Introducing Computational Concepts in a Linguistics Olympiad

Patrick Littell
Department of Linguistics
University of British Columbia
Vancouver, V6T1Z4, Canada
littell@interchange.ubc.ca

Lori Levin
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
lsl@cs.cmu.edu

Jason Eisner
Computer Science Department
Johns Hopkins University
Baltimore, MD 21218, USA
jason@cs.jhu.edu

Dragomir R. Radev
Department of EECS
School of Information
and Department of Linguistics
University of Michigan
radev@umich.edu

Abstract

Linguistics olympiads, now offered in more than 20 countries, provide secondary-school students a compelling introduction to an unfamiliar field. The North American Computational Linguistics Olympiad (NACLO) includes computational puzzles in addition to purely linguistic ones. This paper explores the computational subject matter we want to convey via NACLO, as well as some of the challenges that arise when adapting problems in computational linguistics to an audience that may have no background in computer science, linguistics, or advanced mathematics. We present a small library of reusable design patterns that have proven useful when composing puzzles appropriate for secondary-school students.

1 What is a Linguistics Olympiad?

A linguistics olympiad (LO) (Payne and Derzhanski, 2010) is a puzzle contest for secondary-school students in which contestants compete to solve self-contained linguistics problem sets. LOs have their origin in the Moscow Traditional Olympiad in Linguistics, established in 1965, and have since spread around the world; an international contest `http://www.ioling.org` has been held yearly since 2003.

In an LO, every problem set is self-contained, so no prior experience in linguistics is necessary to compete. In fact, LO contests are fun and rewarding for exactly this reason: by the end of the contest, contestants are managing to read hieroglyphics, conjugate verbs in Swahili, and capable of other amazing feats. Furthermore, they have accomplished this solely through their own analytical abilities and linguistic intuition.

Based on our experience going into high schools and presenting our material, this "linguistic" way of thinking about languages almost always comes as a novel surprise to students. They largely think about languages as collections of known facts that you learn in classes and from books, not something that you can dive into and figure out for yourself. This is a hands-on antidote to the common public misconception that linguists are fundamentally polyglots, rather than language scientists, and students come out of the experience having realized that linguistics is a very different field (and hopefully a more compelling one) than they had assumed it to be.

2 Computational Linguistics at the LO

Our goal, since starting the North American Computational Linguistics Olympiad (NACLO) in 2007 (Radev et al., 2008), has been to explore how this LO experience can be used to introduce students to computational linguistics. Topics in computational linguistics have been featured before in LOs, occasionally in the Moscow LO and with some regularity in the Bulgarian LO.

Our deliberations began with some troubling statistics regarding enrollments in computer science programs (Zweben, 2013). Between 2003 and 2007 enrollments in computer science dropped dramatically. This was attributed in part to the dip in the IT sector, but it also stemmed in part from a perception problem in which teenagers view CS careers as mundane and boring: "I don't want to be Dilbert,[1] sitting in a cubicle programming payroll software my whole life." This is an unrealistically narrow perception of the kinds of problems computer scientists tackle, and NACLO began in part as a way to publicize to teenagers that many interesting problems can be approached using computational methods.

Although enrollments are not yet back to the 2003 levels, there has been a sharp increase since 2007 (Zweben, 2013). The resurgence can be attributed in part to the strength of the IT sector, but also to the realization that computer science is relevant to almost every area of science and technology (Thibodeau, 2013). NACLO aims to be part of this trend by showing students that computer science is used in studying fascinating problems related to human language.

Even "traditional" LO puzzles are inherently computational in that they require pattern recognition, abstraction, generalization, and establishing and pruning a solution space. However, we also want to teach computational linguistics more explicitly. NACLO puzzles have featured a wide variety of topics in computational linguistics and computer science; they may focus on the application itself, or on concepts, tools, and algorithms that underlie the applications. Broadly, computational LO topics fall into three types, summarized below.

