A Baseline Method for Compiling Typed Unification Grammars into Context Free Language Models

Manny Rayner[1], *John Dowding*[2], *Beth Ann Hockey*[2]

[1] netdecisions, Wellington House, East Road, Cambridge CB1 1BH, UK
manny.rayner@netdecisions.co.uk

[2] RIACS, Mail Stop 19–39, NASA Ames Research Center, Moffett Field, CA 94035-1000
{jdowding, bahockey}@riacs.edu

Abstract

This paper presents a minimal enumerative approach to the problem of compiling typed unification grammars into CFG language models, a prototype implementation and results of experiments in which it was used to compile some non-trivial unification grammars. We argue that enumerative methods are considerably more useful than has been previously believed. Also, the simplicity of enumerative methods makes them a natural baseline against which to compare alternative approaches.

1. Introduction

Grammar based language models for constraining speech recognition are particularly attractive as an alternative to statistical models in domains that lack extensive speech corpora. For commercial dialogue systems, the case in which there is not enough speech data to train effective statistical models seems to be the norm. This lack of data also impacts research domains that are relatively novel, such as dialogue interfaces to robots. Given the difficulties involved in using statistical modeling with limited speech data, we think it is important to investigate ways in which grammar based models can be efficiently and effectively produced.

Context free grammars (CFGs) can be tedious to write and difficult to maintain, compared to grammars written in higher level formalisms such as unification based grammars. For each rule in the higher level grammar there are likely to be many rules, very similar to each other, in a comparable CFG. The higher-level formalism provides a more compact representation and expresses linguistic dependencies and relations more transparently and explicitly than a corresponding CFG.

Compiling the CFG language model from a grammar written in the higher level formalism is one way to produce a CFG language model effectively while taking advantage of the attractive properties of the higher-level formalism ([1]), and there have already been several pioneering attempts to write systems which can compile non-trivial unification grammars into useful CFG grammars [2],[1],[3]. Although interesting, these systems have some difficulties. Experience has in particular shown that their characteristics are too unpredictable; apparently small changes in the input unification grammars can have large effects in terms of increased compilation times. The compilation time with these algorithms is unpredictable because they search extremely large spaces using deductive methods that explore many paths simultaneously by performing reasoning directly on unification grammar representations. There is, however, a much simpler alternative, which we argue has not been adequately explored: to expand out the unification grammar into a CFG by non-deterministically instantiating all rules in all possible ways, and then proceed by manipulating the resulting CFG. We will refer to method of this kind as *enumerative compilation*. It is of course clear that completely naive implementations of enumerative compilation will not scale up to substantial grammars. With a few simple enhancements, however, we have discovered that these methods are surprisingly powerful.

We present initial results for REGULUS, a prototype implementation of an enumerative compilation method, which compiles typed unification grammars into CFG language models expressed in Nuance Grammar Specification Language (GSL) notation [4]. REGULUS divides the process of compiling a unification grammar into a CFG into three phases:

Pre-processing: Perform a suitable transformation of the initial unification grammar, in order to make it easier for the following stage to process.

Expansion: Expand out the transformed unification grammar into a CFG by non-deterministically instantiating each feature in each rule to all of its permitted values.

Filtering: Remove all CFG rules which are irrelevant, in the sense that they can deleted without changing the language generated by the grammar.

2. Representing the rule space

Our starting point is a typed unification-based feature grammar such that each feature f has a finite range of possible values $dom(f)$; our goal is to convert this into an equivalent CFG. A subset of these features will take values that are arbitrary boolean combinations of the atomic values for the feature. For example if the feature F has the possible values a, b and c, there are seven possible values: a, b, c, $a \vee b$, $a \vee c$, $b \vee c$ and $a \vee b \vee c$. These *boolean valued* features are represented using the method of Mellish[5], which provides a compact form in which to express the possible $2^n - 1$ values for the feature.

