1. Introduction to Natural Language Processing

1.1 Why Language Processing is Easy

How do we write programs to manipulate natural language? What questions about language could we answer? How would the programs work, and what data would they need? These are just some of the topics we will cover in these tutorials. Before we tackle the subject systematically, we will take a quick look at some simple programs that manipulate language data in a variety of interesting and non-trivial ways. Readers not familiar with programming or with the Python language will still gain a sense of what these programs do. Later chapters will explore the operation of these and other programs in depth.

The following program scans the CMU Pronunciation Dictionary for words putatively having *pre-preantepenultimate* stress. It scans all pronunciation strings **`pron`** and extracts the stress numbers from these strings, storing them in a variable called **`stress_pattern`**. If the stress pattern ends with **`1 0 0 0 0`**, indicating no stressed vowels after the fifth-last vowel, the corresponding word is printed. Observe that the word *spiritualists* appears multiple times, on account of the differing amounts of word-final obstruent reduction.

```
>>> from nltk_lite.corpora import cmudict
>>> from string import join
>>> for word, num, pron in cmudict.raw():
...     stress_pattern = join(c for c in join(pron) if c in "012")
...     if stress_pattern.endswith("1 0 0 0 0"):
...         print word, "/", join(pron)
ACCUMULATIVELY / AH0 K Y UW1 M Y AH0 L AH0 T IH0 V L IY0
AGONIZINGLY / AE1 G AH0 N AY0 Z IH0 NG L IY0
CARICATURIST / K EH1 R AH0 K AH0 CH ER0 AH0 S T
CIARAMITARO / CH ER1 AA0 M IY0 T AA0 R OW0
CUMULATIVELY / K Y UW1 M Y AH0 L AH0 T IH0 V L IY0
DEBENEDICTIS / D EH1 B EH0 N AH0 D IH0 K T AH0 S
DELEONARDIS / D EH1 L IY0 AH0 N AA0 R D AH0 S
FORMALIZATION / F AO1 R M AH0 L AH0 Z EY0 SH AH0 N
GIANNATTASIO / JH AA1 N AA0 T AA0 S IY0 OW0
HYPERSENSITIVITY / HH AY2 P ER0 S EH1 N S AH0 T IH0 V AH0 T IY0
IMAGINATIVELY / IH2 M AE1 JH AH0 N AH0 T IH0 V L IY0
INSTITUTIONALIZES / IH2 N S T AH0 T UW1 SH AH0 N AH0 L AY0 Z AH0 Z
INSTITUTIONALIZING / IH2 N S T AH0 T UW1 SH AH0 N AH0 L AY0 Z IH0 NG
MANGIARACINA / M AA1 N JH ER0 AA0 CH IY0 N AH0
SPIRITUALIST / S P IH1 R IH0 CH AH0 W AH0 L AH0 S T
SPIRITUALISTS / S P IH1 R IH0 CH AH0 W AH0 L AH0 S T S
SPIRITUALISTS / S P IH1 R IH0 CH AH0 W AH0 L AH0 S S
SPIRITUALISTS / S P IH1 R IH0 CH AH0 W AH0 L AH0 S
SPIRITUALLY / S P IH1 R IH0 CH AH0 W AH0 L IY0
```

```
UNALIENABLE / AH0 N EY1 L IY0 EH0 N AH0 B AH0 L
UNDERKOFFLER / AH1 N D ER0 K AH0 F AH0 L ER0
```