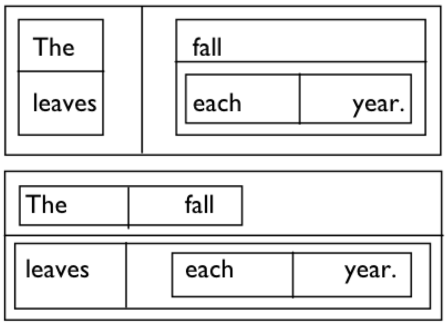
2.1 Technological applications

NACLO has included puzzles on technologies that most people are familiar with, including spell checking, information retrieval, machine translation, document summarization, and dialogue systems. In a typical applications puzzle, the contestants would discover how the application works, how it handles difficult cases, or what its limitations are. In "Summer Eyes" (Radev and Hesterberg, 2009), the contestant discovers the features that are used for selecting sentences in a summarization program, including the position of a sentence in the article, the number of words the sentence shares with the title, etc. In "Springing Up Baby" (Srivastava and Bender, 2008) and "Running on MT" (Somers, 2011), contestants explore word sense disambiguation in the context of machine translation, while "Thorny Stems" (Breck, 2008) and "A fox among the h" (Littell, 2012b) introduce stemming.

[1]An engineer in the eponymous American comic strip, Dilbert has a famously dysfunctional workplace and unrewarding job.

2.2 Formal grammars and algorithms

Some puzzles introduce the formal tools of computational linguistics and linguistic concepts that are important in computational linguistics, often in a whimsical way. For example, "Sk8 Parsr" (Littell, 2009) introduces shift-reduce parsing by means of a hypothetical skateboarding video game. "Aw-TOM-uh-tuh" (Littell, 2008) introduces finite-state machines in the form of a board game used to determine which strings form legal words in the Rotokas language. "Orwell-speak" (Eisner, 2009) asks students to modify a simple context-free grammar, and then to discover that a 4-gram model cannot model this language without precision or recall errors. "Twodee" (Eisner, 2012) invents a two-dimensional writing system, shown below, as a vehicle for helping students discover parsing ambiguity—and production ambiguity—without the full formal apparatus of grammars, nonterminals, or tree notation.



"The Little Engine That Could…Read" (Littell and Pustejovsky, 2012) explores quantifier monotonicity, while "Grice's Grifter Gadgets" (Boyd-Graber, 2013) covers Grice's maxims as part of the specification of a computerized game assistant.

2.3 Computational concepts

NACLO puzzles have also introduced computational concepts that go beyond computational linguistics. "Texting, Texting, One Two Three" (Littell, 2010b) and "The Heads and Tails of Huffman" (DeNero, 2013) introduce data compression. "One, Two, Tree" (Smith et al., 2012) introduces the Catalan numbers and other recurrences via binary bracketing of ambiguous compound nouns. "Nok-nok" (Fink, 2009) introduces Levenshtein distance by describing a hypothetical typing tutor for very bad spellers.

3 The Challenge of Writing Computational Problems

To achieve our goals, it becomes necessary to write computational linguistics puzzles in such a way that they are self-contained, requiring no prior experience in linguistics, computer science, or advanced math. This has proven very difficult, but not impossible, and in the past seven years we have managed to learn a lot about how to (and how not to) write them.

Perhaps the hardest part of writing any LO puzzle is that authors have to remove themselves from their knowledge and experience: to forget technical definitions of "phrase" or "noun" or "string" or "function," and to forget the facts and insights and history that formed our modern understanding of these. This is doubly hard when it comes to puzzles involving computational methods. The ability to write an algorithm that a computer could actually interpret is a specialized skill that we learned through education, and it is very, very hard to back up and imagine what it would be like to not be able to think like this. (It is almost like trying to remember what it was like to not be able to read—not simply not knowing a particular alphabet or language, but not even understanding how reading would work.)

Here is an illustration of an interesting but nonetheless inappropriate LO puzzle:

> *Here are fourteen English compound words:*
>
> | | |
> |---|---|
> | birdhouse | housework |
> | blackbird | tablespoon |
> | blackboard | teacup |
> | boardroom | teaspoon |
> | boathouse | workhouse |
> | cupboard | workroom |
> | houseboat | worktable |
>
> *Even if you didn't know any English, you could probably determine by looking at this list which English words were used to make up the compounds: "black", "bird", "board", etc...*
>
> *How would you do this if you were a computer?*

This task, although potentially appropriate for a programming competition, is inappropriate for an LO: the intended task requires some prior knowledge about what computers can and cannot do. Note that nowhere in the puzzle itself are the properties of this imaginary computer specified. It is assumed that the solver knows roughly the state of modern computing machinery and what kinds of instructions it can execute.