While it is conceivable to generate a CFG simply by instantiating each feature in each rule with every consistent value, the size of the resulting rule space may be prohibitively large. As pointed out in [2], the problem becomes particularly acute in rules with many daughters. If each daughter can be expanded in N ways, and there are M of them, then we have N^M ways to expand the body of the rule. [2] provides as an example a rule with 8 daughters, each of which has 60 possible instantiations, leading to 60^8 instantiated rules. At first sight, this implies that an enumerative strategy has no chance of succeeding on any substantial grammar.

For these reasons, [2] concluded that it was necessary to use a deductive approach that performs an analysis to determine which possible instantiations of rules actually arise. The advantage of the deductive approach is that it may not be necessary to search the whole space; deductive operations in effect consider many paths at once, leading to a real search space that is far smaller than the space of all possible rules. Unfortunately, it is in general impossible to estimate the size of this real search space for most interesting grammars. The upshot is that there is no easy way to know how long any given compilation will take, and practical experience with the deductive approach does indeed suggest that compilation times are extremely unpredictable.

However, it is less clear that the enumerative approach really does lead to an intractably large rule space, once we consider the idea of using grammar transformations to pre-process the grammar before performing non-deterministic expansion. We have evaluated two such grammar transformations, each of which reduces the size of the total rule space while accepting the same language. The two reductions are *boolean valued feature reduction* (BVFR) and *singleton variable elimination* (SVE). Table 2 shows the size of the enumerative rule space with and without these grammar transforms.

The BVFR transform undoes the effect of Mellish's representation [5] by generating multiple grammar rules for each consistent atomic value of a boolean valued feature. For example, if a rule contained a feature $F = a \vee b \vee c$, then it would be replaced by 3 rules, one each for $F = a$, $F = b$, and $F = c$. That this reduces the size of the total rule space can be seen by considering a rule which has two daughters, one contain a boolean valued feature of N atomic values, and the other containing a boolean valued feature of M atomic values. The number of rule instantions allowed using the technique of [5] is on the order of $2^N * 2^M$, while the number of rule instantitations using atomic values only is on the order of $N * M$. As can be seen in Table 2, for one of our sample grammars BVFR reduced the number of total rules from 1.1x10^{17} to 5.5x10^7.

Singleton variable elimination (SVE) has been described elsewhere [2], so we content ourselves with a brief summary. The idea is to replace any singleton variables in the daughters of a grammar rule (so called *don't care* variables) with a unique atomic value 'ANY', and then introduce copies of grammar rules and lexical items that would have unified with the original daughter, with the corresponding feature value in the copy replaced by 'ANY'. SVE reduces the mulitplicative effect of combining feature values for features that a grammar rule does not care about. We use a varient of the technique described in [2], where, instead of introducing copies of grammar rules and lexical items, we introduce instead new unit productions deriving the unifying daughter from the don't care daughter. As can be seen in Table 2, for one of our sample grammars SVE reduced the number of total rules from 5.5x10^7 to 29,342.

3. Efficient filtering of CFGs

The CFG produced by the expansion stage is not the final result, since it generally contains many rules which are irrelevant. Our experiments suggest that the larger the grammar generated by the expansion process, the larger the proportion of irrelevant rules. For the largest grammars we have tried, well over 90% of the generated rules have turned out to be irrelevant. Filtering is an essential part of the compilation process, if only for the practical reason that unfiltered grammars are in general not capable of compilation by the Nuance grammar compiler.

The standard method for filtering a CFG ([6], pp. 87–90) has quadratic complexity. However this filtering can be done in linear-time. A linear time algorithm for performing this task is outlined below.

The basic idea is to find the set of all CFG categories which lack support, in the following obvious sense. A category C lacks support if either i) there are no rules in which C is mother, or ii) for all rules in which C is mother, at least one daughter lacks support. The algorithm uses this definition to compute the set of categories that lack support, starting with the base cases and working backwards until it reaches a fixed-point.

We build a table called $RulesByDaughters$ that indexes the rules by the daughter categories. So if C is a category, $RulesByDaughters(C)$ returns the set of rules in which C is a daughter. This means that rules will be indexed as many times as they have daughters. If D is the total number of daughters, $RulesByDaughters$ can evidently be built in $O(D)$ time. We also build a table $UnsupportedRules$ which associates each rule with a boolean value (initially $false$), and another table $CatSupport$ which initially associates each category C with the number of rules in which C is head. Evidently, $UnsupportedRules$ and $CatSupport$ can also be built in $O(D)$ time. We extract the set of all categories C such that $CatSupport(C) = 0$. We assign the variable $CurrentCats$ to this set.

