Artif Intell Rev (2010) 33:151–173
DOI 10.1007/s10462-009-9151-4

Natural language processing: a prolog perspective

Christian Bitter · David A. Elizondo · Yingjie Yang

Published online: 4 December 2009

Abstract Natural language processing (NLP) is a vibrant field of interdisciplinary Computer Science research. Ultimately, NLP seeks to build intelligence into software so that software will be able to process a natural language as skillfully and artfully as humans. Prolog, a general purpose logic programming language, has been used extensively to develop NLP applications or components thereof. This report is concerned with introducing the interested reader to the broad field of NLP with respect to NLP applications that are built in Prolog or from Prolog components.

Keywords Natural language processing · Prolog · Applications of natural language processing

1 Introduction

Natural language processing (NLP) or *natural language understanding* (NLU) is a field of Computer Science (CS), exploring how natural language, like the English language, can be processed mechanically utilising computer systems. NLP researches the design of mathematical models of natural language structures and their implementation. Human aspiration to make computers process natural language, is as old as the idea of computers themselves. NLP is not an isolated field of CS research but has emerged in several other sciences, such as *computational linguistics* in linguistics, *speech recognition* (SR) in electrical engineering and *computational psycholinguistics* in psychology (Jurafsky and Martin 2008).

C. Bitter · D. A. Elizondo (✉) · Y. Yang
Centre for Computational Intelligence, Faculty of Technology, DeMontfort University,
The Gateway, Leicester, LE1 9BH, United Kingdom
e-mail: elizondo@dmu.ac.uk

C. Bitter
e-mail: christian.bitter@gmail.com

Y. Yang
e-mail: yyang@dmu.ac.uk

Ideally, computers would process natural language as skillfully as humans to carry out tasks, such as analysing and understanding huge amounts of information, communicating with humans in their preferred mode of communication and capturing, understanding and acting on human utterances. According to Turing, computers participating in the mentioned tasks would be ascribed the capacity of being intelligent (Jurafsky and Martin 2008). Unfortunately, until now convincing natural language based computer programs were restricted to quite limited domains of discourse. But the steady increase in raw computing power available on-hand to a wide audience, the ubiquitous availability of information in the form of the Web and readily accessible networks that enable communication anytime, anywhere, have spotlighted NLP applications; such as spelling and grammar checkers, text indexing and information retrieval (IR), speech dictation, voice control of domestic devices, interactive voice response, machine translation (MT), conversational agents (CAs), or systems which enable firms to mine social media content (blog, wiki) and analyse how their products and services are perceived (opinion, sentiment) by reviewers and (prospective) customers (Pang and Lee 2008).

If one looks at a modern NLP system, one realises that the system's architecture is comprised of various components which correspond to tasks or problems within *domains/sub-disciplines/ levels* of NLP (Nugues 2006). For example, Persona is a conversational assistance system, which helps a user to select music from a record database (DB).

At the time of writing, some of the mentioned applications like spell checkers are in widespread use, whereas others are still not mature enough to be adopted by industry-adopted. Maturity or success-rate of a NLP system is usually assessed using statistical terms (e.g. precision, recall), performing either intrinsic or extrinsic evaluation. *Intrinsic evaluation* refers to evaluating a NLP system's performance against a gold standard, which is usually a human-expert defined evaluation result. *Extrinsic evaluation* assesses a NLP system's ability to serve a defined purpose. Unfortunately, identifying suitable processes to evaluate, i.e. measuring the difference between system computed and expected result for a NLP system is not easy and may vary considerably across the different domains and tasks of NLP. For example, creation of a gold standard to perform intrinsic evaluation usually requires identification of suitable data (corpus) to perform evaluation on and reaching consent among human-experts with respect to the evaluation result (annotation). This might be as simple as marking words and numbers within some sentence or defining the translated equivalent to a spoken text. Also, measures used to assess NLP system performance are rather old and may be inappropriate and unable to capture a description of a particular system's performance. For this, current NLP research is also concerned with the advancement of NLP system evaluation procedures and measures. For example, Nakache, Metais and Timsit's development of the K-measure, a synthesisation of the mentioned precision and recall measure, accounts for the diversification and increase in complexity of NLP problems and goals (Nakache et al. 2005).

But why are NLP systems in some cases not mature enough? What is it that makes NLP difficult? One aspect of NLP's difficulty is the problem of *ambiguity* and fuzziness inherent to human language (Zadeh 1965). Ambiguity concerns, among others, word sense, word category, syntactic structure and semantic scope. For example, a speech processing system must be able to disambiguate spoken words that sound alike, such as "I" and "eye". When analyzing the syntax of sentences, it is important to disambiguate the grammatical function a word occupies. For example, in the phrase "the mouse moves", the word "moves" might act as a verb or in conjunction with "mouse" might occupy the role of a noun (proper noun, named entity). Given the lexical category of a word, it might be ambiguous with respect to semantics, i.e. the word might adopt different meanings. For example, in the previous sequence of words, the word "mouse" might refer to an animal, to a quiet and timid person

or to the computer input device. Especially in the context of IR and MT, a common understanding of a word's meaning has to be ensured to avoid dramatic mistakes like the famous MT failure where the English sentence "The spirit is strong, but the flesh is weak." was translated into Russian and incorrectly translated from Russian back to the English sentence "The vodka is good, but the meat is rotten." When trying to mechanically understand and infer knowledge from sentences, ambiguity with respect to contextual semantics and problems like coreference resolution, i.e. forward or backward resolution of expressions pointing to the same referent, have to be dealt with. To resolve ambiguity, knowledge spanning across several NLP sub-disciplines is needed. This knowledge acts as a constraint, limiting the number of possible solutions (e.g., the depth and breadth of solution space) to a particular NLP task. Consequently, providing and integrating knowledge and thereby resolving ambiguity is essential to overcoming immaturity of NLP applications.

Another aspect to consider when developing a NLP system is to choose an appropriate *formal model* and its underlying *algorithmic representation*. Models to choose from include state machines, rule systems, logic, probabilistic models and vector-space models. The choice and/or design of a good model is essential and unfortunately not easy. The reason for this is natural language's critical connection with human *thoughts* and *understanding* (Jurafsky and Martin 2008; Nugues 2006; Russell and Norvig 2003; Manning and Schütze 1999). Consequently, the design and implementation of an appropriate model to solve some NLP problem may require studying the human mind to incorporate peculiarities, with respect to the given NLP task, into the desired model.

Historically, in the 1950s, solutions to NLP problems followed an *empiricist* approach, such as conceiving MT as the problem of reconstructing a message transmitted over a noisy channel using channel characteristics and stochastic techniques. From around the 1960s to 1980s of the last century, this paradigm shifted to assuming that human language faculties directly result from extensive knowledge and inference mechanisms hardwired into the human brain. Consequently, the *rationalist* paradigm dominated the cognition and mindset of the NLP domain leading to embracing symbol processing techniques to treat NLP tasks, such as NLU systems based on pattern matching and syntax generation and acceptance via the grammar formalism and parser. By the 1980s, the availability of large-scale annotated natural language corpora (text, speech), increasing raw computing power and recognition of artificial intelligence (AI) learning techniques, have restored attention and confidence in the empiricist paradigm. Consequently, problems like recognising speech, determining the functional role of words and translating or summarising text were attacked by developing probabilistic models trained on problem-specific data pools (e.g., reports, email correspondence). Lately, two trends have emerged in the development of NLP solutions. Firstly, there has been a focus on unsupervised learning approaches to avoid up front costs associated with the design of annotated corpora used to implement models relying on supervised learning. Secondly, there has been the development of hybrid NLP systems, i.e. incorporating elements from empiricist and rationalist approach.

The rest of this report is structured as follows. Sects. 2 and 3 motivate the use of Prolog in the context of NLP application development. Section 4 presents a classification of NLP systems discussed throughout this report. Sections 5–9 discuss NLP sub-disciplines, following the flow of speech showcasing Prolog-based solutions pertaining to problems of a particular NLP sub-discipline. Finally, Sect. 10 aggregates NLP sub-disciplines and presents Prolog-based NLP systems like a CA. It is assumed that the reader has a working knowledge of NLP and the Prolog programming language. For an introduction to the discussed material see (Jurafsky and Martin 2008; Russell and Norvig 2003; Bratko 2000; Grune and Jacobs 2008; Jurafsky and Martin 2008; Manning and Schütze 1999).

2 Prolog and NLP

Prolog is a declarative general purpose logic programming language. A Prolog program consists of facts and rules relating objects of the problem domain to each other. The set of all logical consequences derivable from a program constitutes the meaning of the program. The meaning of the program is computed by Prolog's inference engine at run-time.

Pereira and Shieber mention that one goal of Prolog's development was to design and construct a language in which phrase-structure and semantic interpretation rules for a natural language question-answering system could be easily expressed (Pereira and Shieber 1987). According to the authors, Prolog is an excellent choice to represent and solve NLP problems, because:

> phrase structure rules have a very simple expression in first-order logic ... logic grammar has come to refer to such uses of logic to formalise grammatical rules ... linguistic rules encoded ... can be run directly by Prolog, providing ... computational realisation of grammar and interpretation rules.

Besides built-in support to represent phrase structure rules, Prolog offers the following advantages regarding problem-solving in general and NLP in particular.