Imagine for a moment what a right answer to this puzzle would look like, and then picture what a wrong answer might look like. Your right answer was probably an algorithm that could run on an abstract computer with capabilities very much like real computers. The wrong answer probably made incorrect assumptions about what sorts of operations computers are capable of, or treated enormously complex operations as if they were primitive.[2]

The problem with the above puzzle is that it is very open-ended, and in the absence of a large body of shared knowledge between the author and the solver, the solver cannot know what it is the author wants or when they have solved it to the author's satisfaction.

In order to avoid this, it is best to set up the puzzle so that the "search space" for possible answers is relatively constrained, and the "win" conditions are clear. Ideally, if a contestant has solved a puzzle, they should know they have solved it, and thus be able to move on confidently to the next puzzle.[3] In this respect, LO puzzles are no different from professionally designed crossword puzzles or clever problems from the Mathematics Olympiads, or engineering problems given at the Computational Thinking Olympiad or in online games such as Kongregate. This feeling of accomplishment is key to the kind of rewarding learning experience that have made LOs so successful.

4 Design Patterns for CL Puzzles

Over the years, we have found several reliable strategies for turning CL ideas into solvable, rewarding puzzles.

[2] Keep in mind that today's contestants were born in the late 1990s. They are unlikely to even remember a world without ubiquitous Internet and powerful natural language search. Their conception of "what computers basically do" is not necessarily going to be the same as those of us who encountered computers when they were still recognizable as a kind of sophisticated calculator.

[3] This is not to say, however, that only those who solve a puzzle in its entirety should feel accomplished or rewarded. The best puzzles often contain layers of mysteries: it may be that only a few will solve every mystery in the puzzle, but most contestants come away with the satisfaction of having discovered something.

Not every computational puzzle makes use of these—some are entirely unique—but many do. In addition, these strategies are not mutually exclusive; many computational puzzles utilize several of these at once. For example, a "Broken Machine" puzzle may then present the solver with a "Troublemaker" task, or an "Assembly Required" machine may, upon assembly, turn out to be a "Broken" one.

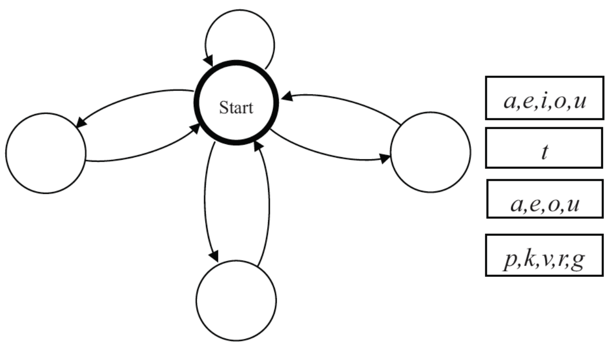
4.1 Assembly Required

The solver is presented with a task to complete, and also a partially specified algorithm for doing so. The partial specification illustrates the desired formal notation and the model of computation. But it may be missing elements, or the ordering or relationship between the elements is unclear, or some other aspect of the system remains unfinished. The solver is asked to complete the system so that it performs the appropriate task or produces the appropriate outputs.

For example, NACLO 2008 included a puzzle on stemming, "Thorny Stems" (Breck, 2008), in which contestants help develop an algorithm to isolate the stems of various words. In this puzzle, the contestant is not required to invent an algorithm *ex nihilo*; this would merely have rewarded those who already understand algorithms, not introduce algorithmic thinking to neophytes. Instead, the overall structure of the intended algorithm (an ordered sequence of if-thens) is made explicit, and the contestant's task is to fill in the details:

> Rule 1: If a word ends in _____, then replace _____ with _____ to form the stem.
>
> Rule 2: If a word ends in _____, then replace _____ with _____ to form the stem.

In another puzzle from the same contest, "AwTOM-uh-tuh" (Littell, 2008), the solver must complete an unfinished finite-state automaton so that it performs a language recognition task. The solver is given a brief introduction to FSAs and a simple sample FSA, and then given an incomplete FSA whose labels lack edges. The contestant's task is to place the labels on the correct edges so that the FSA accepts certain inputs and rejects others.



