Defining a Core Body of Knowledge for the Introductory Computational Linguistics Curriculum

Steven Bird
Department of Computer Science and Software Engineering
University of Melbourne, Victoria 3010, Australia
sb@csse.unimelb.edu.au

Abstract

Discourse in and about computational linguistics depends on a shared body of knowledge. However, little content is shared across the introductory courses in this field. Instead, they typically cover a diverse assortment of topics tailored to the capabilities of the students and the interests of the instructor. If the core body of knowledge could be agreed and incorporated into introductory courses several benefits would ensue, such as the proliferation of instructional materials, software support, and extension modules building on a common foundation. This paper argues that it is worthwhile to articulate a core body of knowledge, and proposes a starting point based on the ACM Computer Science Curriculum. A variety of issues specific to the multidisciplinary nature of computational linguistics are explored.

1 Introduction

Surveys of introductory courses in computational linguistics and natural language processing provide evidence of great diversity.[1] Regional variation is stark: courses may emphasise theory over programming (or vice versa), rule-based approaches over statistical approaches (or vice versa), tagging and parsing over semantic interpretation (or vice versa), and so on. The diversity is hardly surprising given the particular background of a student cohort and the particular expertise of an instructor.

In spite of this variation, the introductory course needs to serve some common, basic needs. For some students, it will be the first step in a pathway leading to specialised courses, graduate research, or employment in this field. These students should receive a solid technical foundation and should come away with an accurate picture of the many opportunities that lie ahead. For students who do not continue, the introductory course will be their main exposure to the field. In addition to the technical content, these students need to understand how the field draws from and contributes back to its parent disciplines of linguistics and computer science, along with technological applications that are helping to shape the future information society. Naturally, this course is also a prime opportunity to promote the field to newcomers and encourage them to pursue advanced studies in this area. In all cases, the introductory course needs to cover a core body of knowledge.

The fact that a core body of knowledge exists in computational linguistics is demonstrated anecdotally: a doctoral student is told to curtail her extended discussions of basic POS tagging and CFG parsing algorithms since they are part of the presumed knowledge of the audience; a researcher presenting work to a general linguistics or computer science audience discovers to his surprise that certain methodologies or algorithms need to be explicated and defended, even though they was uncontroversial when presented at a conference; a keynote speaker at a computational linguistics conference can presume that certain theoretical programs and practical goals of the field are widely accepted. These three areas – terminology, methodology, ideology – constitute

[1] http://aclweb.org/aclwiki/index.php?title=List_of_NLP/CL_courses

Proceedings of the Third Workshop on Issues in Teaching Computational Linguistics (TeachCL-08), pages 27–35,
Columbus, Ohio, USA, June 2008.

part of the core body of knowledge of computational linguistics. They provide us with the starting point for identifying the concepts and skills to be covered in the introductory course.

Identifying a core body of knowledge would bring three major benefits. First, technical support would be consolidated: instructional materials together with implementations of standard algorithms would be available in several programming paradigms and languages. Second, colleagues without a research specialization in computational linguistics would have a non-controversial curriculum with external support, a standard course that could be promoted to a broad range of students as a mainstream option, in both linguistics and computer science. Similarly, new graduates beginning a teaching career would be better equipped to push for the adoption of a new computational linguistics or natural language processing course at institutions where it is not yet established. Third, employers and graduate schools would be able to make assumptions about the knowledge and skills of a new graduate.

The goal of this paper is to advocate the idea of consensus around a body of knowledge as a promising way to coordinate the introductory computational linguistics curriculum, without attempting to mandate the structure of individual courses or the choice of textbooks. The paper is organised as follows: section 2 sets the scene by describing a variety of contexts in which computational linguistics is taught, drawing on the author's first-hand experience; section 3 sets out a possible organization for the introductory topics in computational linguistics, modelled on the ACM Computer Science Curriculum; section 4 explores some implications of this approach for curriculum and assessment. The paper closes with remarks about next steps.

2 Contexts for Teaching and Learning in Computational Linguistics

In this section a variety of scenarios are described in which the author has had direct first-hand experience. All cases involve entry-level courses in computational linguistics. They provide the back-drop to the current proposal, exemplifying a range of contexts in which a core body of knowledge would need to be delivered, contexts imposing different constraints on implementation.