The following program processes a lexicon for the Rotokas language (East Papuan, Bougainville Island), and generates *minimal sets* involving the vowels of the first syllable. Each row of the resulting table provides evidence for vowel distinctions:

```
>>> from nltk_lite.corpora import shoebox
>>> from nltk_lite.utilities import MinimalSet
>>> length, position, min = 4, 1, 3
>>> lexemes = [field[1].lower() for entry in shoebox.raw('rotokas.dic')
...             for field in entry if field[0] == 'lx']
>>> ms = MinimalSet()
>>> for lex in lexemes:
...     if len(lex) == length:
...         context = lex[:position] + '_' + lex[position+1:]
...         target = lex[position]
...         ms.add(context, target, lex)
>>> for context in ms.contexts(3):
...     for target in ms.targets():
...         print "%-4s" % ms.display(context, target, "-"),
...     print
kasi -    kesi kusi kosi
kava -    -    kuva kova
karu kiru keru kuru koru
kapu kipu -    -    kopu
karo kiro -    -    koro
kari kiri keri kuri kori
kapa -    kepa -    kopa
kara kira kera -    kora
kaku -    -    kuku koku
kaki kiki -    -    koki
```

The above examples have used existing lexical resources. We can write programs to analyze plain text for style and patterns of word usage. The following program performs a simple bigram analysis of the book of Genesis (King James Version) then generates nonsense text in the same style. We can use similar techniques for genre and authorship identification.

```
>>> from nltk_lite.corpora import genesis
>>> from nltk_lite.probability import ConditionalFreqDist
>>> from nltk_lite.utilities import print_string
>>> cfdist = ConditionalFreqDist()
>>> prev = None
>>> for word in genesis.raw():
...     word = word.lower()
...     cfdist[prev].inc(word)
...     prev = word
>>> words = []
>>> prev = 'lo,'
>>> for i in range(99):
...     words.append(prev)
...     for word in cfdist[prev].sorted_samples():
...         if word not in words:
```

```
...             break
...     prev = word
>>> print_string(join(words))
lo, it came to the land of his father and he said, i will not be a
wife unto him, saying, if thou shalt take our money in their kind,
cattle, in thy seed after these are my son from off any more than all
that is this day with him into egypt, he, hath taken away unawares to
pass, when she bare jacob said one night, because they were born two
hundred years old, as for an altar there, he had made me out at her
pitcher upon every living creature after thee shall come near her:
yea,
```

The following program reads trees from the Penn Treebank corpus, finds instances of verb phrase conjunctions involving the word *but*, and displays strings of part-of-speech tags corresponding to the two verb phrases.

```
>>> from nltk_lite.corpora import treebank
>>> from string import join
>>> def vp_conj(tree):
...     if tree.node == 'VP' and len(tree) == 3 and tree[1].leaves() == ['but']:
...         return True
...     else:
...         return False
>>> for tree in treebank.parsed():
...     for vp1,conj,vp2 in tree.subtrees(vp_conj):
...         print join(child.node for child in vp1), "*BUT*", join(child.node for c
VBP ADVP-TMP PP-PRD PP *BUT* VBP VP
VBZ VP *BUT* VBZ NP PP-CLR
PP-TMP VBZ VP *BUT* VBD ADVP-TMP S
VBZ SBAR *BUT* VBZ SBAR
VBD SBAR *BUT* VBD RB VP
VBD SBAR *BUT* VBD S
VBP NP-PRD *BUT* VBP RB ADVP-TMP VP
VBN PP PP-TMP *BUT* ADVP-TMP VBN NP
MD VP *BUT* VBZ NP SBAR-ADV
VBD ADVP-CLR *BUT* VBD NP
VBN NP PP *BUT* VBN NP PP SBAR-PRP
VBD NP *BUT* MD RB VP
VBD NP PP-CLR *BUT* VBD PRT NP
VBZ S *BUT* MD VP
```

We have seen that that writing programs to manipulate natural language is can be accomplished using simple techniques and a small amount of Python code. In the following sections we will see why it is interesting and important.

Note

An important aspect of learning NLP using these materials is to experience both the challenge and -- we hope -- the satisfaction of creating software to process natural language. The accompanying software, NLTK, is available for free and runs on most operating systems including Linux/Unix, Mac OSX and Microsoft Windows. You can download NLTK from `http://nltk.sourceforge.net/`, along with extensive documentation. We encourage you to install NLTK on your machine before reading beyond the end of this chapter.