Other examples of the "Assembly Required" pattern can be found in the puzzles "Sk8 Parsr" (Littell, 2009), "The Heads and Tails of Huffman" (DeNero, 2013), and "BrokEnglish!" (Littell, 2011).

4.2 Black Box

The solver is presented with the inputs to a system and the outputs, and must work out how the system generated the outputs. Unlike in the "Assembly Required" pattern, little or no information about the algorithm is provided to the solver; the solver's fundamental task is to characterize this unknown algorithm as thoroughly as possible.

For example, NACLO 2010 featured a puzzle on Huffman text compression, "Texting, Texting, One Two Three" (Littell, 2010b), in which an unspecified algorithm converts strings of letters to strings of numbers:

> Testing testing = 33222143224142341-1222143224142341331
>
> Does anyone copy = 33233322143131-42343324221124232342343331

Working out the basic number-letter correspondences is relatively straightforward, but the real puzzle is working out the rationale behind these correspondences. Some of the answers require letters (like "r" and "x") that do not occur anywhere in the data, but can be deduced once the system as a whole is fully understood.

NACLO 2009 featured a puzzle on Levenshtein distance, "Nok-nok!" (Fink, 2009), that also used this pattern. In it, a spell-checker is rating how well (or poorly) a user has spelled a word.

User Input	Correct word	Output
owll	owl	“almost right”
ples	please	“quite close”
reqird	required	“quite close”
plez	please	“a bit confusing”
mispeln	misspelling	“very confusing”

The solver’s task is to work out the algorithm sufficiently to predict how the system would respond to novel inputs.

Other examples of the “Black Box” pattern can be found in “The Deschamps Codice” (Piperski, 2012) and “The Little Engine that Could. . . Read” (Littell and Pustejovsky, 2012).

Depending on the intended algorithm, the “Black Box” pattern may or may not be appropriate. This pattern works best when the nature of the transformation between input and output is relatively straightforward and the purpose of the transformation is relatively clear. In the Huffman coding puzzle, for example, the nature of the transformation is entirely obvious (replace letters with number sequences) and thus the solution space of the puzzle is relatively constrained (figure out which letters correspond to which number sequences and then try to figure out why). In the spell-checking puzzle, the purpose of the transformation is easily understood, giving the solver a head start on figuring out which features of the input the algorithm might be considering.

When the nature of the transformation is less obvious—for example, the generation of numbers of unclear significance, rating some unknown aspect of a text passage—Black Box is not as appropriate as the other patterns. The potential problem is that not only must the solver come up with an algorithm on their own, they must come up with the same algorithm the author did. Given a complicated algorithm, even small implementation details may lead to very different outputs, so a solver can even have found a basically correct solution but nevertheless not managed to produce the intended outputs.

In such cases, the “Assembly Required” or “Broken Machine” patterns are potentially more appropriate.

4.3 Broken Machine

The solver is presented with a system that purports to perform a particular task, but actually fails on particular inputs. The solver is tasked with figuring out what went wrong and, potentially, fixing the system so that it works. In some cases, the system simply has an error in it; in others, the system is correct but cannot handle certain difficult cases.

NACLO has featured a wide variety of broken machines, often with humorous outputs. “Help my Camera!” (Bender, 2009) features a dialogue system that could not correctly resolve pronoun references:

> Human: “There’s this restaurant on Bancroft that’s supposed to be really good that I heard about from my mother. Can you help me find it?”
> Computer: “Where did you last see your mother?”

“BrokEnglish!” (Littell, 2011) features a runaway script that replaced certain ISO 639-1 codes with language names:

> Hebrewy, ChamorRomanianrICHebrewcHebrewnlandic! whEnglish you get a FrEnglishcHebrewe momEnglisht, cHebrewck out thICHebrewcHebrewnlandic niCHebrewcHebrewn little pRomaniangram i wRomaniante.

Solvers are then tasked with determining why this script produced such a bizarre output, and additionally tasked with determining in what order the replacements had to have occurred in order to get this exact output.

