadsheets

- wordlists, paradigms
- e.g. comparative wordlist (`http://www.rosettaproject.org/`):
  - row for each cognate set
  - column for each language
- export spreadsheets in *CSV format*
- read them using Python's `csv` module
- e.g. find cognates having an edit-distance of at least three from each other

Spreadsheets

- wordlists, paradigms
- e.g. comparative wordlist (`http://www.rosettaproject.org/`):
  - row for each cognate set
  - column for each language
- export spreadsheets in *CSV format*
- read them using Python's `csv` module
- e.g. find cognates having an edit-distance of at least three from each other

Spreadsheets

- wordlists, paradigms
- e.g. comparative wordlist (`http://www.rosettaproject.org/`):
  - row for each cognate set
  - column for each language
- export spreadsheets in *CSV format*
- read them using Python's `csv` module
- e.g. find cognates having an edit-distance of at least three from each other

Spreadsheets

- wordlists, paradigms
- e.g. comparative wordlist (`http://www.rosettaproject.org/`):
  - row for each cognate set
  - column for each language
- export spreadsheets in *CSV format*
- read them using Python's `csv` module
- e.g. find cognates having an edit-distance of at least three from each other

Spreadsheets

- wordlists, paradigms
- e.g. comparative wordlist (`http://www.rosettaproject.org/`):
  - row for each cognate set
  - column for each language
- export spreadsheets in *CSV format*
- read them using Python's `csv` module
- e.g. find cognates having an edit-distance of at least three from each other

Spreadsheets

- wordlists, paradigms
- e.g. comparative wordlist (`http://www.rosettaproject.org/`):
  - row for each cognate set
  - column for each language
- export spreadsheets in *CSV format*
- read them using Python's `csv` module
- e.g. find cognates having an edit-distance of at least three from each other

Databases

- lexicons are stored in a full-fledged relational databases
- databases can implement many well-formedness constraints
  e.g. ensure that all parts-of-speech come from an *enumerated type*
- However, the relational model is often too restrictive
- optional, repeatable fields (e.g. dictionary sense definitions and example sentences).
- SQL cannot express many linguistically-motivated queries
  Find all words that appear in example sentences for which no dictionary entry is provided

Databases

- lexicons are stored in a full-fledged relational databases
- databases can implement many well-formedness constraints
  e.g. ensure that all parts-of-speech come from an *enumerated type*
- However, the relational model is often too restrictive
- optional, repeatable fields (e.g. dictionary sense definitions and example sentences).
- SQL cannot express many linguistically-motivated queries
  Find all words that appear in example sentences for which no dictionary entry is provided

Databases

- lexicons are stored in a full-fledged relational databases
- databases can implement many well-formedness constraints
  e.g. ensure that all parts-of-speech come from an *enumerated type*
- However, the relational model is often too restrictive
- optional, repeatable fields (e.g. dictionary sense definitions and example sentences).
- SQL cannot express many linguistically-motivated queries
  Find all words that appear in example sentences for which no dictionary entry is provided

Databases

- lexicons are stored in a full-fledged relational databases
- databases can implement many well-formedness constraints
  e.g. ensure that all parts-of-speech come from an *enumerated type*
- However, the relational model is often too restrictive
- optional, repeatable fields (e.g. dictionary sense definitions and example sentences).
- SQL cannot express many linguistically-motivated queries
  Find all words that appear in example sentences for which no dictionary entry is provided

Databases

- lexicons are stored in a full-fledged relational databases
- databases can implement many well-formedness constraints
  e.g. ensure that all parts-of-speech come from an *enumerated type*
- However, the relational model is often too restrictive
- optional, repeatable fields (e.g. dictionary sense definitions and example sentences).
- SQL cannot express many linguistically-motivated queries
  Find all words that appear in example sentences for which no dictionary entry is provided