Before embarking on this discussion it is helpful to be reminded of the differing backgrounds and goals of new students. Some want to use computational techniques in the analysis of language, while others want to use linguistic knowledge in the development of language technologies. These backgrounds and goals are orthogonal, leading to the grid shown in Table 1.

I will begin with the most common context of a graduate-level course, before progressing to upper-level undergraduate, lower-level undergraduate, and secondary levels.

2.1 Graduate-Level Courses

Dozens of graduate programs in computer science and in linguistics have an introductory course on computational linguistics or natural language processing. In most cases, this is all the formal training a student will receive, and subsequent training is happens in private study or on the job. In some universities this is the entry point into a suite of more advanced courses in such areas as lexical semantics, statistical parsing, and machine translation. Even so, it is important to consider the shared assumptions of these specialised courses, and the needs of a student who only undertakes the introductory course.

There are two principal challenges faced by instructors at this level. The first is to adequately cover the theoretical and practical sides of the field in a single semester. A popular solution is not to try, i.e. to focus on theory to the exclusion of practical exercises, or to simply teach "programming for linguists." The former deprives students of the challenge and excitement of writing programs to automatically process language. The latter fails to cover any significant domain-specific theories or algorithms.

The second challenge is to address the diverse backgrounds of students, ranging from those with a computer science background to a linguistics background, with a scattering of students who have a background in both or in neither.

The author taught at this level at the University of Pennsylvania over a period of three years. Perhaps the most apt summary of the experience is *triage*. Cohorts fell into three groups: (i) students

	Background: Arts and Humanities	Background: Science and Engineering
Language Analysis	Programming to manage language data, explore linguistic models, and test empirical claims	Language as a source of interesting problems in data modeling, data mining, and knowledge discovery
Language Technology	Knowledge of linguistic algorithms and data structures for high quality, maintainable language processing software	Learning to program, with applications to familiar problems, to work in language technology or other technical field

Table 1: Summary of Students' Backgrounds and Goals, from (Bird et al., 2008a)

who are well prepared in either linguistics or computer science but not both (the majority) who will perform well given appropriate intervention; (ii) students who are well-prepared in both linguistics and computer science, able to complete learning tasks on their own with limited guidance; and (iii) students with minimal preparation in either linguistics or computer science, who lack any foundational knowledge upon which to build. Resources targetted at the first group invariably had the greatest impact.

2.2 Specialised Upper-Level Undergraduate Courses

In contrast with graduate-level courses, a specialised upper-level undergraduate course will typically be an elective, positioned in the later stages of an extended sequence of courses (corresponding to ACM unit *IS7 Natural Language Processing*, see §3). Here it is usually possible to make reliable assumptions about background knowledge and skills, and to provide training that is pitched at exactly the right level.

The author taught at this level in the Computer Science and Linguistics departments at the University of Melbourne during the past five years (five times in Computer Science, once in Linguistics). In the Linguistics department, the course began by teaching programming, with illustrations drawn from linguistic domains, before progressing to topics in text processing (tokenization, tagging), grammars and parsing, and data management. Laboratory sessions focussed on the acquisition of programming skills, and we found that a 1:5 staff-student ratio was insufficient.

In the Computer Science department, the first approach was to introduce linguistics for 2-3 weeks before looking at algorithms for linguistic processing. This was unpopular with many students, who did not see the motivation for learning about such topics as morphology and verb subcategorization in isolation from practical applications. A revised version of the course opened with topics in text processing, including tokenization, extracting text from the web, and moving on to topics in language engineering. (Bird et al. (2008b) provide a more extended discussion of opening topics.)

A third option is to teach computational linguistic topics in the context of a specialised course in an allied field. Thus a course on morphology could include a module on finite-state morphology, and a course on machine learning could include a module on text mining. In the former case, a linguistic domain is presupposed and the instructor needs to teach the linguist audience about a particular corpus to be processed or an algorithm to be implemented or tested. In the latter case, a family of algorithms and data structures is presupposed and the instructor needs to teach a computer science audience about linguistic data, structures, and processes that can serve as a domain of application.