“Orwellspeak” (Eisner, 2009) involves a context-free grammar that produces sentences that were grammatically correct but counter to the ideals of a fictional totalitarian Party. The solver must rewrite the grammar so that only “correct” thoughts can be uttered. In the second part of the puzzle, the solver must show that Markov models would be *inherently* broken.

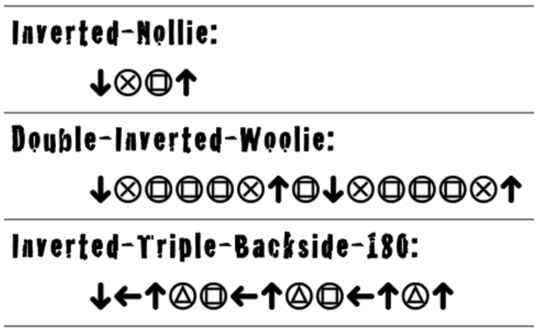
Other examples of “Broken Machines” are “The Lost Tram” (Iomdin, 2007), “Sk8 Parsr” (Littell, 2009), “A fox among the h” (Littell, 2012b), “The Little Engine that Could. . . Read” (Littell and Pustejovsky, 2012), and “Grice’s Grifter Gadgets” (Boyd-Graber, 2013).

4.4 Troublemaker

The solver is presented with a system and some sample inputs and outputs, and must discover an input that causes the system to fail, or produce outputs that are strange, suboptimal, or have some unusual property.

Few puzzles make use of only the "Troublemaker" pattern. Many are basically "Assembly Required" or "Broken Machine" puzzles that use a "Troublemaker" task to get the contestant thinking about the ways in which the system is limited or imperfect. They are also often creative—the contestant usually invents their own inputs—and thus can serve as a refreshing change of pace.[4]

NACLO 2009 featured a "Broken Machine" puzzle about shift-reduce parsing ("Sk8 Parsr") (Littell, 2009), couched in terms of a fictional skateboarding videogame. The solver is given an algorithm by which button presses are transformed into skateboard trick "combos" like those shown below, but many well-formed "combos" cannot correctly be parsed due to a shift-reduce conflict.



The solver is given an example of one such class of inputs, and then asked to discover other classes of inputs that likewise fail.

"Troublemaker" puzzles are not always couched in terms of bugs. "This problem is pretty // easy" (Radev, 2007a) asked contestants to construct eye-catching garden path sentences. In the Huffman text compression puzzle detailed above ("Texting, Texting, One Two Three") (Littell, 2010b), a "Troublemaker" task is introduced to get contestants thinking about the limits of compression. Although the compression algorithm was not "broken" in any way, any compression algorithm will "fail" on some possible input and return an output longer than the input, and the contestant was tasked to discover such an input.

"Troublemaker" tasks can also be found in "Grammar Rules" (Schalley and Littell, 2013) and "Yesbot" (Mitkov and Littell, 2013).

[4]If the "Troublemaker" task asks for an input with a particular formal property (i.e., a sentence generated or not generated from a particular grammar), automated grading scripts can determine the correctness of the answer without human intervention. This means that contestants can get a chance to enter "creative" answers even in large contests (like the NACLO Open Round) that utilize automatic grading.

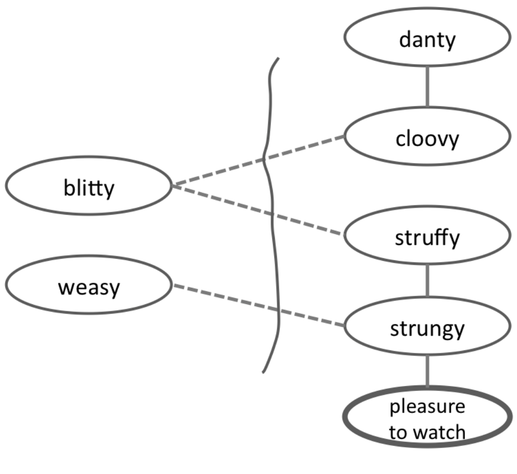
4.5 Jabberwock

Not all puzzle types revolve around abstract machines. Another recurring puzzle type, the "Jabberwock", involves asking the solver to puzzle out the syntactic or semantic properties of unknown words. Often these words are nonsense words, but this puzzle type can also work on natural language data. To perform this task, solvers often have to use the same methods that a computer would.