Firstly, Prolog has integrated search and unification mechanisms that make it an ideal candidate to implement formal models of linguistics. For example, built-in search and unification can be used effectively to solve problems requiring searching on and matching of structures (e.g, graph, tree, etc.), such as generating natural language sentence(s) from a given grammar or (dis)proving whether a natural language sentence is valid (can be generated) by a particular grammar (see Sects. 6, 7).

Secondly, Prolog's declarative nature allows focusing on problem-solving. Declarative rules are a simple yet powerful self-documenting form of representing and modeling domain knowledge. For example, a dictionary provides knowledge about a certain domain by declaring characteristics of domain entities, such as a word which belongs to a particular word class/ common root/ etc. This form of knowledge can be represented easily exploiting Prolog's declarative way of modeling knowledge (facts, rules) of a particular closed domain.

Thirdly, translation from pseudo-code algorithms is much easier to a high level programming language like Prolog than to a low level programming language like C/C++ (McConnell 2004). The reason for this is the smaller semantic gap that has to be bridged between pseudo-code and high level programming language code. For this, Prolog is particularly useful when rapid-prototyping or for problems concerning the implementation of formal methods and general algorithmic concepts. For example, a Prolog programmer may treat concepts like (phrase-structure) grammar rules as a first class citizen from within the programming language. Clearly, source code that is closer to the programmer's way of thinking is easier to read, understand, extend and maintain.

Fourthly, Prolog's variable scoping rules are simple and uniform. If not needed, variable names need not be made explicit. This reduces code size, opportunity for human error and maintenance cost. Fifthly, manipulation of large and complex data structures like lists is easy and efficient (McClain 2003). Also, Prolog programs are easily portable, since Prolog interpreters exist for various platforms, such as MS Windows and Linux. For example, SWI-Prolog, a free Prolog implementation widely used in research, education, and commercial applications, is available on the MS-Windows, Linux and MacOS X platform. SICStus Prolog, a commercial implementation is also available on multiple platforms like MS-Windows, Linux, MaxOS X, AIX, Solaris 8/10 and more. A list of free and commercial Prolog implementations is maintained by Roman Barták (Barták 1998). Clearly, these advantages may aid

in the development of large, complex and oftentimes resource-hungry AI systems that may span and integrate several platforms (OS) and devices (desktop PC, mobile).

Fifthly, a Prolog program is capable of modifying itself, enabling it to learn. The ability to learn is crucial when developing software systems (e.g. software agents), which have to deal with open, dynamic and unfamiliar environments or situations, such as a dialogue between a software system and its user. For example, a CA developed in Prolog may learn throughout the act of conversing with a user about her specific needs and demands. Consequently, the CA may change (personalise) its way of providing a service to the user.

Since Prolog is an interpreted language, it facilitates easy exploration of a program's execution state via for example, the `trace/{0,1,2}` or `debug/{0,1,3}` predicate. Also, run-time memory management, which is important when working with dynamic data types (string) and structures (list, graph), is taken care of by the interpreter. This clearly limits the potential for errors related to allocation and deallocation or access of dynamic memory. Lastly, components developed in Prolog may be reused through code modules (`module/2`),`use_module`/{1,2}), i.e. exposing functionality through a publicly accessible interface and hiding implementation details, thereby allowing for decoupled, modular software architecture, which is easier to maintain and extend.

Disadvantages of Prolog, which are outweighed for the most part by described benefits can be (1) low efficiency in terms of CPU and memory utilisation when compared with low level programming languages like C/C++ (Lodewijk Van Roy 1991) and (2) interfacing capabilities with packages written in different programming languages, such as Java or C# (Kiraz and Grimley-Evans 1998).

3 NLP in Prolog: a natural choice?

Section 2 has focused on presenting Prolog's advantages and limitations without targeting a specific domain. It was argued that one of the design goals of Prolog was suitability with respect to certain NLP tasks and although it is a language rooted in formal logic, Prolog has advanced significantly and can be considered a general purpose programming language. But why should practitioners of NLP consider Prolog a natural, i.e. consequent, choice to mechanically process natural languages? Is Prolog a suitable language to devise language models whether they follow the rationalist, empiricist, or an in-between approach? To answer these questions will be this section's aim.

When looking at the benefits that Prolog offers with respect to NLP, one realises that advantages may be innate to Prolog (internal), i.e. result directly from the language's design (syntax, semantics), or can be attributed to the language's ecosystem (external). External advantages, such as standard library, IDE and tooling support may vary substantially across different Prolog implementations and are therefore not discussed.

Prolog's easy to learn syntax and transparent semantics are rather simple compared to programming languages like Java or C#. Prolog has only one built-in data type—the term (atom, number, variable, compound term)—which can be composed to form more complex relations and queries. Due to Prolog's pureness and simplicity, its learning curve is rather shallow. For example, to define that the word "house" is a noun, one might devise the following Prolog fact `noun(house).`. To allow for a compact representation of differ-ent word classes, one might introduce the following binary relation `word_class(house, noun).`. Assuming one has declared various words, one might ask Prolog about all known nouns via: `?- word_class(X, noun).`).

As illustrated by this example, Prolog programs explicitly describe or declare relations, defined by means of clauses. This declarative form of expressing knowledge about some domain is a concept shared with DBs. Consequently, Prolog can be used to efficiently store, organise and query knowledge about a particular domain, such as a dictionary or encyclopaedia. Although the terminology of primary and foreign key is predominantly used in the domain of DB design and implementation, the concepts of normalisation, redundancy, uniqueness and reference are valid in Prolog (clause DB) as well. For example, imagine a simple dictionary that might be implemented in a relational DB using the following tables: `dict_entry(id, word_class_id, word, word_relation_id)`, `word_class(id, value)` and `word_relation(id, type, word_id)`. Instantiations of these tables, i.e. rows, can be easily expressed using Prolog compound terms . For example, `dict_entry(1, 1, house, []).` `word_class(1, noun)` . declares that the dictionary has an entry `1`, which is the word "house" member of the word class noun. The `[]` (empty list) is used to indicate that there are no relations (synonymy, antonymy, valency, etc.) defined for the word house. To find all nouns in the dictionary one would use the `join` operator and a constraining `where` clause in SQL and the logical conjunction operator, (comma) in Prolog (e.g., `?- word_class(ID, noun), dict_entry(_, ID, WORD, _).`).

Data structures like a graph and computation devices such as finite-state automata (FSA) are often used in NLP (see Sect. 6). Since FSA build on set theoretic relations, they can be implemented in Prolog with relative ease. For example, the unary predicate `node/1` can be used to declare an FSA node. The ternary relation `transition/3` can be employed to represent labeled transitions connecting individual nodes. FSA have either an acceptive or a generative character, i.e. they can be used to generate or check for acceptance of words (character streams) belonging to some underlying alphabet. Also, FSA can be used to implement regular expressions, i.e. a string used to describe sets and subsets of character strings using a canonic compact notation.

Since Prolog has built-in support for numbers (integer, real), one may perform counting tasks (e.g., word, character, sentence) and compute frequency and probability distributions accordingly. Consequently, NLP tools and language models building on stochastic, probabilistic and/or graph structure algorithmic concepts like Hidden Markov Model (HMM), Bayesian network or (un)supervised learning can be realised in Prolog with minimal effort (boilerplate code). HMMs have been used effectively in systems to recognise speech or similar tasks that involve matching of patterns. Bayesian networks have been employed to realise probabilistic expert systems. Unsupervised learning approaches like clustering have been used in the identification of a publication's central theme or main topics. As mentioned, Prolog's support for handling large quantities of data (clauses), symbol processing capabilities and ability to allow for numerical computations can be used to store and make use of data obtained from processing text corpora, such as bigram distributions, i.e. the conditional probability of two letters/syllables/ words/ co-occurring. For this, Prolog facilitates rapid-prototyping of NLP components following the rationalist, empiricist or hybrid approaches.

String handling is of uttermost importance in NLP, given that words are character strings and texts are words separated by special characters like whitespaces or sentence delimiters. Prolog is capable of efficiently representing and performing operations on strings. One may represent a string as an atom or enclosed in quotation marks (`house`, '`House`'). Strings can be manipulated using operations that work on atoms or strings. For example, the atom `house` can be transformed into a list of characters via `atom_chars(house, CharacterList).`. Consequently, `CharacterList` (e.g., `[h, o, u, s, e]`) can be manipulated using list manipulating predicates like `reverse/2`, which reverses

`CharacterList`, i.e. leads to `[e, s, u, o, h]`. In turn, the predicate `sub_string/5` can be used to extract fragments from a string. For example, one might use `sub_string` to test whether some string corresponds to a verb in progressive tense by checking whether the string ends in `ing`, i.e. `sub_string(Word, _, _, 0, 'ing').`.

Sometimes, instead of performing a possibly expensive full analysis of some input stream it might be sufficient for some NLP application to simply try spotting patterns and process them accordingly. Prolog's built-in unification mechanism can be used to match on and process patterns present in an input stream. For example, assuming the word `house` is found in a sentence, `house` may assume the role of verb, adjective or noun. For this, one might decide, if at the current state of input processing the determiner `the` is found, which is immediately succeeded by `house`, then `house` assumes the role of a noun. This pattern can be expressed in Prolog via: `match([the,house|_]). match([_|Y]) :- match(Y).`

As noted, Prolog was developed with NLP in mind. For this, it provides built-in support to express generation of formal and natural languages from an underlying grammar via the definite clause grammar (DCG) formalism, i.e. the representation of grammar production rules using first-order-logic (FOL)'s definite clauses. Consequently, instead of designing and implementing a parser for and processor of grammar rules, one may use the Prolog built-in implementation. DCG can be used to construct lexical and syntactical analysers. For example, the DCG rule `noun -> [house].` declares that the non-terminal symbol `noun` can be rewritten as the terminal symbol `house`.