1.2 The Language Challenge

Language is the chief manifestation of human intelligence. Through language we express basic needs and lofty aspirations, technical know-how and flights of fantasy. Ideas are shared over great separations of distance and time. The following samples from English illustrate the richness of language:

1. Overhead the day drives level and grey, hiding the sun by a flight of grey spears. (William Faulkner, *As I Lay Dying*, 1935)
2. When using the toaster please ensure that the exhaust fan is turned on. (sign in dormitory kitchen)
3. Amiodarone weakly inhibited CYP2C9, CYP2D6, and CYP3A4-mediated activities with Ki values of 45.1-271.6 μM (Medline)
4. Iraqi Head Seeks Arms (spoof headline, **`http://www.snopes.com/humor/nonsense/head97.htm`**
5. The earnest prayer of a righteous man has great power and wonderful results. (James 5:16b)
6. Twas brillig, and the slithy toves did gyre and gimble in the wabe (Lewis Carroll, *Jabberwocky*, 1872)
7. There are two ways to do this, AFAIK :smile: (internet discussion archive)

Thanks to this richness, the study of language is part of many disciplines outside of linguistics, including translation, literary criticism, philosophy, anthropology and psychology. Many less obvious disciplines investigate language use, such as law, hermeneutics, forensics, telephony, pedagogy, archaeology, cryptanalysis and speech pathology. Each applies distinct methodologies to gather observations, develop theories and test hypotheses. Yet all serve to deepen our understanding of language and of the intellect which is manifested in language.

The importance of language to science and the arts is matched in significance by the cultural treasure that is inherent in language. Each of the world's ~7,000 human languages is rich in unique respects, in its oral histories and creation legends, down to its grammatical constructions and its very words and their nuances of meaning. Threatened remnant cultures have words to distinguish plant subspecies according to therapeutic uses which are unknown to science. Languages evolve over time as they come into contact with each other and they provide a unique window onto human pre-history. Technological change gives rise to new words like *blog* and new morphemes like *e-* and *cyber-*. In many parts of the world, small linguistic variations from one town to the next add up to a completely different language in the space of a half-hour drive. For its breathtaking complexity and diversity, human language is as a colourful tapestry stretching through time and space.

Each new wave of computing technology has faced new challenges for language analysis. Early machine languages gave way to high-level programming languages which are automatically parsed and interpreted. Databases are interrogated using linguistic expressions like **`SELECT age FROM employee`**. Recently, computing devices have become ubiquitous and are often equipped with multimodal interfaces supporting text, speech, dialogue and pen gestures. One way or another, building new systems for natural linguistic interaction will require sophisticated language analysis.

Today, the greatest challenge for language analysis is presented by the explosion of text and multimedia content on the world-wide web. For many people, a large and growing fraction of work and leisure time is spent navigating and accessing this universe of information. *What tourist sites can*

I visit between Philadelphia and Pittsburgh on a limited budget? What do expert critics say about Canon digital cameras? What predictions about the steel market were made by credible commentators in the past week? At present, humans require skill, knowledge, and some luck, to extract answers to these questions from existing search engines. Getting a computer to answer them automatically would involve a range of language processing tasks, including information extraction, inference, and summarisation. The scale of such tasks often calls for high-performance computing.

As we have seen, *natural language processing*, or NLP, is important for scientific, economic, social, and cultural reasons. NLP is experiencing rapid growth as its theories and methods are deployed in a variety of new language technologies. For this reason it is important for a wide range of people to have a working knowledge of NLP. Within academia, this includes people in areas from humanities computing and corpus linguistics through to computer science and artificial intelligence. Within industry, this includes people in human-computer interaction, business information analysis, and web software development. We hope that you, a member of this diverse audience reading these materials, will come to appreciate the workings of this rapidly growing field of NLP and will apply its techniques in the solution of real-world problems. The following chapters present a carefully-balanced selection of theoretical foundations and practical application, and equips readers to work with large datasets, to create robust models of linguistic phenomena, and to deploy them in working language technologies. By integrating all of this with the Natural Language Toolkit (NLTK), we hope this book opens up the exciting endeavour of practical natural language processing to a broader audience than ever before.