"We are all molistic in a way" (Radev, 2007b) asks solvers to infer the polarity of various nonsense adjectives based on a series of sentences. The list below includes a subset of the examples used in the real puzzle in 2007.

> The teacher is danty and cloovy.
> Mary is blitty but cloovy.
> Strungy and struffy, Diane was a pleasure to watch.
> Even though weasy, John is strungy.
> Carla is blitty but struffy.

The solver must work out from sentences such as these whether words like "danty" and "weasy" have positive or negative associations. In doing so, the student has essentially constructed and solved a semi-supervised learning problem.



In "Gelda's House of Gelbelgarg" (Littell, 2010a), solvers are presented with a page of fabricated restaurant reviews for an entirely fictional cuisine:

> "A hidden gem in Lower Uptown! Get the färsel-försel with gorse-weebel and you'll have a happy stomach for a week. And top it off with a flebba of sweet-bolger while you're at it!"

"I found the food confusing and disorienting. Where is this from? I randomly ordered the färsel-försel and had to send them back!"

Using various grammatical cues (article and pronoun choice, "less" vs. "fewer", etc.), solvers have to sort the items into things most likely to be discrete, countable objects, things most likely to be liquids or masses, and things most likely to be containers or measures.

This type of puzzle often violates the common LO restriction on using nonsense words and made-up languages, but it is not always possible to base this sort of puzzle on a completely unfamiliar language. Many "Jabberwock" puzzles involve inferring syntactic or semantic information about unknown words in an otherwise known language. The two puzzles above therefore require contestants to consult their own intuitions about English. These puzzles would have been entirely different (and prohibitively difficult) if the language had been completely unfamiliar.

Other Jabberwock puzzles include "Tiger Tale" (Radev, 2011) and "Cat and Mouse Story" (Littell, 2012a). "Tiger Tale" highlights some realistic sources of knowledge for machine translation, such as cognates and cross-language syntactic similarities.

4.6 Combinatorial Problems

Some puzzles effectively force the solver to design *and* run an algorithm, to get an answer that would be too difficult to compute by brute force. Such puzzles involve computational thinking. But since the solver only has to give the *output* of the algorithm, there is no need to agree on a type of computing device or a notation for writing algorithms down.

Such puzzles include combinatorial tasks that involve the counting, maximization, or existence of linguistic objects. They require mathematical and algorithmic skills (just as in math or programming competitions), and demonstrate how these skills apply to linguistics or NLP.

Portions of "One, Two, Tree" (Smith et al., 2012) and "Twodee" (Eisner, 2012) require solvers to count all ways to parse a sentence, or to count all sentences of a certain type. Because the counts are large, the solver must find the pattern, which involves writing down a closed-form formula such as 2^n or a more complex dynamic programming recurrence.

5 Conclusions

Researchers and teachers from the ACL community are invited to contact the NACLO organizing committee at `naclo14org@umich.edu`[5] with their ideas for new puzzles or new types of puzzles. All of the past puzzles and solutions can be browsed at `http://www.naclo.cs.cmu.edu/practice.html`. In general, puzzles in Round 1 each year should be easier and automatically gradable. Puzzles in Round 2 permit more involved questions and answers; this is a smaller contest in which the top Round 1 scorers (usually, the top 10 percent) can qualify for the International Linguistic Olympiad.

Thus far, NACLO's computational puzzles have reached at least 6,000 students at more than 150 testing sites (100+ high schools and 50+ university sites; the latter are open to students from all local high schools) in the U.S. and Canada, as well as at least 10,000 students in the three other English-language countries that share LO puzzles with NACLO.

We observe that most computational puzzles do not need obscure languages, staying on the contestant's home turf of English and technology. This does *not* mean, however, that the computational puzzles are purely formal and lack linguistic content. Some of them in fact probe subtle facts about English (the introspective method in linguistics), and some of them cover areas of linguistics that are underserved by traditional LO puzzles. Traditional LO puzzles instead ask the student to sort out vocabulary and basic morphophonological or orthographic patterns in a mystery language (the fieldwork method in linguistics). Students who enjoy "top-down" thinking or who are deeply interested in "how to do things with words" may prefer the former kind of puzzle.