Grammar rules like "a sentenceis composed from a noun phrase followed by a verb phrase" can be thought of as objects associated with grammatical properties (number, tense, etc.). Properties can be represented by constraints. For example, a constraint might indicate that a verb phrase's number has to agree with the noun phrase's number. Constraints can be implemented via feature structures, which can be realised using Prolog lists containing atomic features or nested lists. For example, the number feature could be represented as `[number, singular]`. Testing for agreement between two objects like noun phrase and verb phrase means testing whether feature structures of both objects can be unified. This can be realised using Prolog's unification operator.

Combining Prolog's DCG formalism, ability to organise and represent knowledge via predicates, and built-in inference mechanism, one is able to associate language with meaning. Prolog predicates (Horn clauses) can be used to effectively model and reason over a universe of discourse. Representing the state of affairs of some world can be achieved using predicate argument structures (PAS). PAS map individual words, or compounds like phrases and sentences onto logical formulae. Nouns and adjectives can be represented via unary predicates, which indicate that the predicate's argument has the property of being of the predicate's type (is, is a). For example, the following sentence "House45 is a house." might be encoded via `house('House45')`. Verbs, depending on their valency, can be modeled using unary, binary, ternary or quaternary relations. N-nary predicates can be used to describe n-nary relations between objects. For example, to indicate that House45 is positioned at the centre of some world, one might devise the following predicate `position('House45', vec(0,0,0))`.

Making use of Prolog's built-in solution search mechanism and backtracking, problems relying on dynamic programming techniques, i.e. computing a global solution by incrementally computing local solutions, can be implemented relatively straightforward. For example, trying to determine the similarity of two strings can be accomplished via computing their Minimum Edit Distance (MED). MED is often used by word processors when performing spell-checking. Computing MED requires finding the shortest (least expensive) path from one string (start state) to the other (goal state). Costs are associated with primitive string

operations like inserting/ deleting/ substituting a character. Given a current state `S` in solution search space, all of `S`'s reachable states S_i can be computed using Prolog's built-in `bagof/3`, `setof/3` and `findall/{3, 4}` predicates.

4 Classification

In the following, a classification is discussed, which arranges software systems of NLP subdisciplines and NLP applications presented throughout this report, into a logical structure. Classification criteria to choose from are manifold. For example, one might classify natural language parsers (see Sect. 7) according to run-time and memory *complexity*, *problem-solving strategy*, such as top-down (TD) or bottom-up (BU) or the employed NLP *model* and *algorithmic* representation thereof. Unfortunately, these criteria do not allow for the arrangement of presented systems in a coherent framework. For example, complexity, while being a suitable criterion in the context of algorithms (e.g. parsing algorithms), is not meaningful in the context of applications spanning multiple subsystems, i.e. making use of several algorithms. For this, two classification criteria are chosen, namely *NLP-discipline* and *knowledge representation*.

NLP-discipline refers to the levels/ sub-disciplines of NLP mentioned in Sect. 1. Please note that NLP applications are encompassed by this criterion as well, although they may span several NLP levels.

The phonetic class covers Prolog software systems that are either partly or completely responsible for the transformation of audible to legible speech and vice versa or the representation of an audible utterance, which can be used to reason on the utterance's characteristics (see Sect. 5). Morphology class members are software components that make use of a language's morphological characteristics, namely language's words belonging to the same word family can (mostly) be formed by considering a common word stem (with respect to the word family) and situationally applying word-specific affixes, in order to identify the morphological, syntactical and semantical properties of individual words (see Sect. 6). Software components belonging to the syntax class deal with the principles and rules for constructing sentences from individual words. For this, formal models are devised and implemented in Prolog, which have generative or acceptive character, i.e. a formal model is used to produce sentences of a particular language or to (dis)prove/ predict a sentence's belonging to the language under consideration (see Sect. 7). Semantics class' members are Prolog components that are concerned with capturing, assigning or representing general or situational meaning contained in pools of knowledge, such as structured documents containing natural language or unstructured documents like images. Therefore, these software systems make use of formal systems, such as logic, to capture sense and represent meaning (see Sect. 8). The lexicon class contains Prolog applications, which organise words of a particular language into a coherent framework according to some distinguishing criterion (see Sect. 9). The classification a lexicon imposes on its entries may be with respect to a covered domain (biology) or language features (word class, relation). For this, a lexicon can be regarded as a form of consultable database representing and arranging a specific language or subset thereof as a closed world entity. The application class consists of software systems, which implement or use functionality from one or more NLP levels in order to solve a particular real-world problem, such as enabling a human user to perform a particular task using her language of choice (see Sect. 10).

The knowledge representation criterion captures three approaches, i.e. the knowledge-deep, knowledge-shallow and hybrid approach. Knowledge-deep refers to NLP systems that

Table 1 Classification of discussed NLP systems

System/author	Citation	NLP level	Knowledge approach
SWI-Speech	McClain (2003)	Phonetic	Knowledge-deep
Alexin et al.	Alexin et al. (1997)	Phonetic	Hybrid
Finite-state toolkit	Kiraz and Grimley-Evans (1998)	Morphology	Knowledge-deep
Kiraz	Kiraz (2000)	Morphology	Knowledge-deep
ProNTo Morph	Schlachter (2003)	Morphology	Knowledge-deep
CHR grammar	Christiansen (2002)	Syntax	Knowledge-deep
PCFG	Abou-Assaleh et al. (2003)	Syntax	Knowledge-shallow
Liang	Liang (2002)	Syntax	Knowledge-deep
Schneider	Schneider (2003)	Syntax	Hybrid
Tascini et al.	Tascini et al. (2001)	Semantics	Knowledge-deep
Baral et al.	Baral et al. (2007)	Semantics	Knowledge-deep
WordNet	Witzig (2003), Princeton University (2009)	Lexicon	Knowledge-deep
TDb/ BP-WordNet	Dias-da-Silva et al. (2002)	Lexicon	Knowledge-deep
Hettige and Karunananda	Hettige and Karunananda (2007)	Lexicon	Knowledge-deep
Xiao-xi and Chang-le	Xiao-xi and Chang-le (2007)	Lexicon	Knowledge-deep
InCA-Probot	Kadous and Sammut (2004)	Application	Knowledge-deep
NL-Ue	Quintano and Rodrigues (2006)	Application	Knowledge-deep

make use of explicit domain knowledge like rules to capture phenomena or peculiarities of natural language. Knowledge-shallow techniques model linguistic structure and effects through stochastic or probabilistic measures. Hybrid refers to approaches that make use of knowledge-deep and knowledge-shallow techniques at the same time. Please note that means to capture and externalise knowledge like lexicon are themselves classified as knowledge-deep because of their explicit modeling of domain knowledge.

The obtained classification (see Table 1), although containing more instances of knowledge-deep NLP systems, does neither imply a trend or best-practice to NLP approaches (see Sect. 1) nor indicate a weakness or deficiency of the Prolog programming language with respect to implementing knowledge-shallow or hybrid NLP systems (see Sects. 2, 3). Where a classified system is nameless, the author/s of that system is/are used as the table's entry.

5 Phonetics

Phonetics is concerned with the production and perception of acoustic sounds that form the speech signal. Vibrations of the vocal cords generate acoustic sounds or signals. Sounds can be sampled using microphones, digitised using analogue-to-digital converters and subsequently processed and transformed by a Fourier analysis. This process chain yields an acoustic signal's *spectrogram* describing the distribution of speech energy as a function of frequency. Acoustic sounds of a language can be grouped into a finite set of *phonemes* (roughly 10 to 80 phonemes). Phonemes can be seen as the smallest discriminating unit of speech. Phonemes can be separated into vowels and consonants. Vowels can be distinguished by monophthongs and diphthongs. Consonant phonemes fall into five categories namely, approximants, nasals, fricatives, plosives and affricates. Individual spoken words are composed from phoneme

syllables. Sentences in turn are formed from words. *Prosody* describes the general rhythm of a sentence. Due to prosody's relation to length, sentence structure and type and meaning of words, it is different among different languages.

Given an acoustic signal describing natural language speech, speech recognition (SR) is the process of transcribing that speech signal into readable text. SR makes use of signal processing, statistical techniques (e.g, Hidden Markov models) and language models. Speech synthesis (SS)/ text-to-speech (TTS) is the reverse process. It uses signal processing techniques, phoneme models and letter-to-phoneme rules to transform written text into its equivalent audible representation.

To enable Prolog applications take advantage of SR and TTS services provided by the MS Windows operating system, McClain designed a Prolog interface to MS Speech API (SAPI), called SWI-Speech (McClain 2003). SWI-Speech addresses the problem of inaccessibility of NLP libraries from Prolog, which is due to their implementation in programming languages like C/C++.