1.3 A Brief History of Natural Language Processing

A long-standing challenge within computer science has been to build intelligent machines. The chief measure of machine intelligence has been a linguistic one, namely the Turing Test: can a dialogue system, responding to a user's typed input with its own textual output, perform so naturally that users cannot distinguish it from a human interlocutor using the same interface? Today, there is substantial ongoing research and development in such areas as machine translation and spoken dialogue, and significant commercial systems are in widespread use. The following dialogue illustrates a typical application:

S: How may I help you?
U: When is Saving Private Ryan playing?
S: For what theater?
U: The Paramount theater.
S: Saving Private Ryan is not playing at the Paramount theater, but
it's playing at the Madison theater at 3:00, 5:30, 8:00, and 10:30.

Today's commercial dialogue systems are strictly limited to narrowly-defined domains. We could not ask the above system to provide driving instructions or details of nearby restaurants unless the requisite information had already been stored and suitable question and answer sentences had been incorporated into the language processing system. Observe that the above system appears to understand the user's goals: the user asks when a movie is showing and the system correctly determines from this that the user wants to see the movie. This inference seems so obvious to humans that we usually do not even notice it has been made, yet a natural language system needs to be endowed with this capability in order to interact naturally. Without it, when asked "Do you know when Saving Private Ryan is playing", a system might simply -- and unhelpfully -- respond with a cold "Yes". While it appears

that this dialogue system can perform simple inferences, such sophistication is only found in cutting edge research prototypes. Instead, the developers of commercial dialogue systems use contextual assumptions and simple business logic to ensure that the different ways in which a user might express requests or provide information are handled in a way that makes sense for the particular application. Thus, whether the user says "When is ...", or "I want to know when ...", or "Can you tell me when ...", simple rules will always result in users being presented with screening times. This is sufficient for the system to provide a useful service.

Despite some recent advances, it is generally true that those natural language systems which have been fully deployed still cannot perform common-sense reasoning or draw on world knowledge. We can wait for these difficult artificial intelligence problems to be solved, but in the meantime it is necessary to live with some severe limitations on the reasoning and knowledge capabilities of natural language systems. Accordingly, right from the beginning, an important goal of NLP research has been to make progress on the holy grail of natural linguistic interaction *without* recourse to this unrestricted knowledge and reasoning capability. This is an old challenge, and so it is instructive to review the history of the field.

The very notion that natural language could be treated in a computational manner grew out of a research program, dating back to the early 1900s, to reconstruct mathematical reasoning using logic, most clearly manifested in the work by Frege, Russell, Wittgenstein, Tarski, Lambek and Carnap. This work led to the notion of language as a formal system amenable to automatic processing. Three later developments laid the foundation for natural language processing. The first was *formal language theory*. This defined a language as a set of strings accepted by a class of automata, such as context-free languages and pushdown automata, and provided the underpinnings for computational syntax.

The second development was *symbolic logic*. This provided a formal method for capturing selected aspects of natural language that are relevant for expressing logical proofs. A formal calculus in symbolic logic provides the syntax of a language, together with rules of inference and, possibly, rules of interpretation in a set-theoretic model; examples are propositional logic and first-order logic. Given such a calculus, with a well-defined syntax and semantics, it becomes possible to associate meanings with expressions of natural language by translating them into expressions of the formal calculus. For example, if we translate *John saw Mary* into a formula `saw(j,m)`, we (implicitly or explicitly) intepret the English verb *saw* as a binary relation, and *John* and *Mary* as denoting individuals. More general statements like *All birds fly* require quantifiers, in this case $\forall$ meaning *for all*: $\forall$ x: bird(x) $\rightarrow$ fly(x). This use of logic provided the technical machinery to perform inferences that are an important part of language understanding.