2.3 Cross-Disciplinary Transition

People entering computational linguistics from either a linguistics or computer science background are faced with a daunting challenge of learning the fundamentals of the other field before they can progress very far with the study of the target domain. A major institution with a long history of teaching computational linguistics will have a cadre of graduate students and post-doctoral researchers who can support an instructor in teaching a course. However, one measure of the success of the approach being advocated here are that such institutions will be in the minority of those where computational linguistics is taught. In such contexts, a computational linguistics course will be

a lone offering, competing for enrolments with a variety of more established electives. To compound the problem, a newcomer to the field may be faced with taking a course in a department other than their host department, a course which presumes background knowledge they lack. Additional support and self-paced learning materials are crucial. Efforts on filling out the computational linguistics content in Wikipedia – by instructors and students alike – will help the entire community.

2.4 Lower-Level Undergraduate Courses

An intriguing option for delivery of an introduction to computational linguistics is in the context of entry-level courses in linguistics and computer science. In some cases, this may help to address the declining interest of students in these individual disciplines.

As computer science finds a broader role in service teaching, rather than in training only those students doing a major, the curriculum needs to be driven by topics of broad appeal. In the author's current first year teaching, such topics include climate change, population health, social anthropology, and finance. Many fundamental concepts in data structures and algorithms can be taught from such starting points. It is possible to include language processing as one of the drivers for such a course.

Many possibilities for including computational linguistics exist in the second-level computer science curriculum. For example, algorithmic methods involving time-space trade-offs and dynamic programming can be motivated by the task of building a simple web search engine (Bird and Curran, 2006). Concrete tasks involve web crawling, text extraction, stemming, and indexing. Spelling correction can be used as a driver for teaching core computer science concepts in associative arrays, linked lists, and sorting by a secondary key.

An analogous opportunity exists in the context of entry-level courses in linguistics. Linguistics students will readily agree that most human knowledge and communication is represented and expressed using language. But it will come as a surprise that language technologies can process language automatically, leading to more natural human-machine interfaces, and more sophisticated access to stored information. In this context, a linguistics student may grasp a broader vision for his/her role in the multilingual information society of the future.

In both cases, the hope is that students are inspired to do further undergraduate study spanning linguistics and computer science, and to enter industry or graduate school with a solid preparation and a suitable mix of theoretical knowledge and technical skills.

The major obstacle is the lack of resources available to the typical instructor, who is not a specialist in computational linguistics, and who has to deliver the course to a large audience having no prior interest or knowledge in this area. They need simple packages and modules that can be incorporated into a variety of teaching contexts.

2.5 Secondary School

Programming and Information Technology have found a place in the secondary curriculum in many countries. The coursework is typically animated with projects involving games, databases, and dynamic websites. In contrast, the curriculum involving the grammar and literature of a major world language typically only uses information technology skills for such mundane tasks as word processing and web-based research. However, as innovators in the language curriculum look for new ways to enliven their classes with technology, computational linguistics offers a ready-made source of interesting problems and methods.

In Australia, the *English Language* curriculum of the Victorian Certificate of Education is a linguistics program offered as part of the last two years of secondary education (VCAA, 2006; Mulder et al., 2001). This course provides a promising host for computational linguistics content in the Victorian secondary curriculum. The author has delivered an "Electronic Grammar" module[2] in an English class in a Victorian secondary school over a three week period, jointly with a teacher who has a double degree in linguistics and computer science. Students were taught the elements of programming together with some simple applications involving taggers, parsers and annotated corpora. These activities served to reinforce students' understanding of lexical categories, lexical semantics, and syntactic

[2] `http://nltk.org/electronic_grammar.html`

ambiguity (i.e. prepositional phrase attachment). Similar methods could be applied in second language learning classes to locate common words and idioms in corpora.