SWI-Speech uses SWI-Prolog's C++ interface exposed through `SWI-cpp.h`. It provides two main predicates to surface SAPI functionality. These predicates are `listen/1` and `speak/1`. `listen` addresses SR and `speak` provides TTS functionality in Prolog. `listen` and `speak` can be adjusted to user needs using additional predicates, such as `set_volume/1`, `set_rate/1`, `set_age/2` or `set_gender/2`. For example, appropriately invoking the predicate `set_gender` before calling `speak`, applies male or female voice characteristic to the generated speech signal.

SWI-Speech allows users to exploit speech processing capabilities directly accessible from MS Windows thereby reducing time spent, opportunity for error and cost to develop a NLP application in Prolog. Existing Prolog-based applications can be equipped with SR or TTS functionality. For example, applications providing traditional console-based user input via natural language text could be extended to support a speech-based user interface, by utilising SWI-Speech's `listen/1` predicate. Unfortunately, SWI-Speech has drawbacks, such as it does not allow specification of new grammars for SAPI instead relying on SAPI's default grammar, it does not expose SAPI's `vendor` attribute and as a high-level wrapper it hides fine-grained control over SAPI. While this may be sufficient in rapid-prototyping scenarios it surely imposes a restriction. However, due to SWI-Speech's available source code, it can be extended to be a Prolog-based equivalent to SAPI.

Using a Prolog implementation for MS's .NET platform like Prolog.NET or P# enables users to access SAPI too (Hodroj 2006; Cook 2004). SAPI functionality is grouped below the `System.Speech` namespace. Allowing access to the extensive feature set provided by the .NET framework is an advantage of Prolog.NET. Unfortunately, Prolog.NET is not a feature-complete port of SWI-Prolog to the .NET platform. For example, fact DB manipulating predicates like `dynamic/1`, `asserta/1` or `retract/1` are not implemented. This clearly limits Prolog.NET's applicability to AI problems relying on dynamic knowledge DBs. For example, software Agents operating autonomously on behalf of a user (human, software) in a possibly large unrestricted dynamic world must continuously question (e.g. assert truth) their knowledge about the current state of the virtual world they exist in. Although P# is neither a feature-complete port of Prolog nor an officially supported alternative to traditional .NET programming languages like C#, it shows that Prolog can be ported to and integrated with the .NET platform. This allows for reusing developed functionality in the form of .NET assemblies (I/O, XML, GUI). Since the .NET platform supports dynamic code execution at run-time and a C# interface to SWI-Prolog exists, C# code may be executed from Prolog.NET and C# may use the SWI-Prolog interpreter at run-time. This "mix-and-match" of programming languages and paradigms (logic vs. imperative programming) allows for

picking the best of both worlds. For example, C# may be used to design a GUI connected with a Prolog-implemented TTS system.

Concerning SR, Alexin et al. propose a rule-based approach to learning phonetic rules to identify vowels in spoken Hungarian words, which is implemented in Prolog (Alexin et al. 1997). Given a Hungarian sentence, an acoustic front-end converts the sentence's digital speech signal into a spectrogram representation. Using the spectrogram, its envelope is computed using GAS. GAS is a function structure discovering algorithm. The identified envelope is used to extract acoustic features, namely *formants* and their characteristics.

A formant is a peak in the frequency spectrum of a sound caused by acoustic resonance. Formants are sufficient to describe vowels (Alexin et al. 1997). For this, the rule DB contains Prolog clauses, which describe a Hungarian vowel in terms of the characteristics of its first four formants, i.e. F1, F2, F3 and F4. A formant in the Prolog DB is given as a sixth-tuple. A set of formants, i.e. set of sixth-tuples, describes an acoustic sound in the Prolog DB. Rules describing vowels in terms of its formants were obtained from a standard Hungarian phonetic rule book. This initial set of rules was not sufficient in terms of vowel classification success rate. For this, the rule set was generalised to serve as the basis for rule learning. Rule learning was performed on a training data set using IMPUT. IMPUT, an inductive logic programming (ILP) tool, learns the target concept, i.e. a Hungarian vowel, via revising the vowel rule DB. Revision of the rule base is done by transforming or removing vowel discriminating rules from the rule DB. Using IMPUT improved the rule DB and consequently, the system's ability to recognise Hungarian vowels.

Advantages of this approach include self-documenting systems because of domain expertise embedded as rules, lowered system complexity and improved maintenance because of rules. The system's vowel recognition performance can be improved by incorporating more acoustic features and rules to describe Hungarian vowels. Also, GAS and IMPUT are not language-specific approaches to modeling vowels in speech and could be reused to provide support for new languages. Clearly, a sufficiently large training data set and carefully chosen target concept (vowel characteristics) would have to be in place to support different languages.

It is often argued that rule-based systems are less flexible than stochastic-based systems (Alexin et al. 1997; Manning and Schütze 1999). For example, to increase coverage of a rule-based system with respect to some specific phenomena (e.g. recognition of Hungarian vowel), it is often necessary to manually fine-tune individual rules responsible for recognition performance, which requires domain (linguistic) and system knowledge on behalf of the person maintaining the system's rule DB. Clearly, this is in stark contrast to stochastic-based systems, where a system simply has to be retrained on some data set, which sufficiently reflects (e.g. contains instances of) the phenomenon in question.

6 Morphology

Morphology is the study of word composition from *morphemes*, i.e. word stem/root and affixes. Morphology distinguishes between *inflection* and *derivation*. Inflection considers the application of a grammatical feature to words (gender, number, tense). Derivation combines affixes to an existing root to derive new words. Most of word's inflectional morphology can be described through morphological rules. Commonly accepted, the *two-level morphology model* links a word's surface form, i.e. the word as it is in a text, to its lexical or underlying form, i.e. its sequence of morphemes. For example, using the English word 'happier', its

surface form is 'happier' and its lexical form (word stem + affix) is 'happy+er'. Given a word's root form, morphological rules allow to generate all word forms from their root.

Because of their ease of implementation and mathematical elegance finite-state automata (FSA) have been used extensively in NLP tasks like morphological parsing (Kiraz and Grimley-Evans 1998; Nugues 2006). There are two kinds of FSA, namely acceptors and transducers. Acceptors have one input tape and transitions between states are labeled with one symbol from the underlying alphabet. Transducers have one input and one output tape. Their state transitions are marked with an input and output symbol of the underlying alphabet. A n-tape FSA is a FSA that, starting from an initial state, scans all symbols on each of the n-tapes such that it transitions from initial to a final state. A n-tape finite state transducer (FST) is a n-tape FSA with each tape marked with the word `domain` or `range`. The advantage of FST over FSA is that FST works bidirectionally allowing it to start from a word's lexical and proceed to the word's surface form and vice versa.

Kiraz and Grimley-Evans propose the implementation of a multi-tape finite-state engine, called finite-state calculus engine, to solve NLP tasks like morphological parsing. The engine allows the construction of a (n-tape) FSA from a regular expression using the Prolog predicate `regexp_to_fsa/2`. An FSA can also be specified in terms of its states and transitions using the predicate `make_automaton/4`. FSA can be manipulated using standard operations, such as intersection, union, difference and concatenation. The finite-state calculus allows the compilation of regular rewrite rules into FSA. A rewrite rule specifies under what condition parts of a word will be substituted with some other part. For example, the finite-state calculus rewrite rule `n:m ⇒--- p:p` states that 'n' rewrites as 'm' before a 'p'. Historically, declarative rewrite rules were used to describe morphology. It has been shown that any two-level rule can be compiled into an equivalent transducer. This provides the theoretical basis of the implementation of rewrite rules in Kiraz and Grimley-Evans' finite-state calculus.

A concise and feature-rich FSA library reduces time spent, opportunity for error and cost of developing NLP applications like a morphological parser. FSA toolkit is built on a strongly researched theoretical basis, consequently, applications utilising FSA toolkit implicitly inherit its theoretical properties like memory and run-time complexity or totality—a concern that might be of importance in real-time environments. Due to equal power of expression between regular expression, FSA and regular languages, problems stated in the latter might be solved more conveniently using FSA or regular expressions and vice versa. The authors tested different strategies to implement the finite-state calculus. For example, they experimented with hashing techniques and different forms of representing FSA transitions. Implementation strategies are discussed in terms of strengths and weaknesses. This helps developers to understand and avoid possible pitfalls related to using and implementing a FSA toolkit. The authors point out that, regarding efficiency, a Prolog implementation differs from an equivalent C/C++ implementation only by a constant factor. Out-of-the-box, FSA toolkit allows for language analysis using regular expressions, such as segmentation or tokenization of sentences. As such it might directly benefit from or be adopted by solutions which have traditionally been implemented using the programming language Perl or the Unix command-line tool grep.

Kiraz and Grimley-Evans's finite-state calculus was later extended by Kiraz to the development of a computational framework able to handle nonlinear, i.e. nonconcatenative, morphology (Kiraz 2000). Kiraz proposes a multi-tier morphological model that is able to handle nonlinear morphology in semitic languages like Syriac or Arabic. The multi-tier model consists of three components, namely multi-tier lexicon, bidirectional rewrite rules mapping between a surface form and multiple lexical forms and morphotactic constraints. Lexicon and rewrite rules are compiled into n-tape FSA and the morphotactic component is compiled

into a FSA. Using compiled FSAs, Kiraz discusses the morphological analysis of Syriac words in the context of different strategies like Kay's multitape or the intersection of lexicon approach. Kiraz's work shows that FSA and FST are suitable techniques for modeling language morphology which is much more complex, i.e. richer in features (e.g. German) or non-linear (e.g. Arabic), than the morphology of English.