We now start the main loop. On each pass through the loop, we do the following:

1. Use $RulesByDaughters$ to find the set of all categories C' such that C' is the mother of a rule R with a daughter in $CurrentCats$, and also such that $UnsupportedRule(R) = false$. Call this set $TmpCats$.
2. Set $UnsupportedRule(R) = true$ for each R found in (1).
3. Set $CatSupport(C) = CatSupport(C) - 1$ for each category in TmpCats.
4. Set $CurrentCats$ to the subset of $TmpCats$ consisting of C such that $CatSupport(C)$ is now equal to zero.

We iterate until we reach a fixed-point.

It is easy to see that the algorithm has complexity $O(D)$ when we consider that the innermost loop consists of using $RulesByDaughters$ to move backwards from an unsupported daughter category to one of its mothers, removing support from the relevant rule. Each rule can at most be accessed in this way as many times as it has daughters, and thus the innermost loop can only be traversed at most a total of D times. This linear-time result is not surprising, since this filtering problem is equivalent to the problem of determining satisfiability of a Horn theory, a problem for which Dowling and Gallier [7] have given a linear-time algorithm.

4. The REGULUS compiler

REGULUS is a prototype system that implements the enumerative compilation method. It is written in SICStus Prolog, and compiles unification grammars written in an extended version of Definite Clause Grammar into annotated CFG grammars written in Nuance Grammar Specification Language (GSL; [4]). A resulting GSL grammar can be compiled into a Nuance recognition package using the `nuance-compile` utility. The REGULUS notation, which is closely based on the notation used in

```
feature_value_space(num_value, [[sing, plur]]).
feature(num, num_value)
category(sigma, [gsem]).
category(np, [sem, num]).
category(spec, [sem, num]).
category(n, [sem, num]).
top_level_category(sigma).

sigma:[gsem=[value=S]] --> np:[sem=S].

np:[sem=[spec=S, n=N], num=Num] -->
    spec:[sem=S, num=Num], n:[sem=N, num=Num].

spec:[sem=the, num=(sing\/plur)] --> the.
spec:[sem=a, num=sing] --> a.
spec:[sem=2, num=plur] --> two.
n:[sem=dog, num=sing] --> dog.
n:[sem=dog, num=plur] --> dogs.
```

Figure 1: Toy REGULUS grammar for noun-phrases

```
SIGMA  NP_ANY:v_0 { < value $v_0 > }

NP_ANY [
( NP_PLUR:v_0 ) {return( $v_0 )}
( NP_SING:v_0 ) {return( $v_0 )}]

NP_PLUR [
( SPEC_PLUR:v_0 N_PLUR:v_1 )
  {return([< spec $v_0 > < n $v_1 >])}}]

NP_SING [
( SPEC_SING:v_0 N_SING:v_1 )
  {return([< spec $v_0 > < n $v_1 >])}]

SPEC_PLUR [ two {return( 2 )}
            the {return( the )} ]

SPEC_SING [ a {return( a )}
            the {return( the )} ]

N_SING [ dog {return( dog )} ]

N_PLUR [ dogs {return( dog )} ]
```

Figure 2: Compiled GSL version of toy grammar

the Gemini system [1], is illustrated by the minimal example grammar in Figure 1, which can accept a few NPs like "the dog" or "two dogs"; this compiles into the GSL grammar shown in Figure 2.

The initial feature_value_space declaration specifies that the range of possible values in the feature-space num_value is the pair {sing, plur}, and the following feature declaration specifies that num is a feature taking values in the space num_value. The next four declarations specify the category symbols sigma, np, spec and n, with their associated features. The distinguished features sem and gsem translate into Nuance semantic annotations in a straightforward way; as can be seen by comparing Figures 1 and 2, sem features translate into local return constructions, while gsem features translate into global slot-filling. The top_level_category declaration specifies a start-symbol for the grammar.