The third development was the *principle of compositionality*. This was the notion that the meaning of a complex expression is comprised of the meaning of its parts and their mode of combination. This principle provided a useful correspondence between syntax and semantics, namely that the meaning of a complex expression could be computed recursively. Given the representation of *It is not true that -$_p$* as `not(p)` and *John saw Mary* as `saw(j,m)`, we can compute the interpretation of *It is not true that John saw Mary* recursively using the above information to get `not(saw(j,m))`. Today, this approach is most clearly manifested in a family of grammar formalisms known as unification-based grammar, and NLP applications implemented in the Prolog programming language.

A separate strand of development in the 1960s and 1970s eschewed the declarative/procedural distinction and the principle of compositionality. They only seemed to get in the way of building practical systems. For example, early question answering systems employed fixed pattern-matching templates such as: *How many -$_i$ does -$_j$ have?*, where slot `i` is a feature or service, and slot `j` is a person or place. Each template came with a predefined semantic function, such as `count(i,j)`. A user's question which matched the template would be mapped to the corresponding semantic function

and then "executed" to obtain an answer, `k = count(i,j)`. This answer would be substituted into a new template: *-$_j$ has -$_k$ -$_i$*. For example, the question *How many airports$_i$ does London$_j$ have?* can be mapped onto a template (as shown by the subscripts) and translated to an executable program. The result can be substituted into a new template and returned to the user: *London has five airports*. Finally, the subscripts are removed and the natural language answer is returned to the user.

This approach to NLP is known as *semantic grammar*. Such grammars are formalized like phrase-structure grammars, but their constituents are no longer grammatical categories such as *Noun Phrase*, but semantic categories like *Airport* and *City*. These grammars work very well in limited domains, and are still widely used in spoken language systems. However, they lack robustness and portability, and they duplicate grammatical structure in different semantic categories.

The contrasting approaches to NLP described in the preceding paragraphs relates back to early metaphysical debates about *rationalism* versus *empiricism* and *realism* versus *idealism* that occurred in the Enlightenment period of Western philosophy. These debates took place against a backdrop of orthodox thinking in which the source of all knowledge was believed to be divine revelation. During this period of the seventeenth and eighteenth centuries, philosophers argued that human reason or sensory experience has priority over revelation. Descartes and Leibniz, amongst others, took the rationalist position, asserting that all truth has its origins in human thought, and in the existence of "innate ideas" implanted in our minds from birth. For example, they saw that the principles of Euclidean geometry were developed using human reason, and were not the result of supernatural revelation or sensory experience. In contrast, Locke and others took the empiricist view, that our primary source of knowledge is the experience of our faculties, and that human reason plays a secondary role in reflecting on that experience. Prototypical evidence for this position was Galileo's discovery -- based on careful observation of the motion of the planets -- that the solar system is heliocentric and not geocentric. In the context of linguistics, this debate leads to the following question: to what extent does human linguistic experience, versus our innate "language faculty", provide the basis for our knowledge of language? In NLP this matter surfaces as differences in the priority of corpus data versus linguistic introspection in the construction of computational models.

A further concern, enshrined in the debate between *realism* and *idealism*, was the metaphysical status of the constructs of a theory. Kant argued for a distinction between phenomena, the manifestations we can experience, and "things in themselves" which can never been known directly. A linguistic realist would take a theoretical construct like "noun phrase" to be real world entity that exists independently of human perception and reason, and which actually *causes* the observed linguistic phenomena. A linguistic idealist, on the other hand, would argue that noun phrases, along with more abstract constructs like semantic representations, are intrinsically unobservable, and simply play the role of useful fictions. The way linguists write about theories often betrays a realist position, while NLP practitioners occupy neutral territory or else lean towards the idealist position.