An extension of Convington's "Part of English Morphology", called ProNTo_Morph, was developed by Schlachter (Schlachter 2003). ProNTo_Morph is a morphological parser for the English language implemented in ISO Prolog. It accepts input in several forms, namely list of tokens, list of characters or list of character lists, and atom or list of atoms.

Given the morphological parsing of a word, ProNTo_Morph returns this parsing as deterministic or indeterministic output, i.e. either a list of all morphological parsings or a list of one morphological parse possibly offering alternative parsings using backtracking. For this, ProNTo_Morph provides six Prolog predicates, i.e. `morph_{tokens|chars|atoms}/2` and `morph_{tokens|chars|atoms}_bag/2` to morphologically parse input. Irregular words, i.e. words that cannot be parsed, are stored in an irregular word list. The extensible irregular word list is based on WordNet's word lists (see Chapter 9). If one wants to use custom morphological parsing rules, one may modify `pronto_morph_spelling_rules.pl`.

ProNTo_Morph is an easy-to-use morphological parser, which due to its limitations, for example, inability to handle ambiguous words (e.g., word that depending on its surrounding context can have multiple morphological parses), can be employed in prototypical scenarios or as a reference implementation. Due to its rule-based approach, it automatically inherits advantages, such as easier maintainability and integrated documentation because of domain knowledge exposed via declarative rules. Also, results (e.g. a morphological parse) can be verified when applied rules are traced at run-time. Being able to deterministically return a word's morphological parsing increases traceability and helps during exception handling. Also, if more than one morphological parse for some word exists, computed parses may be ranked and annotated according to some predetermined characteristic. This in turn may be used by consumers of ProNTo_Morph's output, such as components that are concerned with the syntax of a natural language. ProNTo_Morph integrates into a larger NLP-tool ecosystem—the University of Georgia's Prolog Natural Language Toolkit (PRONTO). Other components of PRONTO include an efficient tokeniser (Convington 2003), a text statistics toolbox (Hu 2003), SWI-Speech (see Chapter 5), a free-word-order dependency parser (Convington 2003) and a Prolog-based implementation of the Earley parsing algorithm (Voss 2004). Unfortunately, Schlachter does not evaluate his approach with respect to correctness and performance of parsing results. The customisation of WordNet files to build an irregular word list, while being feasible, is not WordNet-integrated. This means that ProNTo_Morph does not take advantage of WordNet's ongoing advancement.

7 Syntax

Syntax is concerned with the formation of a sentence from individual words. Syntax is independent from semantics and can be expressed in terms of *grammars* consisting of a set of *rules* describing a language's sentence structure. Grammar rules can be used to generate the complete sentence set of a definite language. Generative grammars consist of syntactic rules, i.e. *phrase-structure* (PS) rules, which fractionate a phrase into subphrases. Different phrase categories exist, such as noun phrase (NP) or verb phrase (VP). A phrase-structure grammar (PSG) is a set of PS rules that can decompose sentences and phrases down to words. Trees

can be used to represent the syntactic structure of sentences and reflect the rules involved in sentence generation.

A PSG can be expressed in Prolog via the DCG mechanism, i.e. a translation of PS rules using Prolog syntax resulting in DCG rules. DCG is a Prolog preprocessor that takes DCG grammar rules and translates them into pure Prolog. DCG parsing corresponds to Prologć6s TD search starting from the DCG's start symbol. Grammars like probabilistic context free grammar (PCFG) exist, which make grammar rules subject to probability thereby enabling them to cope with ambiguous and noisy input, i.e. sentences with multiple parses (Abou-Assaleh et al. 2003).

Grammars can be used to generate or recognise, i.e. parse, syntactically correct sentences. Parsing a sentence means matching it against a grammar's rules to check whether it can be generated from the grammar. Parsing results in a parse tree, i.e. the sequence of grammar rules applied during the parse process. Parsing can be carried out in a TD or BU fashion. TD parsing starts from the start symbol, i.e. the sentence, and proceeds down to the sentence's individual words, whereas BU parsing begins at the individual word level and proceeds up to the sentence symbol (Grune and Jacobs 2008). Commonly used parsing algorithms include the shift-reduce parser, a TD parsing approach, and the CYK and Earley parsing algorithms, which are BU parsers (Thurston 2007; Grune and Jacobs 2008; Nugues 2006).

Christiansen used constraint handling rules (CHR) to devise a grammar formalism – CHR grammars (CHRG) (Christiansen 2002). CHR allows us to combine BU and TD evaluation in Prolog thereby avoiding some disadvantages of either evaluation strategy. It has been shown that if some non-expressibility-limiting restrictions are imposed on CHRG, i.e. the CHRG must not contain cycles/loops and constraints and it must be range restricted, CHRG can be used to implement CFG and DCG (Christiansen 2002). This means, CHRG acts as a BU parser for CFG or DCG. The advantage of CHRG is its inherent ability to cope with ambiguity realised via a generalised form of look-ahead. That is, a CHRG rule can take arbitrary grammar symbols to the left and right of a sequence into account, which is supposed to be matched with a given non-terminal.

The ability to cope with ambiguity and its expressibility make constrained CHRG an ideal candidate to resolve ambiguity and left-recursive rule problems (e.g. non-terminating cycling) of CFG and DCG. Another advantage of CHRG over DCG is that CHRG avoids backtracking to generate alternative solutions. This avoids combinatorial explosion of solution search space, reducing complexity to that of other algorithms like CYK or Earley (e.g. cubic complexity). Christiansen implemented CHRG in a variant of Prolog called SICStus Prolog. Disadvantages of the proposed approach are the restriction to a special class of CHRG, non-intuitive expression of DCG and CFG, i.e. BU instead of TD, increased complexity of grammar rules and reduced run-time performance due to SICStus Prolog when parsing ambiguous grammars.

Schneider describes a state-of-the-art parser, which combines advantages and aims to reduce shortcomings of formal language and probabilistic parsers (Schneider 2003). Formal language parsers generally have good coverage of most syntactic phenomena but sometimes suffer from incomplete lexicon, inflexibility of underlying grammar and run-time complexity. Probabilistic parsers, which learn from large annotated corpora, have good run-time performance but their output does not include deep linguistic information like co-indexation.

Schneider combines both types of parser into a hybrid approach. The parser uses the CYK algorithm to parse a dependency grammar (DG), which consists of manually crafted rules. Crafting DG rules manually allows the author to exclude treatment of linguistic phenomena which, albeit being possible, are considerably rare and would otherwise reduce parser performance and stability. The parser assigns probabilistic scores to DG rules for pruning and

disambiguation of parses. Rule scores are learnt from an annotated corpus using a supervised learning model based on Maximum Likelihood Estimation. Employing probabilised DG rules enables the parser to rank syntactically possible parses and to keep only the n-highest scoring parses, thereby excluding improbable and impossible parses early during the parse process. This reduces memory and run-time complexity, i.e. reduces size of solution search space (state space containing all possible parses) and therefore time spent searching for probable parses.

The parser is implemented in ISO-Prolog and has been tested with SWI-Prolog. Some advantages and disadvantages of this hybrid approach have already been mentioned. Another advantage is that rules are a form of domain knowledge documentation embedded into the parser's implementation. Rules facilitate easier extension, customisation and maintenance of the parser. Disadvantages include the need to provide an annotated corpus to train the parser, automatically derived probabilistic scores which are not intuitively comprehensible and difficult, and non-transferability of DG rules to a different language, i.e. DG rules are rigid with respect to the language they model. Another drawback is that system maintenance/ extension requires system and domain knowledge on behalf of its user. Also, finding an appropriate threshold for the number of highest scoring parses to keep or reject might require non-trivial fine-tuning as it aims to identify an appropriate trade-off between system performance and correctness.

Implementation of compilers is primarily done using imperative languages like C/C++. Liang shows that by using λ-Prolog as the implementation language for a compiler, disadvantages of imperative languages can be circumvented (Liang 2002). He presents a methodology for implementing a compiler for the Sparc architecture. Except for the assembler parts, which are carried out by GCC, all compiler stages are implemented using λ-Prolog. Underlying his methodology is the bounded right context grammar (BRCG). It has been shown that every left-recursive grammar (LRG) can be transformed into an equivalent BRCG. BRCG facilitates the formulation of deterministic BU parsing as a λ-Prolog proof search. Consequently, BCRG like CHRG can be used to resolve ambiguity and non-determinism of DCG.

Formulating compilation in a logic programming language acknowledges that many compilation tasks are declarative in nature. An advantage of Liang's approach is that λ-Terms, which are natively supported by λ-Prolog, can be used to capture locality of names directly. Imperative languages capture name locality using external structures like a symbol table. Externalising knowledge via declarative rules, which guide the compilation process, enables for the separation of specification and execution of compilation phases. This in turn would allow design, verification and optimisation of compilation rules to be treated separately from their interpretation and execution by a specific compiler. Disadvantages of Liang's approach are Sparc machine code generation limits transferability to other machine architectures, high CPU- and memory utilisation during compilation and limited coverage of typical compiler tasks, such as data-flow analysis and object code generation.