Database Example

```
>>> import csv
>>> lexemes = []
>>> defn_words = []
>>> for row in csv.reader(open("dict.csv")):
...     lexeme, pron, pos, defn = row
...     lexemes.append(lexeme)
...     defn_words += defn.split()
>>> undefined = list(set(defn_words).difference(set(lexemes)))
>>> undefined.sort()
>>> print undefined
['...', 'a', 'and', 'body', 'by', 'cease', 'condition', 'down',
'each', 'foot', 'lifting', 'mind', 'of', 'progress', 'setting', 'to']
```

Processing Toolbox Data

- single most popular tool for managing linguistic field data
- many kinds of validation and formatting not supported by Toolbox software
- simple file format
- each file is a collection of *entries* (aka *records*)
- each entry is made up of one or more *fields*
- challenges:
  1. different entry types, implications for which fields are present
  2. field order only sometimes significant

Processing Toolbox Data

- single most popular tool for managing linguistic field data
- many kinds of validation and formatting not supported by Toolbox software
- simple file format
- each file is a collection of *entries* (aka *records*)
- each entry is made up of one or more *fields*
- challenges:
  1. different entry types, implications for which fields are present
  2. field order only sometimes significant

Processing Toolbox Data

- single most popular tool for managing linguistic field data
- many kinds of validation and formatting not supported by Toolbox software
- simple file format
- each file is a collection of *entries* (aka *records*)
- each entry is made up of one or more *fields*
- challenges:
  1. different entry types, implications for which fields are present
  2. field order only sometimes significant

Processing Toolbox Data

- single most popular tool for managing linguistic field data
- many kinds of validation and formatting not supported by Toolbox software
- simple file format
- each file is a collection of *entries* (aka *records*)
- each entry is made up of one or more *fields*
- challenges:
  1. different entry types, implications for which fields are present
  2. field order only sometimes significant

Processing Toolbox Data

- single most popular tool for managing linguistic field data
- many kinds of validation and formatting not supported by Toolbox software
- simple file format
- each file is a collection of *entries* (aka *records*)
- each entry is made up of one or more *fields*
- challenges:
  1. different entry types, implications for which fields are present
  2. field order only sometimes significant

Processing Toolbox Data

- single most popular tool for managing linguistic field data
- many kinds of validation and formatting not supported by Toolbox software
- simple file format
- each file is a collection of *entries* (aka *records*)
- each entry is made up of one or more *fields*
- challenges:
  1. different entry types, implications for which fields are present
  2. field order only sometimes significant

Processing Toolbox Data

- single most popular tool for managing linguistic field data
- many kinds of validation and formatting not supported by Toolbox software
- simple file format
- each file is a collection of *entries* (aka *records*)
- each entry is made up of one or more *fields*
- challenges:
  1. different entry types, implications for which fields are present
  2. field order only sometimes significant

Processing Toolbox Data

- single most popular tool for managing linguistic field data
- many kinds of validation and formatting not supported by Toolbox software
- simple file format
- each file is a collection of *entries* (aka *records*)
- each entry is made up of one or more *fields*
- challenges:
  1. different entry types, implications for which fields are present
  2. field order only sometimes significant

Toolbox Example

```
\lx kaa
\ps N.M
\cl isi
\ge cooking banana
\gp banana bilong kukim
\sf FLORA
\dt 12/Feb/2005
\ex Taeavi iria kaa isi kovopaueva kaparapasia.
\xp Taeavi i bin planim gaden banana bilong kukim tasol
\xe Taeavi planted banana in order to cook it.
```

Accessing Toolbox Data

- scan the file, convert into tree object
- preserves order of fields, gives array and XPath-style access

```
>>> from nltk_lite.corpora import toolbox
>>> lexicon = toolbox.parse_corpus('rotokas.dic', key='l
```

Accessing with Indexes

```
>>> lexicon[3][0]
<Element lx at 77bd28>
>>> lexicon[3][0].tag
'lx'
>>> lexicon[3][0].text
'kaa'
```

