Competitive Grammar Writing*

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

Noah A. Smith
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
nasmith@cs.cmu.edu

Abstract

Just as programming is the traditional introduction to computer science, writing grammars by hand is an excellent introduction to many topics in computational linguistics. We present and justify a well-tested introductory activity in which teams of mixed background compete to write probabilistic context-free grammars of English. The exercise brings together symbolic, probabilistic, algorithmic, and experimental issues in a way that is accessible to novices and enjoyable.

1 Introduction

We describe a hands-on group activity for novices that introduces several central topics in computational linguistics (CL). While the task is intellectually challenging, it requires no background other than linguistic intuitions, no programming,[1] and only a very basic understanding of probability.

The activity is especially appropriate for mixed groups of linguists, computer scientists, and others, letting them collaborate effectively on small teams and learn from one another. A friendly competition among the teams makes the activity intense and enjoyable and introduces quantitative evaluation.

1.1 Task Overview

Each 3-person team is asked to write a generative context-free grammar that generates as much of English as possible (over a small fixed vocabulary). Obviously, writing a *full* English grammar would take years even for experienced linguists. Thus each team will only manage to cover a few phenomena, and imperfectly.

To encourage precision but also recall and linguistic creativity, teams are rewarded for generating sentences that are (prescriptively) grammatical but are not anticipated by other teams' grammars. This somewhat resembles scoring in the Boggle word game, where players are rewarded for finding valid words in a grid that are not found by other players.

A final twist is that the exercise uses *probabilistic* context-free grammars (PCFGs); the actual scoring methods are based on sampling and cross-entropy. Each team must therefore decide how to allocate probability mass among sentences. To avoid assigning probability of zero when attempting to parse another team's sentences, a team is allowed to "back off" (when parsing) to a simpler probability model, such as a part-of-speech bigram model, also expressed as a PCFG.

1.2 Setting

We have run this activity for six consecutive years, as a laboratory exercise on the *very first* afternoon of an intensive 2-week summer school on various topics in human language technology.[2] We allot 3.5 hours in this setting, including about 15 minutes for setup, 30 minutes for instructions, 15 minutes for evaluation, and 30 minutes for final discussion.

The remaining 2 hours is barely enough time for team members to get acquainted, understand the requirements, plan a strategy, and make a small dent in

[*] This work was supported by NSF award 0121285, "ITR/IM+PE+SY: Summer Workshops on Human Language Technology: Integrating Research and Education," and by a Fannie and John Hertz Foundation Fellowship to the second author. We thank David A. Smith and Markus Dreyer for co-leading the lab in 2004–2007 and for implementing various improvements in 2004–2007 and for providing us with data from those years. The lab has benefited over the years from feedback from the participants, many of whom attended the JHU summer school thanks to the generous support of NAACL. We also thank the anonymous reviewers for helpful comments.

[1] In our setup, students do need the ability to invoke scripts and edit files in a shared directory, e.g., on a Unix system.

[2] This 2-week course is offered as a prelude to the Johns Hopkins University summer research workshops, sponsored by the National Science Foundation and the Department of Defense. In recent years the course has been co-sponsored by the North American ACL.

Proceedings of the Third Workshop on Issues in Teaching Computational Linguistics (TeachCL-08), pages 97–105,
Columbus, Ohio, USA, June 2008. ©2008 Association for Computational Linguistics

the problem. Nonetheless, participants consistently tell us that the exercise is enjoyable and pedagogically effective, almost always voting to stay an extra hour to make further progress.

Our 3-person teams have consisted of approximately one undergraduate, one junior graduate student, and one more senior graduate student. If possible, each team should include at least one member who has basic familiarity with some syntactic phenomena and phrasal categories. Teams that wholly lack this experience have been at a disadvantage in the time-limited setting.

1.3 Resources for Instructors

We will maintain teaching materials at `http://www.clsp.jhu.edu/grammar-writing`, for both the laboratory exercise version and for homework versions: scripts, data, instructions for participants, and tips for instructors. While our materials are designed for participants who are fluent in English, we would gladly host translations or adaptations into other languages, as well as other variants and similar assignments.