8 Semantics

Semantics as part of NLP is concerned with the study of *meaning*, as inherent to words, phrases, sentences and texts. A key concern of semantics is how meaning attaches itself to larger portions of text, possibly due to its composition from smaller units of meaning. Formal semantics is concerned with the modeling of meaning in terms of the semantics of *logic*. Its underlying argument is that logic can model language and consequently, human thought. Semantics addresses four important areas. Firstly, it deals with the representation of phrases

and sentences in terms of logical formula. Secondly, formal semantics is considered with the definition of truth, i.e. computing the truth value of a sentence's logical representation using deduction or inference. Thirdly, formal semantics accounts for the determination of reference, i.e. linking a word to an object of the real world. Lastly, formal semantics allows for reasoning, i.e. given the truth value of sentences, conjectures and undiscovered knowledge may be proven or disproven. A special form of semantics is *pragmatics*, which is semantics restricted to a specific context (Truszczyński 2007).

Images and video, i.e. a series of static images, can be regarded as large semantic-rich documents of unstructured data. Adding meta-data to content greatly improve IR tasks. But proper granularity has to be ensured. This means the description of data (meta-data) must be apt to the user and application targeted by the IR task. If a description is too general, valuable detail regarding the described content is lost. On the other hand, general principles and concepts are hidden if the description is too specific.

For this, Tascini et al. focus on extraction of characteristic features of video, using extracted features to describe the video and embedding semantics into the video from extracted features (Tascini et al. 2001). Characteristics of a video frame are extracted using the Mosaic technique, i.e. isolating moving objects from scene background. Using extracted background and moving objects, an Extractor captures characteristics of a frame. This means that a scene's background and moving objects are stored as facts in a Prolog DB. For example, a scene's object `car` is stored in the DB along with its coordinates within the scene. The set of all facts of all video frames denotes the video's syntax or syntactic description.

A linguistic interface is used to embed semantics into a video frame. That is, relations among features of a frame, such as occlusion or motion of scene objects given via Prolog predicates, are stored in the Prolog DB for a particular frame. Consequently, a video's syntactic and semantic description can be exploited when executing an IR task. Assuming a semantically indexed video DB, a particular video can be regarded as a query to the DB where videos with similar semantics are answers to the initial query.

Having a chain of semantically connected videos allows us to agglomerate semantic knowledge and consequently, derive new knowledge from existing knowledge. Regarding an individual video as a knowledge base allows a user to query this knowledge DB. For example, a user might query a video about its temporal and spatial progression, i.e. whether a video conforms to a particular abstract plot. The proposed approach is advantageous in the context of IR regarding image and video databases and exploitation of image and video semantical richness. For example, semantically enriched images and videos may be analysed with respect to oddities or outliers, i.e. whether the concept usual/normal can be logically inferred from the semantics of an image. This may be beneficial in the field of intrusion detection. Other possible applications of this approach could be indexing to support IR among similar content or automatic narration and summarisation of video data to assist visually or hearing impaired people. Clearly, enabling access to and integration with information stored in visual content stores may enhance today's search experience, which is mostly perceived as a text box accepting rather simple queries. Organising and providing access to information using a wholly new metaphor—image—can provide an advancement to those who prefer visual information. Drawbacks of the proposed system include, limited set of predicates to describe relations among objects, a limited set of characteristics to describe a scene, the manual addition of semantics with respect to non-trivial relationships between frame constituents, the manual maintenance of frames and associated knowledge DB, and missing evaluation on feasibility of feature extraction and semantically motivated retrieval of videos.

Once a proper means to represent knowledge about some domain is found, one is left with actually devising and filling a knowledge DB with facts and relations about that particular

domain. Given the complexity of the domain or task to model, this undertaking can be time-consuming and challenging. For this, Baral et al. propose an automatic approach to acquire domain-specific knowledge from natural language input (Baral et al. 2007). Knowledge is represented in FOL form – AnsProlog clauses. A domain-specific knowledge DB is built from natural language input defining contextual knowledge in five steps. Firstly, natural language text is parsed using CCG parser. This produces a parse tree in FOL and one clause for each word in the input text. Secondly, semantic analysis is performed on the parse tree and word-specific clauses using Boxer. Boxer uses provided input to create discourse representation structures and box representations of discourse representation theory. Thirdly, using special rules (e.g. remove all quantifiers) Boxer output is translated into FOL – AnsProlog facts. Lastly, if necessary, rules can be added manually to the AnsProlog DB. This yields the final knowledge DB.

The authors evaluated their approach for automatically acquiring pragmatics in the domain of simple puzzles. A simple puzzle with four domains and five possibilities per domain is captured as an Answer Set Prolog knowledge DB. Given the puzzle's knowledge DB, the system is queried about a possible target state that satisfies all the puzzle's constraints, i.e. a solution to the puzzle. The authors evaluated the system with regard to textual entailment. That is, given a natural language text, does a certain hypothesis logically follow from knowledge embedded in that text. Finally, the proposed approach was evaluated with regard to answering questions. For example, given a story in natural language about some person's travel, the system is asked a series of questions regarding possible states of that travel.

Tackling the problem of knowledge acquisition is very important since a major aspect of knowledge-deep NLP systems is an appropriate knowledge base. Using natural language texts as basis for automatic knowledge acquisition recognises the fact that huge amounts of knowledge already exist and they only need to be identified and processed. Unfortunately, the approach has some limitations, which are mainly related to limitations inherent in tools like CCG and Boxer. Other limitations include the manual inspection and alteration of the generated knowledge DB. Although the approach was evaluated in the context of toy domains, evaluation revealed that manual interference was necessary even in a small and limited environment. Also, the system infers very specific knowledge from given input. This specific knowledge is not appropriate, i.e. fails to capture or to reason about general concepts of a particular domain.

9 Lexicon

Languages are not only a means to communicate with others but also to seize and understand reality. The structure of knowledge and thought is deeply connected with linguistic structure (Nugues 2006; Russell and Norvig 2003; Majumdar et al. 2008). To enable communication, language has to be shared by a group or community of people. A *lexicon* captures the perception of reality of speakers of the language underlying that lexicon. A lexicon, often spanning a particular *domain*, is a list of words – *lexemes*. It can be divided into *parts of speech* (POS), i.e. classes whose words share common grammatical properties. POS can be clustered into the closed and open class.

Open word classes, are classes which constantly acquire new members as opposed to the closed word classes (e.g. "internet" member of the noun class), which if at all acquire new members rather rarely (e.g. "he" member of the personal pronoun class).

WordNet, developed by Princeton University, is a lexicon for organising English words and has served as a model to mechanically organise other languages like German

(Clough and Stevenson 2004). Presently, it contains more than 200,000 words (Princeton University 2009). WordNet has been used for NLP tasks like cross-language IR (CLIR) (Clough and Stevenson 2004). It arranges words or word forms along with word meanings into a lexical matrix. A row in the matrix defines a set of synonymous words -fb Synset. Synset is the core concept of WordNet. It represents concepts and knowledge that map onto words. A column of the lexical matrix captures different meanings of a word form. WordNet establishes various semantic relations between words like antonymy or homonymy. WordNet represents only open class words, such as nouns, verbs, adjectives and adverbs. Nouns are partitioned into twenty-five primitive concepts, such as natural object or time and top divisions like state or event. Adjectives are divided into descriptive, relational or color adjectives. Descriptive adjectives modify a noun (e.g., hot and cold). Relational adjectives are modified nouns having the syntactic properties of adjectives. For example, dental is a descriptive adjective. Verbs are grouped according to two principles namely (1) 15 categories (e.g., verbs of size and feeling) and (2) entailment. For example, to sleep is an entailment of to snore.

Witzig describes predicates to query WordNet's Prolog DB (Witzig 2003). She elaborates on how WordNet's extensive knowledge base may negatively influence query response time and testability of NLP applications. She highlights strategies to alleviate this situation, namely using Prolog's `index/1` clause to index a predicate on its arguments, rearranging predicates to help Prolog's inference algorithm and only using a representative subset of WordNet. Witzig mentions that these optimisation strategies improved query response time by a factor of nine. Unfortunately, neither the type of queries used for evaluation nor the exact statistics of the evaluation are presented to strengthen her argument. But Witzig's discussion aids Prolog developers new to WordNet to gain an understanding for its functionality, advantages and disadvantages. This helps to lower the steepness of the learning curve required to familiarise oneself with WordNet's extensive functionality, which is spread across several Prolog modules.

Dias-da-Silva et al. discuss how WordNet has been used to develop a prototypical version of WordNet for Brazilian Portuguese (BP)—the BP thesaurus lexical DB (TDb) (Dias-da-Silva et al. 2002). TDb was compiled semi-automatically from a reference corpus consisting of seven BP dictionaries. When necessary due to ambiguity, classification of words was done manually by domain experts. Predefined criteria for extraction and filtering of data guaranteed a minimum level of consistency of TDb data. TDb follows WordNet's core design principles, such as Synset, lexical matrix and the differential method (Nugues 2006; Dias-da-Silva et al. 2002). Words and their relations are implemented as two lists, namely Entry List (EL) and Set List (SL). EL stores words in alphabetical order and SL manages Synsets. Syntactical and semantical relationships between words are modeled using memory pointers on SL and EL. Although TDb is implemented atop of a relational DB management system (RDMBS), relations can be converted to Prolog using compound terms (Pereira and Shieber 1987).