Competitions are popular in many North American high schools, perhaps in part as a way to impress college admissions officers. We have exploited this to give students a taste of our interdisciplinary field before they choose a college major. Some students may be specifically attracted to NACLO by the word "computational" or the word "linguistics," or may be intrigued by their juxtaposition. Many NACLO participants reveal that

[5]Or `nacloXXorg@umich.edu`, where XX is the last two digits of the calendar year of the upcoming February.

they had started to study linguistics on their own before encountering NACLO, and have welcomed NACLO as an outlet for their enthusiasm and a place where they can interact with other students who have the same interests.

NACLO's past puzzles remain freely available on the web for anyone who is interested. Two volumes of NACLO-style puzzles (most of them from real competitions), edited by program chair Dragomir Radev, have recently been published by Springer (Radev, 2013a; Radev, 2013b). Adult hobbyists and home-schooled students may discover computational linguistics through encountering these puzzles. Avid LO contestants use them to prepare for upcoming contests. Finally, high school and college teachers can use them as the basis of whole-class or small-group classroom activities that expose students to computational thinking.

Acknowledgments

We would like to thank the National Science Foundation for supporting NACLO through the following grants: IIS0633871, BCS1137828, and IIS0838848. We also express our gratitude to NSF program managers Tatiana Korelsky, Terry Langendoen, and Joan Maling for their effort in initiating and maintaining NACLO. The Linguistic Society of America and the North American Chapter of the Association for Computational Linguistics provide ongoing support. Other sponsors, volunteers, and problem writers are too numerous to name. They are listed on the contest booklets each year, which can be found on the NACLO web site: `http://www.naclo.cs.cmu.edu`.

References

Emily Bender. 2009. Help my camera! In *North American Computational Linguistics Olympiad 2009*. http://www.naclo.cs.cmu.edu/assets/problems/naclo09F.pdf.

Jordan Boyd-Graber. 2013. Grice's grifter gadgets. In *North American Computational Linguistics Olympiad 2013*. http://www.naclo.cs.cmu.edu/2013/NACLO2013ROUND2.pdf.

Eric Breck. 2008. Thorny stems. In *North American Computational Linguistics Olympiad 2008*. http://www.naclo.cs.cmu.edu/assets/problems/NACLO08h.pdf.

John DeNero. 2013. The heads and tails of Huffman. In *North American Computational Linguistics Olympiad 2013*. http://www.naclo.cs.cmu.edu/2013/NACLO2013ROUND1.pdf.

Jason Eisner. 2009. Orwellspeak. In *North American Computational Linguistics Olympiad 2009*. http://www.naclo.cs.cmu.edu/assets/problems/naclo09M.pdf.

Jason Eisner. 2012. Twodee. In *North American Computational Linguistics Olympiad 2013*. http://www.naclo.cs.cmu.edu/problems2012/NACLO2012ROUND2.pdf.

Eugene Fink. 2009. Nok-nok! In *North American Computational Linguistics Olympiad 2009*. http://www.naclo.cs.cmu.edu/assets/problems/naclo09B.pdf.

Boris Iomdin. 2007. The lost tram. In *North American Computational Linguistics Olympiad 2007*. http://www.naclo.cs.cmu.edu/assets/problems/naclo07_f.pdf.

Patrick Littell and James Pustejovsky. 2012. The little engine that could...read. In *North American Computational Linguistics Olympiad 2012*. http://www.naclo.cs.cmu.edu/problems2012/NACLO2012ROUND2.pdf.

Patrick Littell. 2008. Aw-TOM-uh-tuh. In *North American Computational Linguistics Olympiad 2008*. http://www.naclo.cs.cmu.edu/assets/problems/NACLO08i.pdf.

Patrick Littell. 2009. Sk8 parsr. In *North American Computational Linguistics Olympiad 2009*. http://www.naclo.cs.cmu.edu/assets/problems/naclo09G.pdf.

Patrick Littell. 2010a. Gelda's house of gelbelgarg. In *North American Computational Linguistics Olympiad 2010*. http://www.naclo.cs.cmu.edu/problems2010/A.pdf.