2 Why Grammar Writing?

A computer science curriculum traditionally starts with *programming*, because programming is accessible, hands-on, and necessary to motivate or understand most other topics in computer science. We believe that *grammar writing* should play the same role in computational linguistics—as it often did before the statistical revolution[3]—and for similar reasons.

Grammar writing remains central because many theoretical and applied CL topics center around grammar formalisms. Much of the field tries to design expressive formalisms (akin to programming languages); solve linguistic problems within them (akin to programming); enrich them with probabilities; process them with efficient algorithms; learn them from data; and connect them to other modules in the linguistic processing pipeline.

Of course, there are interesting grammar formalisms at all levels of language processing. One might ask why *syntax* is a good level at which to begin education in computational linguistics.

First, starting with syntax establishes at the start that there are formal and computational methods specific to natural language. Computational linguistics is not merely a set of applied tasks to be solved with methods already standardly taught in courses on machine learning, theory of computation,[4] or knowledge representation.

Second, we have found that syntax captures students' interest rapidly. They quickly appreciate the linguistic phenomena, see that they are non-trivial, and have little trouble with the CFG formalism.

Third, beginning specifically with PCFGs pays technical dividends in a CL course. Once one understands PCFG models, it is easy to understand the simpler finite-state models (including n-gram models, HMMs, etc.) and their associated algorithms, either by analogy or by explicit reduction to special cases of PCFGs. CFGs are also a good starting point for more complex syntactic formalisms (BNF, categorial grammars, TAG, LFG, HPSG, etc.) and for compositional semantics. Indeed, our exercise motivates these more complex formalisms by forcing students to work with the more impoverished PCFG formalism and experience its limitations.

3 Educational Goals of the Exercise

Our grammar-writing exercise is intended to serve as a touchstone for discussion of many subsequent topics in NLP and CL (which are *italicized* below). As an instructor, one can often refer back later to the exercise to remind students of their concrete experience with a given concept.

Generative probabilistic models. The first set of concepts concerns *language models*. These are easiest to understand as processes for generating text. Thus, we give our teams a script for *generating random sentences* from their grammar and their backoff

[3] The first author was specifically inspired by his experience writing a grammar in Bill Woods's NLP course at Harvard in 1987. An anonymous reviewer remarks that such assignments were common at the time. Our contributions are to introduce statistical and finite-state elements, to make the exercise into a game, and to provide reusable instructional materials.

[4] Courses on theory of computation do teach pushdown automata and CFGs, of course, but they rarely touch on parsing or probabilistic grammars, as this exercise does. Courses on compilers may cover parsing algorithms, but only for restricted grammar families such as unambiguous LR(1) grammars.

model—a helpful way to observe the generative capacity and qualitative behavior of any model.

Of course, in practice a generative grammar is most often run "backwards" to *parse an observed sentence* or score its *inside probability*, and we also give the teams a script to do that. Most teams do actually run these scripts repeatedly to test their grammars, since both scripts will be central in the evaluation (where sentences are randomly generated from one grammar and scored with another grammar).

It is common for instructors of NLP to show examples of randomly-generated text from an n-gram model (e.g., Jurafsky and Martin, 2000, pp. 202–203), yet this amusing demonstration may be misinterpreted as merely illustrating the inadequacy of n-gram models. Our use of a hand-crafted PCFG *combined with* a bigram-based (HMM) backoff grammar demonstrates that although the HMM is much worse at *generating* valid English sentences (precision), it is much better at robustly assigning nonzero probability when *analyzing* English sentences (recall).

Finally, generative models do more than assign probability. They often involve linguistically meaningful *latent variables*, which can be recovered given the observed data. Parsing with an appropriate PCFG thus yields a intuitive and useful analysis (a *syntactic parse tree*), although only for the sentences that the PCFG covers. Even parsing with the simple backoff grammar that we initially provide yields some coarser analysis, in the form of a part-of-speech tagging, since this backoff grammar is a right-branching PCFG that captures part-of-speech bigrams (for details see §1.1, §4.1, and Table 2). In fact, parsing with the backoff PCFG is isomorphic to Viterbi decoding in an *HMM part-of-speech tagger*, a topic that is commonly covered in NLP courses.