In this context, key challenges are the installation of specialised software (even a programming language interpreter), overcoming the impenetrable nature of standard part-of-speech tagsets by mapping them to simplified tagsets, and providing suitable training for teachers. A promising solution is to provide a self-paced web-based programming and testing environment, side-stepping issues with school infrastructure and teacher training.[3]

3 Defining the CL Body of Knowledge

A promising approach for identifying the CL body of knowledge is to begin with the ACM *Computing Curricula 2001 Computer Science Volume* (ACM, 2001). In this scheme, the body of knowledge within computer science is organised in a three-level hierarchy: subfields, units and topics. Each subfield has a two-letter designator, such as OS for operating systems. Subfields are divided into several units, each being a coherent theme within that particular area, and each identified with a numeric suffix. Within each unit, individual topics are identified. We can select from this body of knowledge the areas that are commonly assumed in computational linguistics (see the Appendix), and then expect them to be part of the background of an incoming computer science student.

The field of linguistics is less systematised, and no professional linguistics body has attempted to devise an international curriculum standard. Helpful compendia of topics exist, such as the *Language Files* (Stewart and Vaillette, 2008). However, this does not attempt to define the curriculum but to provide supporting materials for introductory courses.

Following the ACM scheme, one could try to establish a list of topics comprising the body of knowledge in computational linguistics. This is not an attempt to create a comprehensive ontology for the field (cf. Cole (1997), Uszkoreit et al. (2003)), but rather a simple practical organization of introductory topics.

[3]This is a separate activity of the author and colleagues, available via `ivle.sourceforge.net`

CL. Computational Linguistics

CL1. Goals of computational linguistics
roots, philosophical underpinnings, ideology, contemporary divides

CL2. Introduction to Language
written vs spoken language; linguistic levels; typology, variation and change

CL3. Words, morphology and the lexicon
tokenization, lexical categories, POS-tagging, stemming, morphological analysis, FSAs

CL4. Syntax, grammars and parsing
grammar formalisms, grammar development, formal complexity of natural language

CL5. Semantics and discourse
lexical semantics, multiword expressions, discourse representation

CL6. Generation
text planning, syntactic realization

CL7. Language engineering
architecture, robustness, evaluation paradigms

CL8. Language resources
corpora, web as corpus, data-intensive linguistics, linguistic annotation, Unicode

CL9. Language technologies
named entity detection, coreference, IE, QA, summarization, MT, NL interfaces

Following the ACM curriculum, we would expect to designate some of these areas as core (e.g. CL1-3), while expecting some number of additional areas to be taken as electives (e.g. three from the remaining six areas). A given curriculum would then consist of three components: (a) bridging studies so students can access the core knowledge; (b) the core body of knowledge itself; and (c) a selection of electives chosen to give students a balance of linguistic models, computational methodologies, and application domains. These issues involve fleshing out the body of knowledge into a sequential curriculum, the topic of the next section.

4 Implications for the Curriculum

The curriculum of an introductory course builds out from the body of knowledge of the field by linearizing the topic areas and adding bridging studies and electives. The result is a pathway that mediates between students' backgrounds and their goals as already schematised in Table 1. Figure 1 displays two hypothetical pathways, one for students

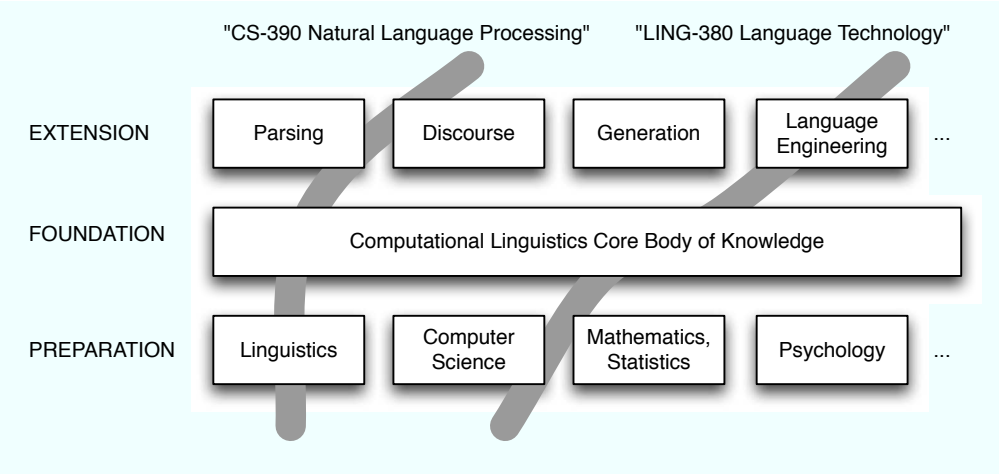

Figure 1: Curriculum as a Pathway Through the Core Body of Knowledge, with Two Hypothetical Courses