Accessing with Indexes (cont)

```
>>> toolbox.to_sfm_string(lexicon[3])
\lx kaa
\ps N.M
\cl isi
\ge cooking banana
\gp banana bilong kukim
\sf FLORA
\dt 12/Feb/2005
\ex Taeavi iria kaa isi kovopaueva kaparapasia.
\xp Taeavi i bin planim gaden banana bilong kukim tasol long paia.
\xe Taeavi planted banana in order to cook it.
```

Accessing with Paths

- lexicon is a series of `record` objects
- each contains field objects, such as `lx` and `ps`
- address all the lexemes: `record/lx`

```
>>> [lexeme.text.lower() for lexeme in lexicon.findall('
['kaa', 'kaa', 'kaa', 'kaakaaro', 'kaakaaviko', 'kaakaav
'kaakasi', 'kaakau', 'kaakauko', 'kaakito', 'kaakuupato'
```

Adding New Fields

- Example: add CV field
- Aside: utility function to do CV template

```
>>> import re
>>> def cv(s):
...     s = s.lower()
...     s = re.sub(r'[^a-z]',     r'_', s)
...     s = re.sub(r'[aeiou]',    r'V', s)
...     s = re.sub(r'[^V_]',      r'C', s)
...     return (s)
```

Adding New Fields (cont)

```
>>> from nltk_lite.etree.ElementTree import SubElement
>>> for entry in lexicon:
...     for field in entry:
...     if field.tag == 'lx':
...         cv_field = SubElement(entry, 'cv')
...         cv_field.text = cv(field.text)
```

Adding New Fields (cont)

```
>>> toolbox.to_sfm_string(lexicon[50])
\lx kaeviro
\cv CVVCVCV
\ps V.A
\ge lift off
\ge take off
\gp go antap
\nt used to describe action of plane
\dt 12/Feb/2005
\ex Pita kaeviroroe kepa kekesia oa vuripierevo kiuvu.
\xp Pita i go antap na lukim haus win i bagarapim.
\xe Peter went to look at the house that the wind destroyed.
```

Removing Fields

- sanitise our data before giving it to others

```
>>> retain = ('lx', 'ps', 'ge')
>>> for entry in lexicon:
...     entry[:] = [f for f in entry if f.tag in retain]
>>> toolbox.to_sfm_string(lexicon[50])
\lx kaeviro
\ps V.A
\ge lift off
\ge take off
```

Formatting Entries

- request specific fields without concern for ordering

```
>>> lexicon = toolbox.parse_corpus('rotokas.dic', key='lx')
>>> for entry in lexicon[70:80]:
...     lx = entry.findtext('lx')
...     ps = entry.findtext('ps')
...     ge = entry.findtext('ge')
...     print "%s (%s) '%s'" % (lx, ps, ge)
kakapikoto (N.N2) 'newborn baby'
kakapu (V.B) 'place in sling for purpose of carrying'
kakapua (N.N) 'sling for lifting'
kakara (N.N) 'arm band'
Kakarapaia (N.PN) 'village name'
kakarau (N.F) 'frog'
Kakarera (N.PN) 'name'
Kakareraia (N.???) 'name'
kakata (N.F) 'cockatoo'
kakate (N.F) 'bamboo water tube'
```

Generating HTML Tables

- Publish shoebox lexicon on the web

```
>>> html = "<table>\n"
>>> for entry in lexicon[70:80]:
...     lx = entry.findtext('lx')
...     ps = entry.findtext('ps')
...     ge = entry.findtext('ge')
...     html += "  <tr><td>%s</td><td>%s</td><td>%s</td></tr>\n" %
...       (lx, ps, ge)
>>> html += "</table>"
>>> print html
```

```
<table>
  <tr><td>kakapikoto</td><td>N.N2</td><td>newborn baby</td></tr>
  <tr><td>kakapu</td><td>V.B</td><td>place in sling for purpose of carrying</td></tr>
  <tr><td>kakapua</td><td>N.N</td><td>sling for lifting</td></tr>
  <tr><td>kakara</td><td>N.N</td><td>bracelet</td></tr>
  <tr><td>Kakarapaia</td><td>N.PN</td><td>village name</td></tr>
  <tr><td>kakarau</td><td>N.F</td><td>stingray</td></tr>
  <tr><td>Kakarera</td><td>N.PN</td><td>name</td></tr>
  <tr><td>Kakareraia</td><td>N.???</td><td>name</td></tr>
  <tr><td>kakata</td><td>N.F</td><td>cockatoo</td></tr>
  <tr><td>kakate</td><td>N.F</td><td>bamboo tube for water</td></tr>
</table>
```