Modeling grammaticality. The next set of concepts concerns linguistic grammaticality. During the evaluation phase of our exercise (see below), students must make *grammaticality judgments* on other teams' randomly generated sentences—which are usually nonsensical, frequently hard for humans to parse, and sometimes ungrammatical. This concrete task usually prompts questions from students about how grammaticality ought to be defined, both for purposes of the task and in principle. It could also be used to discuss why some of the sentences are so hard for humans to understand (e.g., garden-path and frequency effects) and what *parsing strategies* humans or machines might use.

The exercise of modeling grammaticality with the CFG formalism, a formalism that appears elsewhere in the computer science curriculum, highlights some important differences between natural languages and formal languages. A natural language's true grammar is unknown (and may not even exist: perhaps the CFG formalism is inadequate). Rather, a grammar must be *induced or constructed* as an approximate model of corpus data and/or certain native-speaker intuitions. A natural language also differs from a programming language in including *ambiguous sentences*. Students observe that the parser *uses probabilities to resolve ambiguity*.

Linguistic analysis. Grammar writing is an excellent way to get students thinking about *linguistic phenomena* (e.g., adjuncts, embedded sentences, *wh*-questions, clefts, point absorption of punctuation marks). It also forces students to think about appropriate *linguistic formalisms*. Many phenomena are tedious to describe within CFGs (e.g., agreement, movement, subcategorization, selectional restrictions, morphological inflection, and phonologically-conditioned allomorphy such as a vs. an). They can be treated in CFGs only with a large number of repetitive rules. Students appreciate these problems by grappling with them, and become very receptive to designing expressive improvements such as feature structures and slashed categories.

Parameter tuning. Students observe the *effects of changing the rule probabilities* by running the scripts. For example, teams often find themselves generating unreasonably long (or even infinite) sentences, and must damp down the probabilities of their recursive rules. Adjusting the rule probabilities can also change the score and optimal tree that are returned by the parser, and can make a big difference in the final evaluation (see §5). This appreciation for the role of numerical parameters helps motivate future study of *machine learning* in NLP.

Quantitative evaluation. As an engineering pursuit, NLP research requires objective evaluation measures to know how well systems work.

Our first measure is the *precision* of each team's probabilistic grammar: how much of its probability mass is devoted to sentences that are truly grammatical? Estimating this requires human grammaticality judgments on a *random sample C* of sentences generated from all teams' grammars. These binary judgments are provided by the participants themselves, introducing the notion of linguistic *annotation* (albeit of a very simple kind). Details are in §4.3.3.

Our second measure is an upper-bound approximation to *cross-entropy* (or log-perplexity—in effect, the *recall* of a probability model): how well does each team's probabilistic model (this time including the backoff model of §1.1) anticipate unseen data that are truly grammatical? (Details in §4.3.3.)

Note that in contrast to parsing competitions, we do not evaluate the quality of the parse trees (e.g., PARSEVAL). Our cross-entropy measure evaluates only the grammars' ability to predict word strings (language modeling). That is because we impose no annotation standard for parse trees: each team is free to develop its own theory of syntax. Furthermore, many sentences will only be parsable by the backoff grammar (e.g., a bigram model), which is not expected to produce a full syntactic analysis.

The lesson about cross-entropy evaluation is slightly distorted by our peculiar choice of test data. In principle, the instructors might prepare a batch of grammatical sentences ahead of time and split them into a *test set* (used to evaluate cross-entropy at the end) and a *development set* (provided to the students at the start, so that they know which grammatical phenomena are important to handle). The activity could certainly be run in this way to demonstrate *proper experimental design* for evaluating a language model (discussed further in §5 and §6). We have opted for the more entertaining "Boggle-style" evaluation described in §1.1, where teams try to stump one another by generating difficult test data, using the fixed vocabulary. Thus, we evaluate each team's cross-entropy on all grammatical sentences in the collection C, which was generated *ex post facto* from all teams' grammars.

4 Important Details

4.1 Data

A few elements are provided to participants to get them started on their grammars.

Vocabulary. The terminal vocabulary Σ consists of words from early scenes of the film *Monty Python and the Holy Grail* along with some inflected forms and function words, for a total vocabulary of 220 words. For simplicity, only 3rd-person pronouns, nouns, and verbs are included. All words are case-sensitive for readability (as are the grammar nonterminals), but we do not require or expect sentence-initial capitalization.