The remainder of the grammar comprises the actual rule-set. The num feature is used to enforce agreement between SPEC and N (thus for example blocking NPs like "*a dogs"); this is realised in the compiled GSL form as different singular and plural versions of each rule. Note the disjunctive feature value for num on the lexical entry for "the", which can be either sing or plur; the formalism allows arbitrary Boolean constraints on features. Note also the first and second rules in the compiled grammar, where the non-terminal NP_ANY (" NP with ANY value for the num feature") has been generated by the SVE transform described in Section 2.

5. Empirical results

This section presents results of a series of experiments in which we compiled several different versions of a non-trivial Swedish unification grammar into CFG language models. Tests were carried out using both REGULUS and Gemini [1], in order to be able to compare REGULUS with an established system. The grammars were originally written in REGULUS format, and automatically translated into Gemini notation using a simple Prolog-based tool.

The Swedish grammars used for the test came from the Advanced House project [8], and define a fairly rich command and query vocabulary for a set of devices installed at the Telia Vision Center, Stockholm. The devices are controlled through a LonWorks network, and comprise ceiling lights with on/off switches, dimmer lights, spotlights, heaters, and light and temperature sensors. All versions of the grammar include coverage of commands ("Tänd lampan i köket tack" *[Switch on the light in the kitchen please]*, Y-N and WH-questions ("Är lampan tänd i köket?" *[Is the lamp switched on in the kitchen?]*, "Finns det någon lampa i hallen?" *[Is there any light in the hall?]*, "Vilka lampor lyser?" *[Which lights are on?]*), conjunction ("Tänd lamporna i TV-rummet och i flickrummet" *[Switch on the lights in the TV room and in the girl's room]*, pronouns ("Släck dem igen" *[Switch them off again]*), and ellipsis ("Tänd lampan i flickrummet... och så lampan i allrummet" *[Switch on the light in the girl's room... and then the light in the living room]*).

The full grammar is divided up into several pieces, with the separation being carried out in such a way that each type of device is associated with the specific extra linguistic knowledge needed for its control and query vocabulary. The four grammars used in this paper are composed of these pieces and bear the following relationship to each other: **Basic** ⊂ **BasicDim** ⊂ **AllLights** ⊂ **Full**.

Table 1 presents statistics about these four grammars, following which Table 2 summarises the sizes of the various search spaces. As can be seen, the use of the BVFR and SVE transforms greatly reduces the size of each search space. The main reason for the increase in the size of the spaces as more components are added is not the larger number of grammar rules and lexicon entries, but rather the increase in the size of the feature spaces as new semantic types of object are introduced.

The following two tables present quantitative results for running the two compilers on each of the grammars. (Times refer to SICStus 3.8.4 on a Sun Ultra 10 300 MHz). Table 3 gives the figures for REGULUS, and Table 4 for Gemini. Note that compilation time per expanded CFG rule for REGULUS stays almost constant as the size of the enumerative search space increases, confirming that the algorithms are indeed close to

Version	Rules	Lex	Largest Spaces
Basic	45	116	4, 3, 3, 3, 3
BasicDim	59	184	6, 6, 6, 6, 4
AllLights	62	197	7, 7, 7, 7, 4
Full	64	201	9, 9, 9, 9, 4

Table 1: Statistics for four versions of the Swedish Advanced House grammar. "Rules" = number of REGULUS grammar rules; "Lex" = number of REGULUS lexical entries; "Largest Spaces" = sizes of the five largest feature spaces.

Version	Total Rule Space	+BVFR	+SVE
Basic	$3.84\mathrm{x}10^{9}$	$7.04\mathrm{x}10^{5}$	$2.46\mathrm{x}10^{3}$
BasicDim	$2.51\mathrm{x}10^{13}$	$1.09\mathrm{x}10^{7}$	$8.56\mathrm{x}10^{3}$
AllLights	$4.18\mathrm{x}10^{14}$	$2.01\mathrm{x}10^{7}$	$1.10\mathrm{x}10^{4}$
Full	$1.08\mathrm{x}10^{17}$	$5.46\mathrm{x}10^{7}$	$1.70\mathrm{x}10^{4}$

Table 2: Sizes of rule search spaces for four versions of the Swedish Advanced House grammar as output undergoes the BVFR transform followed by the SVE transform.

linear-time.