Generating XML

```
>>> import sys
>>> from nltk_lite.etree.ElementTree import ElementTree
>>> tree = ElementTree(lexicon[3])
>>> tree.write(sys.stdout)
<record>
  <lx>kaakaaro</lx>
  <ps>N.N</ps>
  <ge>mixtures</ge>
  <gp>???</gp>
  <eng>mixtures</eng>
  <eng>charm used to keep married men and women youthful and attractive</eng>
  <cmt>Check vowel length. Is it kaakaaro or kaakaro?</cmt>
  <dt>14/May/2005</dt>
  <ex>Kaakaroto ira purapaiveira aue iava opita, voeao-pa airepa oraouirara, ra va aiopaive.<
  <xp>Kokonas ol i save wokim long ol kain samting bilong ol nupela marit, bai ol i ken kaika
  <xe>Mixtures are made from coconut, ???.</xe>
</record>
```

Reduplication

- create a table of lexemes and their glosses

```
>>> lexgloss = {}
>>> for entry in lexicon:
...     lx = entry.findtext('lx')
...     if lx and entry.findtext('ps')[0] == 'V':
...         lexgloss[lx] = entry.findtext('ge')
```

- For each lexeme, check if the lexicon contains the reduplicated form:

```
>>> for lex in lexgloss:
...     if lex+lex in lexgloss:
...         print "%s (%s); %s (%s)" % (lex, lexgloss[lex], lex+le
```

Reduplication (cont)

```
kuvu (fill.up); kuvukuvu (stamp the ground)
kitu (save); kitukitu (scrub clothes)
kopa (ingest); kopakopa (gulp.down)
kasi (burn); kasikasi (angry)
koi (high pitched sound); koikoi (groan with pain)
kee (chip); keekee (shattered)
kauo (jump); kauokauo (jump up and down)
kea (deceived); keakea (lie)
kove (drop); kovekove (drip repeatedly)
kape (unable to meet); kapekape (grip with arms not meeting)
kapo (fasten.cover.strip); kapokapo (fasten.cover.strips)
koa (skin); koakoa (remove the skin)
kipu (paint); kipukipu (rub.on)
koe (spoon out a solid); koekoe (spoon out)
kovo (work); kovokovo (surround)
kiru (have sore near mouth); kirukiru (crisp)
kotu (bite); kotukotu (grind teeth together)
kavo (collect); kavokavo (work black magic)
...
```

Complex Search Criteria

- for phonological description, identify segments, alternations, syllable canon...
- what syllable types occur in lexemes (MSC, conspiracies)?

```
>>> from nltk_lite.tokenize import regexp
>>> from nltk_lite.probability import FreqDist
>>> fd = FreqDist()
>>> lexemes = [lexeme.text.lower() for lexeme in lexicon.findall('reco
>>> for lex in lexemes:
...     for syl in regexp(lex, pattern=r'[^aeiou][aeiou]'):
...         fd.inc(syl)
```

Complex Search Criteria (cont)