All teams are restricted to this vocabulary, so that the sentences that they generate will not frustrate other teams' parsers with out-of-vocabulary words. However, they are free to use words in unexpected ways (e.g., using castle in its verbal sense from chess, or building up unusual constructions with the available function words).

Initial lexicon. The initial *lexical rules* take the form $T \rightarrow w$, where $w \in \Sigma^+$ and $T \in \mathcal{T}$, with $\mathcal{T}$ being a set of six coarse part-of-speech tags:

Noun: 21 singular nouns starting with consonants

Det: 9 singular determiners

Prep: 14 prepositions

Proper: 8 singular proper nouns denoting people (including multiwords such as Sir Lancelot)

VerbT: 6 3rd-person singular present transitive verbs

Misc: 183 other words, divided into several commented sections in the grammar file

Students are free to change this tagset. They are especially encouraged to refine the Misc tag, which includes 3rd-person plural nouns (including some proper nouns), 3rd-person pronouns (nominative, accusative, and genitive), additional 3rd-person verb forms (plural present, past, stem, and participles), verbs that cannot be used transitively, modals, adverbs, numbers, adjectives (including some comparative and superlative forms), punctuation, coordinating and subordinating conjunctions, *wh*-words, and a few miscellaneous function words (to, not, 's).

The initial lexicon is ambiguous: some words are associated with more than one tag. Each rule has weight 1, meaning that a tag T is equally likely to rewrite as any of its allowed nonterminals.

Initial grammar. We provide the "S1" rules in Table 1, so that students can try generating and parsing

1	S1	→	NP VP .
1	VP	→	VerbT NP
20	NP	→	Det Nbar
1	NP	→	Proper
20	Nbar	→	Noun
1	Nbar	→	Nbar PP
1	PP	→	Prep NP

Table 1: The S1 rules: a starting point for building an English grammar. The start symbol is S1. The weights in the first column will be normalized into generative probabilities; for example, the probability of expanding a given NP with NP → Det Nbar is actually $20/(20+1)$.

1	S2	→	
1	S2	→	_Noun
1	S2	→	_Misc
1	_Noun	→	Noun
1	_Noun	→	Noun _Noun
1	_Noun	→	Noun _Misc
1	_Misc	→	Misc
1	_Misc	→	Misc _Noun
1	_Misc	→	Misc _Misc

Table 2: The S2 rules (simplified here where $\mathcal{T} = \{\text{Noun}, \text{Misc}\}$): a starting point for a backoff grammar. The start symbol is S2. The _Noun nonterminal generates those phrases that start with Nouns. Its 3 rules mean that following a Noun, there is 1/3 probability each of stopping, continuing with another Noun (via _Noun), or continuing with a Misc word (via _Misc).

sentences right away. The S1 and lexical rules together implement a very small CFG. Note that no Misc words can yet be generated. Indeed, this initial grammar will only generate some simple grammatical SVO sentences in singular present tense, although they may be unboundedly long and ambiguous because of recursion through Nbar and PP.

Initial backoff grammar. The provided "S2" grammar is designed to assign positive probability to any string in Σ^* (see §1.1). At least initially, this PCFG generates only right-branching structures. Its nonterminals correspond to the states of a weighted finite-state machine, with start state S2 and one state per element of $\mathcal{T}$ (the coarse parts of speech listed above). Table 2 shows a simplified version.

From each state, transition into any state except the start state S2 is permitted, and so is stopping. These rules can be seen as specifying the *transitions* in a bigram hidden Markov model (HMM) on part-of-speech tags, whose *emissions* are specified by the lexical rules. Since each rule initially has weight 1, all part-of-speech sequences of a given length are equally likely, but these weights could be changed to arbitrary transition probabilities.