These issues are still alive today, and show up in the distinctions between symbolic vs statistical methods, deep vs shallow processing, binary vs gradient classifications, and scientific vs engineering goals. However, these contrasts are highly nuanced, and the debate is no longer as polarised as it once was. In fact, most of the discussions -- and most of the advances even -- involve a *balancing act* of the two extremes. For example, one intermediate position is to assume that humans are innately endowed with analogical and memory-based learning methods (weak rationalism), and use these methods to identify meaningful patterns in their sensory language experience (empiricism). For a more concrete illustration, consider the way in which statistics from large corpora may serve as evidence for binary choices in a symbolic grammar. For instance, dictionaries describe the words *absolutely* and *definitely* as nearly synonymous, yet their patterns of usage are quite distinct when combined with a following verb, as shown below:

Absolutely vs Definitely (Liberman 2005, LanguageLog.org)				
Google hits	**adore**	**love**	**like**	**prefer**
absolutely	289,000	905,000	16,200	644
definitely	1,460	51,000	158,000	62,600
ratio	198/1	18/1	1/10	1/97

Observe that *absolutely adore* is about 200 times as popular as *definitely adore*, while *absolutely prefer* is about 100 times rarer then *definitely prefer*. This information is used by statistical language models, but it also counts as evidence for a symbolic account of word combination in which *absolutely* can only modify extreme actions or attributes. This information could be represented as a binary-valued feature of certain lexical items. Thus, we see statistical data informing symbolic models. Once this information has been codified, it is available to be exploited as a contextual feature for a statistical language modelling, alongside many other rich sources of symbolic information, like hand-constructed parse trees and semantic representations. Now the circle is closed, and we see symbolic information informing statistical models.

This new rapprochement is giving rise to many exciting new developments. We will touch on some of these in the ensuing pages. We too will perform this balancing act, employing approaches to NLP that integrate these historically-opposed philosophies and methodologies.

1.4 The Architecture of linguistic and NLP systems

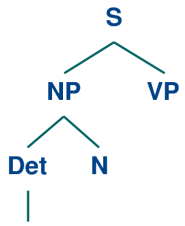
Within the approach to linguistic theory known as generative grammar, it is claimed that humans have distinct kinds of linguistic knowledge, organised into different modules: for example, knowledge of a language's sound structure (phonology), knowledge of word structure (morphology), knowledge of phrase structure (syntax), and knowledge of meaning (semantics). In a formal linguistic theory, each kind of linguistic knowledge is made explicit as different *module* of the theory, consisting of a collection of basic elements together with a way of combining them into complex structures. For example, a phonological module might provide a set of phonemes together with an operation for concatenating phonemes into phonological strings. Similarly, a syntactic module might provide labelled nodes as primitives together wih a mechanism for assembling them into trees. A set of linguistic primitives, together with some operators for defining complex elements, is often called a level of representation.

As well as defining modules, a generative grammar will prescribe how the modules interact. For example, well-formed phonological strings will provide the phonological content of words, and words will provide the terminal elements of syntax trees. Well-formed syntactic trees will be mapped to semantic representations, and contextual or pragmatic information will ground these semantic representations in some real-world situation.

As we indicated above, an important aspect of theories of generative grammar is that they are intended to model the linguistic knowledge of speakers and hearers; they are not intended to explain how humans actually process linguistic information. This is, in part, reflected in the claim that a generative grammer encodes the *competence* of an idealized native speaker, rather than the speaker's *performance*. A closely related distinction is to say that a generative grammar encodes *declarative* rather than *procedural* knowledge. As you might expect, computational linguistics has the crucial role of proposing procedural models of language. A central example is parsing, where we have to develop computational mechanisms which convert strings of words into structural representations such as syntax trees. Nevertheless, it is widely accepted that well-engineered computational models of

language contain both declarative and procedural aspects. Thus, a full account of parsing will say how declarative knowledge in the form of a grammar and lexicon combines with procedural knowledge which determines how a syntactic analysis should be assigned to a given string of words. This procedural knowledge will be expressed as an algorithm: that is, an explicit recipe for mapping some input into an appropriate output in a finite number of steps.