from a computer science background wanting to learn about natural language processing, and one for students from a linguistics background wanting to learn about language technology. These could serve as templates for individual advanced undergraduate courses with names that are driven by local marketing needs rather than the need to emphasise the computational linguistics content. However, they could also serve as a guide for a whole series of course selections in the context of a coursework masters program. Clearly, the adoption of a core body of knowledge has rather limited implications for the sequence of an individual curriculum.

This section explores these implications for the curriculum and raises issues for wider discussion and exploration.

4.1 Diverse Entry Points

An identified body of knowledge is not yet a curriculum. It must sit in the context of the background and goals of a particular audience. An analysis of the author's experience in teaching computational linguistics to several types of audience has led to a four-way partitioning of the possible entry points, shown in Figure 2.

The approaches in the top half of the figure are driven by applications and skills, while those in the bottom half are driven by theoretical concerns both inside and outside computational linguistics. The entry points in the top-left and bottom-right of the diagram seem to work best for a computer science audience, while the other two seem to work best for a linguistics audience (though further work is required to put such impressionistic observations on a sound footing).

By definition, all students would have to cover the core curriculum regardless of their entry point. Depending on the entry point and the other courses taken, different amounts of the core curriculum would already be covered. For students with minimal preparation, it might actually take more than one course to cover the core curriculum.

4.2 Bridging Studies

One approach to preparation, especially suitable at the graduate level, is to mandate bridging studies for students who are not adequately prepared for the introductory course. This could range from an individual program of preparatory readings, to a summer intensive course, to a full semester course (e.g. auditing a first or second year undergraduate course such as *Introduction to Language* or *Algorithms and Data Structures*).

It is crucial to take seriously the fact that some students may be learning to program for the first time in their lives. Apart from learning the syntax of a particular programming language, they need to learn a new and quite foreign algorithmic approach to problem solving. Students often report that they understand the language constructs and follow the examples provided by the instructor, but find they are unable to write new programs from scratch.

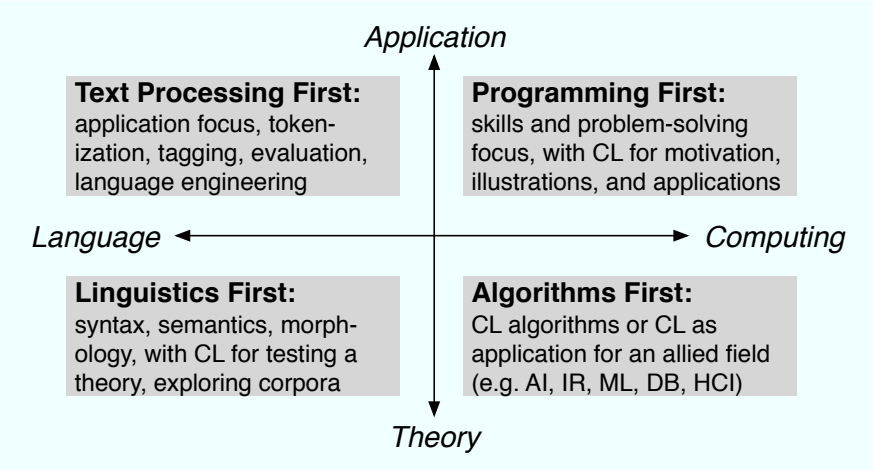

Figure 2: Approaches to Teaching NLP

This accords with the finding that the way in which programmers understand or write programs differs greatly between the novice and the expert (Lister et al., 2006). The issue is independent of the computational linguistics context, and fits the more general pattern that students completing an introductory programming course do not perform as well as expected (McCracken et al., 2001).