TDb is not a feature-complete adaptation of WordNet but a basis from which the authors wish to design the BP WordNet (BP-WordNet). Dias-da-Silva et al. envision BP-WordNet supporting even more cross categorical relations than WordNet and wish to integrate BP-WordNet with other languages. TDb is a first attempt to provide a lexical DB for BP. TDb's ensured minimum level of consistency and accessibility through a scalable RDBMS can be beneficial for many NLP tasks, such as CLMT or IR. Also, lessons learned from the design and implementation of TDb may be used to guide the creation of the lexicon in general and the BP lexicon in particular. TDb's limited set of functionality and its exposure through an additional layer of complexity, i.e. RDBMS, could reduce user-adoption.

Hettige and Karunananda mention that the quality of a CLMT largely depends on background knowledge to resolve ambiguity between source and target language concepts (Hettige and Karunananda 2007). Regarding English to Sinhala MT, the authors created six dictionaries as a set of Prolog DBs to provide background knowledge. To effectively design and implement these dictionaries, Hettige and Karunananda analysed the English and Sinhala language with respect to POS and morphology. Dictionaries were created from the Malalasekra English to Sinhala bilingual dictionary. The six dictionaries are English/ Sinhala word dictionaries, English/ Sinhala concept dictionaries, English-Sinhala bilingual dictionary and Sinhala rule dictionary. Word dictionaries contain English/ Sinhala words and lexical information about them. Concept dictionaries represent relations among words like synonymy and antonymy. The bilingual dictionary maps an English word to a Sinhala base word and vice versa. The rule dictionary contains morphological rules to allow for inflection of Sinhala words. At the time of writing, the dictionaries are rather limited in terms of words, concepts and rules. Another limitation of this approach is the missing integration with existing WordNets. However, Hettige and Karunananda's work can be seen as a stepping stone, i.e. the basis from which languages of the same family of languages can be lexicalised. For example, instead of devising the same set of dictionaries to map a language L onto English, it might be more advantageous to map L onto the linguistically closer Sinhala language, thereby maybe exploiting shared concepts, words or grammatical features. By mapping L onto Sinhala and Sinhala onto English, the L to English mapping is realised implicitly. Consequently, the coverage of rare languages like Sinhala and their representation in computer knowledge bases is one step close to global information accessibility.

The Semantic Web as the intelligent Web, which can be used by intelligent services, has sparked interest in extending existing knowledge bases to the Web. Xiao-xi and Chang-le propose the conversion of WordNet using the Web Ontology Language (OWL) to enable applications and Webservices to leverage knowledge stored in WordNet, and to exploit OWL's built-in reasoning theory and description logics to derive various logical consequences from WordNet's DB (Xiao-xi and Chang-le 2007). The authors extended an existing approach utilising Resource Description Framework (RDF) and OWL.

Xiao-xi and Chang-le's extended system contributed to the existing approach by implementing WordNet version 2.1 features, namely instance hyponym relation, frame- and sentencegroup concepts. The authors developed a metaphor understanding system to evaluate the feasibility of their approach. The system makes use of WordNet's lexical and semantical knowledge to understand and resolve natural language metaphors. Combining WordNet's extensive knowledge base and OWL's inference capabilities is essential to bringing NLP and intelligent software to the Web. Unfortunately, the authors did neither acknowledge already-mentioned shortcomings of WordNet (e.g. missing semantic relations) nor present strategies to overcome them. Evaluation of the feasibility of querying a large knowledge base over the Web, and effectiveness of the presented metaphor resolution system is also absent.

10 NLP applications

This chapter discusses applications, which either make use of NLP functionality built into Prolog or are themselves developed using the Prolog programming language. A survey by Obrst and Jha published in 1997 reports on industry adoption of NLP tools and techniques with special attention to the aircraft manufacturer Boeing (Obrst and Jha 1997).

Current mobile devices (MD) like personal desktop assistants (PDA) mostly deliver functionality, such as email or newspaper access, through touch-based or keyboard-based user

interface (UI) navigation. However, natural language UIs (NUI)s are starting to emerge in various end-consumer products, such as in-car entertainment and navigation systems or voice mail services on mobile phones. Kadous and Sammut propose a NLP interface to expose MD functionality (Kadous and Sammut 2004). The authors developed a CA, called InCA, which acts as an intermediary between the user and the system and its capabilities. User and CA are partners in a human-machine dialogue. The current version of InCA offers three main features. Firstly, InCA offers personal assistant type services, such as news aggregation and email reading. Secondly, InCA is able to receive speaker-dependent spoken natural language input. Lastly, it is able to respond in spoken natural language. Spread of MD and readily-accessible computing power has drastically increased in recent years. Unfortunately, MD do not offer the same processing and I/O power as desktop PCs. Therefore, simple transfer of desktop computer CA to MD may not be successful. For this, the authors implemented InCA via a client-server architecture. The client residing on a PDA, serves as a means to receive user input (e.g. speech) and present system output (e.g. text and speech). The server is responsible for processing input into concrete output, which is transmitted to the client over a network. The InCA server is responsible for three core tasks. Firstly, when presented with spoken natural language, the server has to recognise the spoken content. Secondly, the server has to synthesise speech from text and pass the result back to a client. Thirdly, the server manages the Probot and Coordinator component. InCA uses IBM ViaVoice and IBM ViaVoice TTS for SR and TTS tasks. Coordinator is responsible for retrieving user-requested information (e.g. news, email) from various sources, using standard protocols like RSS, POP3 or SOAP.

Probot manages the machine-user dialogue. It is implemented as a rule-based system embedded in a Prolog interpreter. Probot rules consist of patterns and responses. The motivation behind rules is, if user input (e.g. sentence) matches some rule's pattern, Probot outputs the rule's response (e.g. output sentence). Probot rules can be refined using Prolog clauses. This refinement can be seen as side effect executed if a rule fires or as a constraint imposed on some rule. Rules are grouped into contexts (e.g. email, translation). Probot uses special rules called Filters to determine or change context based on user input. Provided with some user input, Filters restrict the rule search space and add flexibility, in that Probot may change between contexts to satisfy a users enquiry. If no Filters or rules match a given user input, Probot resorts to Backup rules. Backup rules improve resilience to noisy input data. Also, Backup rules enable InCA to act more like a conversational partner insofar as if InCA is not able to compute the answer (cannot be inferred from its knowledge base) to a user's enquiry, it may politely indicate its inability and possibly ask the user to refine her initial question.

The extension of CAs to MDs is reliant upon computing power on MDs increasing steadily, the ubiquitousness of the Web expanding, and demand for access to information anytime and anywhere being satisfied. With the recent trend towards cloud-based/ utility computing, enormous amounts of computing power become available to the individual on a "use and pay on demand" (e.g. transactional) basis. This in turn has enabled software developers to exploit the client-server principle (e.g. data local to where it is processed) on a large scale. Clearly, InCA would strongly benefit from the cloud-based computing paradigm, as it means potentially infinite computing resources and high availability on the processing side while being able to serve commodity MDs on the client side. InCA acknowledges that different people have different preferences regarding human-machine interaction and information delivery (e.g. audible, legible). Its resilience to noisy input and resource-efficiency take the operational environment of MDs into account. A drawback of InCA is its reliance on a network connection. Current features of InCA address typical PDA functionality like information aggregation from various sources (e.g. email, news). Clearly, extending InCA to serve as a NUI to more advanced functionality, such as location-based or social network-based services (e.g. where

is X, who is close to Y) is needed to transform the MD of today into a true personal assistant. Also, it is often argued that a pure rule-based approach makes modeling NLP systems too rigid.

The second application discussed, named NL-Ue, aims to reduce the complexity of and knowledge needed to access and use information systems like RDBMS (Minock 2008; Fodor 2008). NL-Ue, developed by Universidade de Évora (UE), is a natural language interface to the UE Integrated Information System (IIS-UE) for students, teachers and staff of UE (Quintano and Rodrigues 2006). Users of NL-Ue may query IIS-UE using natural language questions like "What teacher teaches the Management course?". Natural language questions will be translated into a RDBMS query returning the appropriate answer from IIS-UE databases.

NL-Ue uses Constraint Logic Programming (CLP) and the GNU Prolog compiler to transform natural language queries into FOL clauses and subsequently into RDBMS queries. The system's architecture is comprised of five core components interacting with each other through a set of well-defined APIs. NL-Ue answers a user's enquiry for a specific resource or entity (e.g. name of a teacher, course) through a series of questions and answers. It identifies and resolves potentially ambiguous user questions by means of clarification requests.

To answer a user question, NL-Ue translates the question into FOL clauses using the so called VISL syntactic parser and a custom built semantic parser. To be able to query information stored inside some DB, the DB is translated into its FOL representation using ISCO. Afterwards, NL-Ue can try to unify a user's query with knowledge about the DB using Prolog's inference mechanism. The inference process is guided by a set of control rules. Control rules are generic, i.e. independent of IIS-UE, and provide structural knowledge about DB entities, DB data and how to map them to a user request. For example, a noun rule may try to map a noun within a question to a table of IIS-UE DBs.

If a user request is ambiguous or the response to a request is ambiguous, NL-Ue needs to resolve this ambiguity. A user query is ambiguous, if it is not sufficiently specific, i.e. does not lead to an answer. The response to a query is ambiguous, if it were one of several answers satisfying the query. NL-Ue resolves ambiguity via the mentioned clarification requests. That is, it uses the current context of a query and asks a user specific questions to constrain the solution search space until a single answer fulfills the initial query and imposed constraints. NL-Ue identifies what questions to ask using manually derived heuristics.