- Tabulate the results:

```
>>> for vowel in 'aeiou':
...     for cons in 'ptkvsr':
...         print '%s%s:%4d ' %
...             (cons, vowel, fd.count(cons+vowel)),
...     print
pa:  84  ta:  43  ka: 414  va:  87  sa:   0  ra: 185
pe:  32  te:   8  ke: 139  ve:  25  se:   1  re:  62
pi:  97  ti:   0  ki:  88  vi:  96  si:  95  ri:  83
po:  31  to: 140  ko: 403  vo:  42  so:   3  ro:  86
pu:  49  tu:  35  ku: 169  vu:  44  su:   1  ru:  72
```

- NB `t` and `s` columns
- `ti` not attested, while `si` is frequent: palatalization?
- which lexeme contains `su`? *kasuari*

Prosodically-motivated search

- segmental and prosodic constraints
- well-formed morphemes and lexemes
- highly-specific query (e.g. find things predicted not to exist by our theoretical model?)
- *Find all trisyllabic lexemes ending in a long vowel*
- bottom-up design: need to count syllables, and test for vowels

```
>>> def num_cons(word):
...     template = cv(word)
...     return template.count('C')
>>> def is_vowel(char):
...     return (char in 'aeiou')
```

Prosodically-motivated search (cont)

```
>>> for entry in lexicon:
...     lx = entry.findtext('lx')
...     if lx:
...         ps = entry.findtext('ps')
...         if num_cons(lx) == 3 and ps[0] == 'V'\
...           and is_vowel(lx[-1]) and is_vowel(lx[-2]):
...             ge = entry.findtext('ge')
...             print "%s (%s) '%s'" % (lx, ps, ge)
kaetupie (V.B) 'tighten'
kakupie (V.B) 'shout'
kapatau (V.B) 'add to'
kapuapie (V.B) 'wound'
kapupie (V.B) 'close tight'
kapuupie (V.B) 'close'
karepie (V.B) 'return'
karivai (V.A) 'have an appetite'
kasipie (V.B) 'care for'
kaukaupie (V.B) 'shine intensely'
kavorou (V.A) 'covet'
kavupie (V.B) 'leave.behind'
...
kuverea (V.A) 'incomplete'
```

Finding Minimal Sets

- E.g. mace vs maze, face vs faze
- minimal set parameters: context, target, display

Minimal Set	Context	Target	Display
bib, bid, big	first two letters	third letter	word
deal (N), deal (V)	whole word	pos	word (pos)

Finding Minimal Sets

- E.g. mace vs maze, face vs faze
- minimal set parameters: context, target, display

Minimal Set	**Context**	**Target**	**Display**
bib, bid, big	first two letters	third letter	word
deal (N), deal (V)	whole word	pos	word (pos)

Finding Minimal Sets: Example 1

```
>>> from nltk_lite.utilities import MinimalSet
>>> pos = 1
>>> ms = MinimalSet((lex[:pos] + '_' + lex[pos+1:], lex[pos], lex)
...                 for lex in lexemes if len(lex) == 4)
>>> for context in ms.contexts(3):
...     print context + ':',
...     for target in ms.targets():
...         print "%-4s" % ms.display(context, target, "-"),
...     print
k_si: kasi -    kesi -    kosi
k_ru: karu kiru keru kuru koru
k_pu: kapu kipu -    -    kopu
k_ro: karo kiro -    -    koro
k_ri: kari kiri keri kuri kori
k_pa: kapa -    kepa -    kopa
k_ra: kara kira kera -    kora
k_ku: kaku -    -    kuku koku
k_ki: kaki kiki -    -    koki
```

Finding Minimal Sets: Example 2

```
>>> entries = [(e.findtext('lx'), e.findtext('ps'), e.findtext('ge'))
...                for e in lexicon
...                if e.findtext('lx') and e.findtext('ps') and e.findt
>>> ms = MinimalSet((lx, ps[0], "%s (%s)" % (ps[0], ge))
...           for (lx, ps, ge) in entries)
>>> for context in ms.contexts()[:10]:
...     print "%10s:" % context, "; ".join(ms.display_all(context))
  kokovara: N (unripe coconut); V (unripe)
     kapua: N (sore); V (have sores)
      koie: N (pig); V (get pig to eat)
      kovo: C (garden); N (garden); V (work)
    kavori: N (crayfish); V (collect crayfish or lobster)
    korita: N (cutlet?); V (dissect meat)
      keru: N (bone); V (harden like bone)
  kirokiro: N (bush used for sorcery); V (write)
    kaapie: N (hook); V (snag)
       kou: C (heap); V (lay egg)
```