Patrick Littell. 2010b. Texting, texting, one two three. In *North American Computational Linguistics Olympiad 2010*. http://www.naclo.cs.cmu.edu/problems2010/E.pdf.

Patrick Littell. 2011. BrokEnglish! In *North American Computational Linguistics Olympiad 2011*. http://www.naclo.cs.cmu.edu/problems2011/E.pdf.

Patrick Littell. 2012a. Cat and mouse story. In *North American Computational Linguistics Olympiad 2012*. http://www.naclo.cs.cmu.edu/problems2012/NACLO2012ROUND1.pdf.

Patrick Littell. 2012b. A fox among the h. In *North American Computational Linguistics Olympiad 2012*. http://www.naclo.cs.cmu.edu/problems2012/NACLO2012ROUND2.pdf.

Ruslan Mitkov and Patrick Littell. 2013. Grammar rules. In *North American Computational Linguistics Olympiad 2013*. http://www.naclo.cs.cmu.edu/2013/NACLO2013ROUND2.pdf.

Thomas E. Payne and Ivan Derzhanski. 2010. The linguistics olympiads: Academic competitions in linguistics for secondary school students. In Kristin Denham and Anne Lobeck, editors, *Linguistics at school*. Cambridge University Press.

Alexander Piperski. 2012. The Deschamps codice. In *North American Computational Linguistics Olympiad 2012*. http://www.naclo.cs.cmu.edu/problems2012/NACLO2012ROUND2.pdf.

Dragomir Radev and Adam Hesterberg. 2009. Summer eyes. In *North American Computational Linguistics Olympiad 2009*. http://www.naclo.cs.cmu.edu/assets/problems/naclo09E.pdf.

Dragomir R. Radev, Lori Levin, and Thomas E. Payne. 2008. The North American Computational Linguistics Olympiad (NACLO). In *Proceedings of the Third Workshop on Issues in Teaching Computational Linguistics*, pages 87–96, Columbus, Ohio, June. Association for Computational Linguistics. http://www.aclweb.org/anthology/W/W08/W08-0211.

Dragomir Radev. 2007a. This problem is pretty // easy. In *North American Computational Linguistics Olympiad 2007*. http://www.naclo.cs.cmu.edu/assets/problems/naclo07_h.pdf.

Dragomir Radev. 2007b. We are all molistic in a way. In *North American Computational Linguistics Olympiad 2007*. http://www.naclo.cs.cmu.edu/assets/problems/naclo07_a.pdf.

Dragomir Radev. 2011. Tiger tale. In *North American Computational Linguistics Olympiad 2011*. http://www.naclo.cs.cmu.edu/problems2011/F.pdf.

Dragomir Radev, editor. 2013a. *Puzzles in Logic, Languages, and Computation: The Green Book*. Springer: Berlin.

Dragomir Radev, editor. 2013b. *Puzzles in Logic, Languages, and Computation: The Red Book*. Springer: Berlin.

Andrea Schalley and Patrick Littell. 2013. Grammar rules! In *North American Computational Linguistics Olympiad 2013*. http://www.naclo.cs.cmu.edu/2013/NACLO2013ROUND1.pdf.

Noah Smith, Kevin Gimpel, and Jason Eisner. 2012. One, two, tree. In *North American Computational Linguistics Olympiad 2012*. http://www.naclo.cs.cmu.edu/problems2012/NACLO2012ROUND2.pdf.

Harold Somers. 2011. Running on MT. In *North American Computational Linguistics Olympiad 2011*. http://www.naclo.cs.cmu.edu/problems2011/A.pdf.

Ankit Srivastava and Emily Bender. 2008. Springing up baby. In *North American Computational Linguistics Olympiad 2008*. http://www.naclo.cs.cmu.edu/assets/problems/prob08b.pdf.

Patrick Thibodeau. 2013. Computer science enrollments soared last year, rising 30%, March. http://www.computerworld.com/s/article/9237459/Computer_science_enrollments_soared_last_year_rising_30_.

Stuart Zweben. 2013. Computing degree and enrollment trends, March. http://cra.org/govaffairs/blog/wp-content/uploads/2013/03/CRA_Taulbee_CS_Degrees_and_Enrollment_2011-12.pdf.