Arthur is the king .
Arthur rides the horse near the castle .
riding to Camelot is hard .
do coconuts speak ?
what does Arthur ride ?
who does Arthur suggest she carry ?
are they suggesting Arthur ride to Camelot ?
Guinevere might have known .
it is Sir Lancelot who knows Zoot !
neither Sir Lancelot nor Guinevere will speak of it .
the Holy Grail was covered by a yellow fruit .
do not speak !
Arthur will have been riding for eight nights .
Arthur , sixty inches , is a tiny king .
Arthur and Guinevere migrate frequently .
he knows what they are covering with that story .
the king drank to the castle that was his home .
when the king drinks , Patsy drinks .

Table 3: Example sentences. Only the first two can be parsed by the initial S1 and lexical rules.

Start rules. The initial grammar S1 and the initial backoff grammar S2 are tied together by a single symbol START, which has two production rules:

99	START	→	S1
1	START	→	S2

These two rules are obligatory, but their weights may be changed. The resulting model, rooted at START, is a *mixture* of the S1 and S2 grammars, where the weights of these two rules implement the mixture coefficients. This is a simple form of backoff smoothing by linear interpolation (Jelinek and Mercer, 1980). The teams are warned to pay special attention to these rules. If the weight of START → S1 is decreased relative to START → S2, then the model relies more heavily on the backoff model—perhaps a wise choice for keeping cross-entropy small, if the team has little faith in S1's ability to parse the forthcoming data.

Sample sentences. A set of 27 example sentences in Σ^+ (subset shown in Table 3) is provided for linguistic inspiration and as practice data on which to run the parser. Since only 2 of these sentences can be parsed with the initial S1 and lexical rules, there is plenty of room for improvement. A further development set is provided midway through the exercise (§4.3.2).

4.2 Computing Environment

We now describe how the above data are made available to students along with some software.

4.2.1 Scripts

We provide scripts that implement two important capabilities for PCFG development. Both scripts are invoked with a set of grammar files specified on the command line (typically all of them, "`*.gr`"). A PCFG is obtained by concatenating these files and stripping their comments, then normalizing their rule weights into probabilities (see Table 1), and finally checking that all terminal symbols of this PCFG are legal words of the vocabulary Σ.

The **random generation** script prints a sample of n sentences from this PCFG. The generator can optionally print trees or flat word sequences. A start symbol other than the default S1 may be specified (e.g., NP, S2, START, etc.), to allow participants to test subgrammars or the backoff grammar.[5]

The **parsing** script prints the most probable parse tree for each sentence read from a file (or from the standard input). A start symbol may again be specified; this time the default is START. The parser also prints each sentence's probability, total *number of parses*, and the fraction of the probability that goes to the best parse.

Tree outputs can be pretty-printed for readability.

4.2.2 Collaborative Setup

Teams of three or four sit at adjacent workstations with a shared filesystem. The scripts above are publicly installed; a handout gives brief usage instructions. The instructor and teaching assistant roam the room and offer assistance as needed.

Each team works in its own shared directory. The Emacs editor will warn users who are simultaneously editing the same file. Individual participants tend to work on different sub-grammar files; all of a team's files can be concatenated (as `*.gr`) when the scripts are run. (The directory initially includes separate files for the S1 rules, S2 rules, and lexical rules.) To avoid unexpected interactions among these grammar fragments, students are advised to divide work based on nonterminals; e.g., one member of a team may claim jurisdiction over all rules of the form $\mathsf{VP\_plural} \rightarrow \cdots$.

4.3 Activities

4.3.1 Introductory Lecture

Once students have formed themselves into teams and managed to log in at adjacent computers, we begin with an 30-minute introductory lecture. No background is assumed. We explain PCFGs simply by showing the S1 grammar and hand-simulating the action of the random sentence generator.

We explain the goal of extending the S1 grammar to cover more of English. We explain how each team's precision will be evaluated by human judgments on a sample, but point out that this measure gives no incentive to increase coverage (recall). This motivates the "Boggle" aspect of the game, where teams must also be able to parse one another's grammatical sentences, and indeed assign them as high a probability as possible. We demonstrate how the parser assigns a probability by running it on the sentence that we earlier generated by hand.[6]

We describe how the parser's probabilities are turned into a cross-entropy measure, and discuss strategy. Finally, we show that parsing a sentence that is not covered by the S1 grammar will lead to infinite cross-entropy, and we motivate the S2 backoff grammar as an escape hatch.