Example Application: Improving Access to Lexical Resources

- lexicon as a *language resource*
- existence of standardized writing system? literacy?
- lookup by approximate spelling
- soundex
- edit distance

Example Application: Improving Access to Lexical Resources

- lexicon as a *language resource*
- existence of standardized writing system? literacy?
- lookup by approximate spelling
- soundex
- edit distance

Example Application: Improving Access to Lexical Resources

- lexicon as a *language resource*
- existence of standardized writing system? literacy?
- lookup by approximate spelling
- soundex
- edit distance

Example Application: Improving Access to Lexical Resources

- lexicon as a *language resource*
- existence of standardized writing system? literacy?
- lookup by approximate spelling
- soundex
- edit distance

Example Application: Improving Access to Lexical Resources

- lexicon as a *language resource*
- existence of standardized writing system? literacy?
- lookup by approximate spelling
- soundex
- edit distance

Computing Soundex

```
>>> group = {
...     ' ':0,                                  # blank (for short words)
...     'p':1,  'b':1,  'v':1,                  # labials
...     't':2,  'd':2,  's':2,                  # alveolars
...     'l':3,  'r':3,                          # sonorant consonants
...     'i':4,  'e':4,                          # high front vowels
...     'u':5,  'o':5,                          # high back vowels
...     'a':6                                   # low vowels
... }
```

Computing Soundex (cont)

```
>>> def soundex(word):
...     if len(word) == 0: return word  # sanity check
...     word += '    '                  # ensure word long enough
...     c0 = word[0].upper()
...     c1 = group[word[1]]
...     cons = filter(lambda x: x in 'pbvtdslr ', word[2:])
...     c2 = group[cons[0]]
...     c3 = group[cons[1]]
...     return "%s%d%d%d" % (c0, c1, c2, c3)
>>> print soundex('kalosavi')
K632
>>> print soundex('ti')
T400
```

Soundex Index

```
>>> soundex_idx = {}
>>> for lex in lexemes:
...     code = soundex(lex)
...     if code not in soundex_idx:
...         soundex_idx[code] = set()
...     soundex_idx[code].add(lex)
```

Sorting and Lookup

```
>>> from nltk_lite.utilities import edit_dist
>>> def fuzzy_spell(target):
...     scored_candidates = []
...     code = soundex(target)
...     for word in soundex_idx[code]:
...         dist = edit_dist(word, target)
...         scored_candidates.append((dist, word))
...     scored_candidates.sort()
...     return [w for (d,w) in scored_candidates[:10]]

>>> fuzzy_spell('kokopouto')
['kokopeoto', 'kokopuoto', 'kokepato', 'koovoto', 'koepato',
 'kooupato', 'kopato', 'kopiito', 'kovuto', 'koavaato']
>>> fuzzy_spell('kogou')
['kogo', 'koou', 'kokeu', 'koko', 'kokoa', 'kokoi', 'kokoo',
 'koku', 'kooe', 'kooku']
```

Generating Summary Reports

- overall picture of the quality and organisation of the data
- e.g. average number of fields per record

```
>>> sum(len(entry) for entry in lexicon) / len(lexicon)
10
```

Generating Summary Reports (cont)

- E.g. entry patterns

```
>>> fd = FreqDist()
>>> for entry in lexicon:
...     for field in entry:
...         fd.inc(field.tag)
>>> fd.sorted_samples()[:10]
['ge', 'ex', 'xe', 'xp', 'gp', 'lx', 'ps',
'dt', 'rt', 'eng']
```