A simple parsing algorithm for context-free grammars, for instance, looks first for a rule of the form $\mathtt{S} \rightarrow \mathtt{X}_1 \cdots \mathtt{X}_n$, and builds a partial tree structure. It then steps through the grammar rules one-by-one, looking for a rule of the form $\mathtt{X}_1 \rightarrow \mathtt{Y}_1$ ... $\mathtt{Y}_j$ which will expand the leftmost daughter introduced by the `S` rule, and further extends the partial tree. This process continues, for example by looking for a rule of the form $\mathtt{Y}_1 \rightarrow \mathtt{Z}_1$ ... $\mathtt{Z}_k$ and expanding the partial tree appropriately, until the leftmost node label in the partial tree is a lexical category; the parser then checks to see if the first word of the input can belong to the category. To illustrate, let's suppose that the first grammer rule chosen by the parser is `S` → `NP VP` and the second rule chosen is `NP` → `Det N`; then the partial tree will be:



If we assume that the input string we are trying to parse is *the cat slept*, we will succeed in identifying *the* as a word which can belong to the category `Det`. In this case, the parser goes on to the next node of the tree, `N`, and next input word, *cat*. However, if we had built the same partial tree with an input string *did the cat sleep*, the parse would fail at this point, since *did* is not of category `Det`. The parser would throw away the structure built so far and look for an alternative way of going from the `S` node down to a leftmost lexical category (e.g., using a rule `S` → `V NP VP`). The important point for now is not the details of this or other parsing algorithms; we discuss this topic much more fully in the chapter on parsing. Rather, we just want to illustrate the idea that an algorithm can be broken down into a fixed number of steps which produce a definite result at the end.

In the following figure we further illustrate some of these points in the context of a spoken dialogue system, such as our earlier example of an application that offers the user information about movies currently on show.

Down the lefthand side of the diagram we have shown a pipeline of some representative speech understanding *components*. These map from speech input via syntactic parsing to some kind of meaning representation. Up the righthand side is an inverse pipeline of components for concept-to-speech generation. These components constitute the procedural aspect of the system's natural language processing. In the central column of the diagram are some representative declaratives aspects: the repositories of language-related information which are called upon by the processing components.

In addition to embodying the declarative/procedural distinction, the diagram also illustrates that linguistically motivated ways of modularizing linguistic knowledge are often reflected in computational systems. That is, the various components are organized so that the data which they exchange corresponds roughly to different levels of representation. For example, the output of the speech analysis component will contain sequences of phonological representations of words, and the output of the parser will be a semantic representation. Of course the parallel is not precise, in part because it is often a matter of practical expedience where to place the boundaries between different processing components. For example, we can assume that within the parsing component there is a level of syntactic representation, although we have chosen not to expose this at the level of the system diagram. Despite

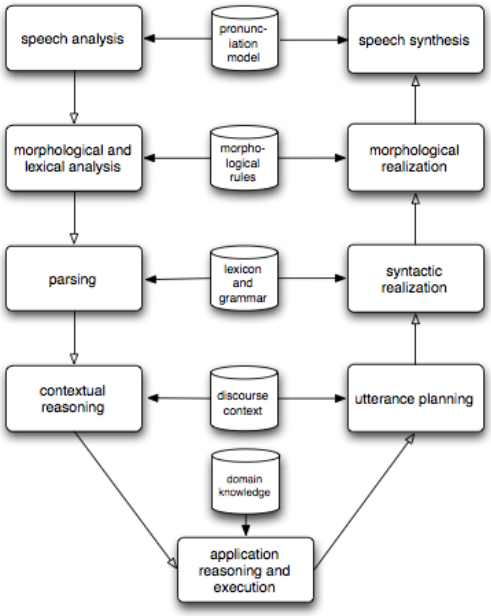


Figure 1: Architecture of Spoken Dialogue System

such idiosyncracies, most NLP systems break down their work into a series of discrete steps. In the process of natural language understanding, these steps go from more concrete levels to more abstract ones, while in natural language production, the direction is reversed.