4.3.2 Midpoint: Development data

Once or more during the course of the exercise, we take a snapshot of all teams' S1 grammars and sample 50 sentences from each. The resulting collection of sentences, in random order, is made available to all teams as a kind of development data. While we do not filter for grammaticality as in the final evaluation, this gives all participants an idea of what they will be up against when it comes time

[5] For some PCFGs, the stochastic process implemented by the script has a nonzero probability of failing to terminate. This has not caused problems to date.

[6] The probability will be tiny, as a product of many rule probabilities. But it may be higher than expected, and students are challenged to guess why: there are additional parses beyond the one we hand-generated, and the parser sums over all of them.

to parse other teams' sentences. Teams are on their honor not to disguise the true state of their grammar at the time of the snapshot.

4.3.3 Evaluation procedure

Grammar development ends at an announced deadline. The grammars are now evaluated on the two measures discussed in §3. The instructors run a few scripts that handle most of this work.

First, we generate a collection C by sampling 20 sentences from each team's probabilistic grammar, using S1 as the start symbol. (Thus, the backoff S2 grammar is not used for generation.)

We now determine, for each team, what fraction of its 20-sentence sample was grammatical. The participants play the role of grammaticality judges. In our randomized double-blind procedure, each individual judge receives (in his or her team directory) a file of about 20 sentences from C, with instructions to delete the ungrammatical ones and save the file, implying coarse Boolean grammaticality judgments.[7] The files are constructed so that each sentence in C is judged by 3 different participants; a sentence is considered grammatical if ≥ 2 judges thinks that it is.

We define the test corpus $\hat{C}$ to consist of all sentences in C that were judged grammatical. Each team's full grammar (using START as the start symbol to allow backoff) is used to parse $\hat{C}$. This gives us the $\log_2$-probability of each sentence in $\hat{C}$; the cross-entropy score is the sum of these $\log_2$-probabilities divided by the length of $\hat{C}$.

4.3.4 Group discussion

While the teaching assistant is running the evaluation scripts and compiling the results, the instructor leads a general discussion. Many topics are possible, according to the interests of the instructor and participants. For example: What linguistic phenomena did the teams handle, and how? Was the CFG formalism adequately expressive? How well would it work for languages other than English?

What strategies did the teams adopt, based on the evaluation criteria? How were the weights chosen? How would you build a better backoff grammar?[8]

How would you organize a real long-term effort to build a full English grammar? What would such a grammar be good for? Would you use any additional tools, data, or evaluation criteria?

		cross-entropy	new rules	
team	precision	(bits/sent.)	lex.	other
A	0.30	**35.57**	202	111
B	0.00	54.01	304	80
C	**0.80**	38.48	179	48
D	0.25	49.37	254	186
E	0.55	39.59	198	114
F	0.00	39.56	193	37
G	0.65	40.97	71	15
H	0.30	36.53	176	9
I	0.70	36.17	181	54
J	0.00	∞	193	29

Table 4: Teams' evaluation scores in one year, and the number of new rules (not including weight changes) that they wrote. Only teams A and H modified the relative weights of the START rules (they used 80/20 and 75/25, respectively), giving them competitive perplexity scores. (Cross-entropy in this year was approximated by an upper bound that uses only the probability of each sentence's single best parse.)

5 Outcomes

Table 4 shows scores achieved in one year (2002).

A valuable lesson for the students was the importance of backoff. None but the first two of the example sentences (Table 3) are parseable with the small S1 grammar. Thus, the best way to reduce perplexity was to upweight the S2 grammar and perhaps spend a little time improving its rules or weights. Teams that spent all of their time on the S1 grammar may have learned a lot about linguistics, but tended to score poorly on perplexity.

Indeed, the winning team in a later year spent nearly all of their effort on the S2 grammar. They placed almost all their weight on the S2 grammar, whose rules they edited and whose parameters they estimated from the example sentences and development data. As for their S1 grammar, it generated only a small set of grammatical sentences with ob-

[7] Judges are on their honor to make fair judgments rather than attempt to judge other teams' sentences ungrammatical. Moreover, such an attempt might be self-defeating, as they might unknowingly be judging some of their own team's sentences ungrammatical.