Generating Summary Reports (cont)

```
>>> fd = FreqDist()
>>> for entry in lexicon:
...     fd.inc(':'.join(field.tag for field in entry))
>>> top_ten = fd.sorted_samples()[:10]
>>> print '\n'.join(top_ten)
lx:rt:ps:ge:gp:dt:ex:xp:xe
lx:ps:ge:gp:dt:ex:xp:xe
lx:ps:ge:gp:dt:ex:xp:xe:ex:xp:xe
lx:rt:ps:ge:gp:dt:ex:xp:xe:ex:xp:xe
lx:ps:ge:gp:nt:dt:ex:xp:xe
lx:ps:ge:gp:dt
lx:ps:ge:ge:gp:dt:ex:xp:xe:ex:xp:xe
lx:rt:ps:ge:ge:gp:dt:ex:xp:xe:ex:xp:xe
lx:ps:ge:ge:gp:dt:ex:xp:xe
lx:rt:ps:ge:ge:gp:dt:ex:xp:xe
```

Generating Summary Reports (cont)

```
>>> fd = FreqDist()
>>> for entry in lexicon:
...     previous = "0"
...     for field in entry:
...         current = field.tag
...         fd.inc("%s->%s" % (previous, current))
...         previous = current
>>> fd.sorted_samples()[:10]
['ex->xp', 'xp->xe', '0->lx', 'ge->gp', 'ps->ge',
'dt->ex', 'lx->ps', 'gp->dt', 'xe->ex', 'lx->rt']
```

Grammar

```
>>> from nltk_lite import parse
>>> grammar = parse.cfg.parse_grammar('''
...    S -> Head "ps" Glosses Comment "dt" Examples
...    Head -> "lx" | "lx" "rt"
...    Glosses -> Gloss Glosses
...    Glosses ->
...    Gloss -> "ge" | "gp"
...    Examples -> Example Examples
...    Examples ->
...    Example -> "ex" "xp" "xe"
...    Comment -> "cmt"
...    Comment ->
...    ''')
>>> rd_parser = parse.RecursiveDescent(grammar)
```

Grammar

```
>>> for entry in lexicon[10:20]:
...     marker_list = [field.tag for field in entry]
...     if rd_parser.get_parse_list(marker_list):
...         print "+", ':'.join(marker_list)
...     else:
...         print "-", ':'.join(marker_list)
- lx:ps:ge:gp:sf:nt:dt:ex:xp:xe:ex:xp:xe
- lx:rt:ps:ge:gp:nt:dt:ex:xp:xe:ex:xp:xe
- lx:ps:ge:gp:nt:dt:ex:xp:xe:ex:xp:xe
- lx:ps:ge:gp:nt:sf:dt
- lx:ps:ge:gp:dt:cmt:ex:xp:xe:ex:xp:xe
+ lx:ps:ge:ge:ge:gp:cmt:dt:ex:xp:xe
+ lx:rt:ps:ge:gp:cmt:dt:ex:xp:xe:ex:xp:xe
+ lx:rt:ps:ge:ge:gp:dt
- lx:rt:ps:ge:ge:ge:gp:dt:cmt:ex:xp:xe:ex:xp:xe:ex:xp:xe
+ lx:rt:ps:ge:gp:dt:ex:xp:xe
```

Looking at Timestamps

```
>>> fd = FreqDist()
>>> from string import split
>>> for entry in shoebox.dictionary('rotokas'):
...     if 'dt' in entry:
...         (day, month, year) = split(entry['dt'], '/')
...         fd.inc((month, year))
>>> for time in fd.sorted_samples():
...     print time[0], '/', time[1], ':', fd.count(time)
Feb / 2005 : 307
Dec / 2004 : 151
Jan / 2005 : 123
Feb / 2004 : 64
Sep / 2004 : 49
May / 2005 : 46
Mar / 2005 : 37
Apr / 2005 : 29
Jul / 2004 : 14
Nov / 2004 : 5
Oct / 2004 : 5
...
```