1.5 The Python Programming Language

NLTK is written in the Python language, a simple yet powerful scripting language with excellent functionality for processing linguistic data. Python can be downloaded for free from **`http://www.python.org/`**. Here is a five-line Python program which takes text input and prints all the words ending in **`ing`**:

```
>>> import sys                          # load the system library
>>> for line in sys.stdin.readlines():  # for each line of input
...     for word in line.split():       # for each word in the line
...         if word.endswith('ing'):    # does the word end in 'ing'?
...             print word              # if so, print the word
```

This program illustrates some of the main features of Python. First, whitespace is used to *nest* lines of code, thus the line starting with **`if`** falls inside the scope of the previous line starting with **`for`**, so the **`ing`** test is performed for each word. Second, Python is *object-oriented*; each variable is an entity which has certain defined attributes and methods. For example, **`line`** is more than a sequence of characters. It is a string object that has a method (or operation) called **`split`** that we can use to break a line into its words. To apply a method to an object, we give the object name, followed by a period, followed by the method name. Third, methods have *arguments* expressed inside parentheses. For instance, **`split`** had no argument because we were splitting the string wherever there was white space. To split a string into sentences delimited by a period, we could write **`split('.')`**. Finally, and

most importantly, Python is highly readable, so much so that it is fairly easy to guess what the above program does even if you have never written a program before.

We chose Python as the implementation language for NLTK because it has a shallow learning curve, its syntax and semantics are transparent, and it has good string-handling functionality. As a scripting language, Python facilitates interactive exploration. As an object-oriented language, Python permits data and methods to be encapsulated and re-used easily. Python comes with an extensive standard library, including components for graphical programming, numerical processing, and web data processing.

NLTK defines a basic infrastructure that can be used to build NLP programs in Python. It provides:

- Basic classes for representing data relevant to natural language processing.
- Standard interfaces for performing tasks, such as tokenization, tagging, and parsing.
- Standard implementations for each task, which can be combined to solve complex problems.
- Extensive documentation, including tutorials and reference documentation.

1.6 Further Reading

The Association for Computational Linguistics (ACL) is the foremost professional body in NLP. Its journal and conference proceedings, approximately 10,000 articles, are available online with a full-text search interface, via **`http://www.aclweb.org/anthology/`**.

Several NLP systems have online interfaces that you might like to experiment with, e.g.:

- WordNet: **`http://wordnet.princeton.edu/`**
- Translation: **`http://world.altavista.com/`**
- ChatterBots: **`http://www.loebner.net/Prizef/loebner-prize.html`**
- Question Answering: **`http://www.answerbus.com/`**
- Summarisation: *http://newsblaster.cs.columbia.edu/*‘

Useful websites with substantial information about NLP: **`http://www.hltcentral.org/`**, **`http://www.lt-wo`**
`http://www.aclweb.org/`, **`http://www.elsnet.org/`**. The ACL website contains an overview of computational linguistics, including copies of introductory chapters from recent textbooks, at **`http://www.aclweb.org/archive/what.html`**.

Recent field-wide surveys: Mitkov, Dale et al, HLT Survey.

Acknowledgements: The dialogue example is taken from Bob Carpenter and Jennifer Chu-Carroll’s ACL-99 Tutorial on Spoken Dialogue Systems.

1.6.1 Development of NLTK

Edward Loper and Steven Bird (2002). NLTK: The Natural Language Toolkit, *Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*, Somerset, NJ: Association for Computational Linguistics, pp. 62-69, http://arXiv.org/abs/cs/0205028

Steven Bird and Edward Loper (2004). NLTK: The Natural Language Toolkit, *Proceedings of the ACL demonstration session*, pp 214-217.

Steven Bird (2005). NLTK-Lite: Efficient Scripting for Natural Language Processing, *4th International Conference on Natural Language Processing*, pp 1-8.

Edward Loper (2004). NLTK: Building a Pedagogical Toolkit in Python, *PyCon DC 2004* Python Software Foundation, http://www.python.org/pycon/dc2004/papers/