[8] E.g., training the model weights, extending it to trigrams, or introducing syntax into the S2 model by allowing it to invoke nonterminals of the S1 grammar.

scure constructions that other teams were unlikely to model well in *their* S1 grammars. This gave them a 100% precision score on grammaticality while presenting a difficult parsing challenge to other teams. This team gamed our scoring system, exploiting the idiosyncrasy that S2 would be used to parse but not to generate. (See §3 for an alternative system.)

We conducted a *post hoc* qualitative survey of the grammars from teams in 2002. Teams were not asked to provide comments, and nonterminal naming conventions often tend to be inscrutable, but the intentions are mostly understandable. All 10 teams developed more fine-grained parts of speech, including coordinating conjunctions, modal verbs, number words, adverbs. 9 teams implemented singular and plural features on nouns and/or verbs, and 9 implemented the distinction between base, past, present, and gerund forms of verbs (or a subset of those). 7 teams brought in other features like comparative and superlative adjectives and personal vs. possessive pronouns. 4 teams modeled pronoun case. Team C created a "location" category.

7 teams explicitly tried to model questions, often including rules for *do*-support; 3 of those teams also modeled negation with *do*-insertion. 2 teams used gapped categories (team D used them extensively), and 7 teams used explicit $\bar{X}$ nonterminals, most commonly within noun phrases (following the initial grammar). Three teams used a rudimentary subcategorization frame model, distinguishing between sentence-taking, transitive, and intransitive verbs, with an exploded set of production rules as a result. Team D modeled appositives.

The amount of effort teams put into weights varied, as well. Team A used 11 distinct weight values from 1 to 80, giving 79 rules weights > 1 (next closest was team 10, with 7 weight values in $[1, 99]$ and only 43 up-weighted rules). Most teams set fewer than 25 rules' weights to something other than 1.

6 Use as a Homework Assignment

Two hours is not enough time to complete a *good* grammar. Our participants are ambitious but never come close to finishing what they undertake; Table 4 reflects incomplete work. Nonetheless, we believe that the experience still successfully fulfills many of the goals of §2–3 in a short time, and the participants enjoy the energy in a roomful of peers racing toward a deadline. The fact that the task is open-ended and clearly impossible keeps the competition friendly.

An alternative would be to allot 2 weeks or more as a homework assignment, allowing teams to go more deeply into linguistic issues and/or backoff modeling techniques. A team's grade could be linked to its performance. In this setting, we recommend limiting the team size to 1 or 2 people each, since larger teams may not be able to find time or facilities to work side-by-side for long.

This homework version of our exercise might helpfully be split into two assignments:

Part 1 (non-competitive, smaller vocabulary). "Extend the initial S1 grammar to cover a certain small set of linguistic phenomena, as illustrated by a development set [e.g., Table 3]. You will be evaluated on the cross-entropy of your grammar on a test set that closely resembles the development set [see §3], and perhaps also on the acceptability of sentences sampled from your grammar (as judged by you, your classmates, or the instructor). You will also receive qualitative feedback on how correctly and elegantly your grammar solves the linguistic problems posed by the development set."

Part 2 (competitive, full 220-word vocabulary). "Extend your S1 grammar from Part 1 to generate phenomena that stump other teams, and add an S2 grammar to avoid being stumped by them. You will be evaluated as follows . . . [see §4.3.3]."

We have already experimented with simpler non-competitive grammar-writing exercises (similar to Part 1) in our undergraduate NLP courses. Given two weeks, even without teammates, many students do a fine job of covering several non-trivial syntactic phenomena. These assignments are available for use by others (see §1.3). In some versions, students were asked to write their own random generator, judge their own sentences, explain how to evaluate perplexity, or guess why the S2 grammar was used.

7 Conclusion

We hope that other instructors can make use of these materials or ideas. Our competitive PCFG-writing game touches upon many core CL concepts, is challenging and enjoyable, allows collaboration, and is suitable for cross-disciplinary and intro courses.

References

F. Jelinek and R. L. Mercer. 1980. Interpolated estimation of Markov source parameters from sparse data. In *Proc. of Workshop on Pattern Recognition in Practice*.

D. Jurafsky and J.H. Martin. 2000. *Speech and Language Processing*. Prentice Hall.