Table 5 compares recognition results using models compiled by Regulus and Gemini from the full Swedish AH grammar. Tests were run on the 728 utterances from the AH corpus [8] that were inside grammar coverage.

6. Conclusions

We have presented a minimal enumerative approach to the problem of compiling typed unification grammars into CFG language models, a prototype implementation in the form of the REGULUS system, and the results of experiments in which REGULUS was used to compile some non-trivial unification grammars. Our overall impression is that enumerative methods are considerably more useful than has been previously believed, and may well form a viable alterative to sophisticated deductive methods. In particular, enumerative methods have clearly defined complexity characteristics, making it easy to produce a rapid estimate of the time needed to compile a given unification grammar into a CFG.

Irrespective of the final verdict concerning the choice between enumerative and deductive methods, the extreme simplicity of enumerative methods makes them a natural baseline against which to compare alternative approaches; ability to consistently outscore a well-engineered enumerative method lends empirical substance to claims about the utility of more sophisticated ideas. Our current plan is to continue our programme of further developing REGULUS and comparing its performance with that of the deductive Gemini system; in particular, we should soon be in a position to experiment with using REGULUS to compile some of the large grammars developed under the Gemini projects. We hope to present the results of these experiments in due course.

Version	Time	Time/rule		Rules
		Expand	Filter	
Basic	1.3	0.23	0.05	498
BasicDim	5.1	0.23	0.05	916
AllLights	6.7	0.24	0.05	1240
Full	10.8	0.26	0.05	1801

Table 3: Figures for REGULUS compilation of four versions of the Swedish Advanced House grammar. "Time" = total compilation time in seconds; "Time/rule" = time per expanded CFG rule for each phase in msecs/rule; "Expand" = expansion phase; "Filter" = filtering phase; "Rules" = number of CFG rules in final GSL grammar.

Version	Time	Rules
Basic	2.4	474
BasicDim	3.6	712
AllLights	10.8	1137
Full	14.3	1336

Table 4: Figures for Gemini compilation of four versions of the Swedish Advanced House grammar. "Time" = total compilation time in seconds; "Rules" = number of CFG rules in final GSL grammar.

	WER(%)	SER(%)	xRT
Regulus	8.64	14.70	0.257
Gemini	8.31	14.29	0.209

Table 5: Regulus vs Gemini recognition results (word and sentence error rates) on 728 utterances

7. References

[1] J. Dowding, M. Gawron, D. Appelt, L. Cherny, R. Moore, and D. Moran, "Gemini: A natural language system for spoken language understanding," in *Proceedings of the Thirty-First Annual Meeting of the Association for Computational Linguistics*, 1993.

[2] R. Moore, "Using natural language knowledge sources in speech recognition," in *Proceedings of the NATO Advanced Studies Institute*, 1998.

[3] B. Kiefer and H. Krieger, "A context-free approximation of head-driven phrase structure grammar," in *Proceedings of the 6th International Workshop on Parsing Technologies*, 2000, pp. 135–146.

[4] Nuance Communications, *Nuance Speech Recognition System Developer's Manual version 6.2*, 1380 Willow Road, Menlo Park, CA 94025, 1999.

[5] C. Mellish, "Implementing Systemic Classification by Unification," *Computational Linguistics*, vol. 14, no. 1, pp. 40–51, 1988.

[6] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Addison-Wesley, Reading, Massachusetts, 1979.

[7] William F. Dowling and Jean H. Gallier, "Linear-time algorithms for testing the satisfiability of propositional horn formulae," *The Journal of Logic Programming*, vol. 1, no. 3, pp. 267–284, 1984.

[8] M. Rayner, G. Gorrell, B.A. Hockey, J. Dowding, and J. Boye, "Do cfg based language models need agreement constraints?," in *Proceedings of 2nd NAACL*, 2001.