Bridging studies can also overlap with the course itself, as already indicated in Figure 1. For example, in the first week of classes one could run a quiz that identifies students who are not sufficiently prepared for the programming component of the course. Such a quiz could include a diagnostic non-programming task, like articulating the search process for looking up a name in a telephone book, which is a predictor of programming skill (Simon et al., 2006). Early intervention could include extra support, readings, classes, and so on. Some students could be alerted to the fact that they will find the course very challenging. Some students in this category may opt to switch to a less demanding course, which might actually be the best outcome for all concerned.

4.3 Organizational Models

Linguistics Model: A natural way to structure the computational linguistics curriculum is to adopt organizational patterns from linguistics courses. This could involve progression up through the linguistic levels from phonology to discourse, or a focus on the analysis of a particular language or language family, the implementation of a particular linguistic theory, or skills development in corpus linguistics or field methods. In this way, content can be packaged to meet local needs, while retaining latitude to enter and exit the core body of knowledge in computational linguistics.

Computer Science Model: The curriculum could adopt organizational patterns from other computer science courses. This could involve progression through families of algorithms, or navigating the processing pipeline of speech understanding, or exploring the pieces of a multi-component system (e.g. question answering). As with the linguistics model, the course would be badged to appeal to students in the local context, while covering the core body of knowledge in computational linguistics.

Vocational Model: In some contexts, established theoretical courses dominate, and there is room to promote a course that is focussed on building programming skills in a new language or for some new application area. This may result in a popular elective that gives students a readily marketable skill.[4] This approach may also work at the secondary level in the form of an after-school club. The course is structured according to the features of a particular programming language, but examples and projects on text processing succeed in covering the core body

[4]The author found this approach to be successful in the case of a database theory course, in which a semester project on building a web database using PHP and MySQL added significant appeal to an otherwise dry subject.

of knowledge in computational linguistics.

Dialectic Model: As discussed above, a major goal for any curriculum is to take students from one of the entry points in Figure 2 into the core body of knowledge. One approach is to consider transitions to topics covered in one of the other entry points: the entry point is a familiar topic, but from there the curriculum goes across to the other side, attempting to span the divide between computer science and linguistics. Thus, a computational linguistics curriculum for a computer science audience could begin with algorithms (bottom-left) before applying these to a range of problems in text processing (top-left) only to discover that richer sources of linguistic knowledge were required (bottom-right). Similarly a curriculum for a linguistics audience could begin with programming (top-right), then seek to apply these skills to corpus processing for a particular linguistic domain (bottom-left).

This last approach to the curriculum criss-crosses the divide between linguistics and computer science. Done well, it will establish a dialectic between the two fields, one in which students reach a mature understanding of the contrasting methodologies and ideologies that exist within computational linguistics including: philosophical assumptions (e.g. rationalism vs empiricism); the measurement of success (e.g. formal evaluation vs linguistic explanation); and the role of observation (e.g. a single datum as a valuable nugget vs massive datasets as ore to be refined).

5 Conclusion

A core body of knowledge is presumed background to just about any communication within the field of computational linguistics, spanning terminology, methodology, and ideology. Consensus on this body of knowledge would serve to underpin a diverse range of introductory curricula, ensuring they cover the core without imposing much restriction on the details of any particular course. Curricula beginning from four very different starting points can progress towards this common core, and thence to specialised topics that maximise the local appeal of the course and its function of attracting newcomers into the field of computational linguistics.

There is enough flexibility in the curriculum of most existing introductory computational linguistics courses to accommodate a core body of knowledge, regardless of the aspirations of students or the research interests of an instructor. If the introductory course is part of a sequence of courses, a larger body of knowledge is in view and there will be scope for switching content into and out of the first course. If the introductory course stands alone as an elective that leads to no other courses, there will also be scope for adding or removing content.

The preliminary discussion of this paper leaves many areas open for discussion and exploration. The analyses and recommendations remain at the level of folk pedagogy and need to be established objectively. The various pathways have only been described schematically, and still need to be fleshed out into complete syllabuses, down to the level of week-by-week topics. Support for skill development is crucial, especially in the case of students learning to program for the first time. Finally, obstacles to conceptual learning and skill development need to be investigated systematically, with the help of more sophisticated and nuanced approaches to assessment.