NL-Ue enables users to navigate structurally complex and data-rich information systems using natural language without requiring users to know a system's underlying implementation. This enables users to search and find information more quickly and easily. NL-Ue is a step towards managing information sources using natural language, thereby reducing the need for DB administrators and users to learn DB domain specific languages like SQL. General control rules and CLP to represent knowledge make NL-Ue domain independent. Unfortunately, NL-Ue's capability with respect to dialogue management is rather limited. A user still needs some knowledge about the mini-world contained in some DB because NL-Ue does not offer facilities to report about the domain it governs. The need to compile a DB into FOL representation upfront can be cumbersome. Other potential weaknesses of NL-Ue include manually derived heuristics to obtain solution space constraining questions, and the order in which constraining questions are asked is arbitrary and if many equally likely constraining questions are derived the question/answer dialogue might comprise many iterations. A potentially exciting extension of NL-Ue would be integration with MD, such as integration with InCA. This would allow UE staff and students to access UE-specific information (e.g. office hours, course information) anytime and anywhere. Also, adding speech support to NL-Ue would be beneficial for people who suffer from reading disabilities or may be sight impaired or blind.

11 Conclusions

This report has presented a high level introduction to the diverse field of NLP with a special focus on the Prolog programming language. NLP is a vibrant field of interdisciplinary CS research seeking to build conversational intelligence into software systems. Although NLP has evolved considerably within recent years, it is still far from reaching its goal set out in the midst of the last century.

It has been argued that Prolog offers advantages with respect to problem-solving in the AI and NLP domain. Still, Prolog is flexible enough to be used as a general purpose programming language. Well-known program structuring techniques like Refactoring and object-oriented programming can and have been extended to Prolog to enable development of large and complex software systems (Schrijvers and Serebrenik 2004; Bratko 2000). The report has discussed components of current NLP systems developed in Prolog. It has been shown that Prolog has not only been used to build individual NLP components like parsers but also complete NLP systems. No matter what the programming language or problem-solving methodology of choice, NLP is still an open, diverse and challenging field of CS research waiting for new contributions.

References

Abou-Assaleh T, Cercone N, Kešelj V (2003) Expressing probabilistic context-free grammars in the relaxed unification formalism. In: Conference Pacific association for computational linguistics, PACLING'03, Halifax, Nova Scotia, Canada, August 2003. Dalhousie University, pp 29–36

Alexin Z, Csirik J, Gyimóthy T, Jelasity M, Tóth L (1997) Learning phonetic rules in a speech recognition system. In: Inductive logic programming, vol 1297. Springer-Verlag, Berlin, Heidelberg, pp 35–44

Barták R (1998) Guide to prolog programming. http://ktiml.mff.cuni.cz/~bartak/prolog/implementations.html; last visited 26.09.2009

Baral C, Dzifcak J, Tari L (2007) Towards overcoming the knowledge acquisition bottleneck in answer set prolog applications: embracing natural language inputs. In: Logic programming, vol 4670. Springer-Verlag, Berlin, Heidelberg, pp 1–21

Bratko I (2000) Prolog programming for artifical intelligence. International computer science series. 3. Addison-Wesley Longman, Amsterdam

Christiansen H (2002) Logical grammars based on constraint handling rules. In: 18th international conference on logic programming. Springer-Verlag, p 481

Clough P, Stevenson M (2004) Cross-language information retrieval using eurowordnet and word sense disambiguation. In: Advances in information retrieval, vol 2997. Springer-Verlag, Berlin, Heidelberg, pp 327–337

Convington MA (2003) Et: an efficient tokenizer in iso prolog. Technical report, The University of Georgia, February

Convington MA (2003) A free-word-order dependency parser in prolog. Technical report, The University of Georgia

Cook JJ (2004) P#: a concurrent prolog for the .net framework. Softw: Pract Exp 34(9):815–845

Dias-da-Silva BC, de Oliveira MF, de Moraes HR (2002) Groundwork for the development of the brazilian portuguese wordnet. In: Advances in natural language processing, vol 2389. Springer-Verlag, Berlin, Heidelberg, pp 179–192

Fodor P, Lally A, Ferrucci D (2008) The prolog interface to the unstructured information management architecture. CoRR, abs/0809.0680

Grune D, Jacobs CJH (2008) Parsing techniques a practical guide, monographs in computer science. 2. Springer Science+Business Media LLC, New York

Hettige B, Karunananda AS (2007) Developing lexicon databases for english to sinhala machine translation. Industrial and Information Systems, 2007. ICIIS 2007. International Conference on 9(11):215–220

Hodroj AM (2006) Prolog. NET manual. http://prolog.hodroj.net/index.html

Hu C (2003) Text statistics tool box for natural language processing. Technical report, The University of Georgia, May

Jurafsky D, Martin JH (2008) Speech and language processing, prentice hall series in artificial intelligence. 2. Prentice Hall, NJ
Kadous MW, Sammut C (2004) Inca: a mobile conversational agent. In: PRICAI 2004: trends in artificial intelligence. Springer-Verlag Berlin, Heidelberg, pp 644–653
Kiraz GA (2000) Multitiered nonlinear morphology using multitape finite automata: a case study on syriac and arabic. Comput Linguist 26(1):77–105
Kiraz GA, Grimley-Evans E (1998) Multi-tape automata for speech and language systems: a prolog implementation. In: Automata implementation, vol 1436. Springer-Verlag, Berlin, Heidelberg, pp 87–103
Liang CC (2002) Compiler construction in higher order logic programming. In: Practical aspects of declarative languages, vol 2257. Springer-Verlag, Berlin, Heidelberg, pp 47–63
Majumdar A, Sowa J, Stewart J (2008) Pursuing the goal of language understanding. In: ICCS '08: Proceedings of the 16th international conference on conceptual structures. Springer-Verlag, Berlin, Heidelberg, pp 21–42
Manning CD, Schütze H (1999) Foundations of statistical natural language processing. The MIT Press, Cambridge, MA
McClain JT (2003) Swi-speech: an interface between swi-prolog and microsoft sapi. Technical report, The University of Georgia, May
McConnell S (2004) Code complete: a practical handbook of software construction. 2. Microsoft Press Corp., USA
Minock M, Peter Olofsson P, Näslund A (2008) Towards building robust natural language interfaces to databases. In: Natural language and information systems, vol 5039. Springer-Verlag, Berlin, Heidelberg, pp 187–198
Nakache D, Metais E, Timsit JF (2005) Evaluation and nlp. vol 3588. Springer-Verlag, Berlin, Heidelberg, pp 626–632
Nugues PM (2006) An introduction to language processing with Perl and Prolog: an outline of theories, implementation, and application with special consideration of English, French, and German cognitive technologies. 1. Springer-Verlag, New York
Obrst L, Jha KN (1997) Nlp and industry: transfer and reuse of technologies. In: Workshop On from research to commercial applications: making NLP work in practice, pp 57–64
Pang B, Lee L (2008) Opinion mining and sentiment analysis. 2. Now Publishers Inc, Hanover, MA, USA
Pereira FCN, Shieber SM (1987) Prolog and natural-language analysis—digital edition cognitive technologies. Microtome Publishing, Brookline, Massachusetts, July
Peter Lodewijk Van Roy (1991) Can logic programming execute as fast as imperative programming? PhD thesis, University of California at Berkeley, Computer Science Division
Princeton University. Wordnet a lexical database for the english language. http://wordnet.princeton.edu; last visited 02.03.2009
Quintano L, Rodrigues I (2006) Using a logic programming framework to control database query dialogues in natural language. In: Logic programming, vol 4079. Springer-Verlag, Berlin, Heidelberg, pp 406–420
Russell S, Norvig P (2003) Artificial intelligence—a modern approach Prentice Hall Series in artificial intelligence. 2. Prentice Hall International, NJ
Schlachter JG (2003) Pronto morph: morphological analysis tool for use with pronto (prolog natural language toolkit). Technical report, The University of Georgia, May
Schneider G (2003) Learning to disambiguate syntactic relations. Linguist online 17:117–136
Schrijvers T, Serebrenik A (2004) Improving prolog programs: Refactoring for prolog. In: Logic programming, vol 3132. Springer-Verlag, Berlin, Heidelberg, pp 58–72
Tascini G, Montesanto A, Palombo R, Puliti P (2001) Behind the image sequence: The semantics of moving shapes. In: Visual form 2001, vol 2059. Springer-Verlag, Berlin, Heidelberg, pp 619–629
Thurston A (2007) Generalized parsing techniques for computer languages. Technical report, School of Computing, Queen's University, April
Truszczyński M (2007) Logic programming for knowledge representation. In: Logic programming, vol 4670. Springer-Verlag, Berlin, Heidelberg, pp 76–88
Voss M (2004) Improving upon earley's parsing algorithm in prolog. Technical report, The University of Georgia, May
Witzig S (2003) Accessing wordnet from prolog. Technical report, The University of Georgia, May
Xiao-xi H, Chang-le Z (2007) An owl-based wordnet lexical ontology. In: Journal of Zhejiang University—Science A, vol 8. Zhejiang University Press, co-published with Springer-Verlag GmbH, pp 864–870, May
Zadeh LA (1965) Fuzzy sets. Inf Control 8(3):338–353