Acknowledgments

The experiences and ideas discussed in this paper have arisen during my computational linguistics teaching at the Universities of Edinburgh, Pennsylvania and Melbourne. I'm indebted to several co-teachers who have accompanied me on my journey into teaching computational linguistics, including Edward Loper, Ewan Klein, Baden Hughes, and Selina Dennis. I am also grateful to many students who have willingly participated in my explorations of ways to bridge the divide between linguistics and computer science over the past decade. This paper has benefitted from the feedback of several anonymous reviewers.

References

ACM. 2001. *Computing Curricula 2001: Computer Science Volume*. Association for Computing Machinery. http://www.sigcse.org/cc2001/.

Steven Bird and James Curran. 2006. Building a search engine to drive problem-based learning. In *Proceedings of the Eleventh Annual Conference on Innovation and Technology in Computer Science Education*. http://eprints.unimelb.edu.au/archive/00001618/.

Steven Bird, Ewan Klein, and Edward Loper. 2008a. Natural Language Processing in Python. http://nltk.org/book.html.

Steven Bird, Ewan Klein, Edward Loper, and Jason Baldridge. 2008b. Multidisciplinary instruction with the Natural Language Toolkit. In *Proceedings of the Third Workshop on Issues in Teaching Computational Linguistics*. Association for Computational Linguistics.

Ronald Cole, editor. 1997. *Survey of the State of the Art in Human Language Technology*. Studies in Natural Language Processing. Cambridge University Press.

Raymond Lister, Beth Simon, Errol Thompson, Jacqueline L. Whalley, and Christine Prasad. 2006. Not seeing the forest for the trees: novice programmers and the SOLO taxonomy. In *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, pages 118–122.

Michael McCracken, Vicki Almstrum, Danny Diaz, Mark Guzdial, Dianne Hagan, Yifat Ben-David Kolikant, Cary Laxer, Lynda Thomas, Ian Utting, and Tadeusz Wilusz. 2001. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *SIGCSE Bulletin*, 33:125–180.

Jean Mulder, Kate Burridge, and Caroline Thomas. 2001. *Macmillan English Language: VCE Units 1 and 2*. Melbourne: Macmillan Education Australia.

Simon Simon, Quintin Cutts, Sally Fincher, Patricia Haden, Anthony Robins, Ken Sutton, Bob Baker, Ilona Box, Michael de Raadt, John Hamer, Margaret Hamilton, Raymond Lister, Marian Petre, Denise Tolhurst, and Jodi Tutty. 2006. The ability to articulate strategy as a predictor of programming skill. In *Proceedings of the 8th Australian Conference on Computing Education*, pages 181–188. Australian Computer Society.

Thomas W. Stewart and Nathan Vaillette, editors. 2008. *Language Files: Materials for an Introduction to Language and Linguistics*. Ohio State University Press.

Hans Uszkoreit, Brigitte Jörg, and Gregor Erbach. 2003. An ontology-based knowledge portal for language technology. In *Proceedings of ENABLER/ELSNET Workshop "International Roadmap for Language Resources"*.

VCAA. 2006. *English Language: Victorian Certicate of Education Study Design*. Victorian Curriculum and Assessment Authority. http://www.vcaa.vic.edu.au/vce/studies/englishlanguage/englangindex.htm%l.

Appendix: Selected Topics from ACM CS Body of Knowledge Related to Computational Linguistics

DS. Discrete Structures
- DS1. Functions, relations and sets
- DS2. Basic logic
- DS5. Graphs and trees
- DS6. Discrete probability

PF. Programming Fundamentals
- PF1. Fundamental programming constructs
- PF2. Algorithms and problem solving
- PF3. Fundamental data structures
- PF4. Recursion

AL. Algorithms and Complexity
- AL1. Basic algorithmic analysis
- AL2. Algorithmic strategies

IS. Intelligent Systems
- IS1. Fundamental issues in intelligent systems
- IS2. Search and constraint satisfaction
- IS3. Knowledge representation and reasoning
- IS7. (Natural language processing)

IM. Information Management
- IM1. Information models and systems
- IM3. Data modeling

SP. Social and Professional Issues
- SP4. Professional and ethical responsibilities
- SP5. Risks and liabilities of computer-based systems

SE. Software Engineering
- SE1. Software design
- SE2. Using application programming interfaces
- SE9. Component-based computing