NEALT

Northern European Association for Language Technology

NEALT Proceedings Series Vol. 33

Proceedings of the NoDaLiDa 2017 Workshop on Constraint Grammar - Methods, Tools, and Applications

NoDaLiDa

22 MAY 2017
GOTHENBURG SWEDEN
CONFERENCE CENTRE WALLENBERG

Edited by Eckhard Bick & Trond Trosterud

Proceedings of the NoDaLiDa 2017 Workshop on Constraint Grammar - Methods, Tools and Applications

edited by
Eckhard Bick and Trond Trosterud

22 May 2017
Gothenburg

Proceedings of the NoDaLiDa 2017 Workshop on Constraint Grammar - Methods, Tools and Applications
Edited by Eckhard Bick and Trond Trosterud
NEALT Proceedings Series 33
ISBN: 978-91-7685-465-5

Linköping Electronic Conference Proceedings No. 140
ISSN: 1650-3686, eISSN: 1650-3740

Photo front cover: Kjell Holmner, Göteborg & Co

Preface

This workshop on practical and theoretical aspects of CG was co-located with NoDaLiDa 2017 in Gothenburg, and held on the 22nd of May, 2017. The latest edition of the workshop has been number 7 in a row of CG workshops at NoDaLiDa, unbroken since 2005, and emphasizing the Nordic roots of Constraint Grammar.

Apart from the traditional CG field of corpus-oriented tagging and parsing, there is a growing body of applicational work, where CG provides the backbone of end user-oriented systems in various areas of language technology, such as spell and grammar checking, comma correction, ICALL, machine translation, lexicography and others. We therefore invited workshop contributions both regarding basic grammatical research and corpus linguistics on the one hand, and CG-based applications on the other hand. CG has always elicited a strong interest from researchers working on less-resourced languages, and we therefore expected papers targeting minor languages, such as the Sami languages, Greenlandic, Faroese, Tibetan and the Celtic languages.

However, contrary to expectations and in stark contrast to the 2015 workshop, neither minority languages nor applications, but another target area - methodological research - turned out to completely dominate this year's submissions. Thus, papers explored topics like automatic rule creation and rule optimization as well as what one might call CG typology - the expressivity and power of the formalism as such. Another methodological issue was cross-platform compatibility for CG dependency corpora. Where these papers did touch on language, examples were drawn - with the notable exception of Basque - from only a few major languages, English, Spanish and Portuguese. Given the de-facto wide language-spread of ongoing CG work, this under-representation of languages represents a challenge to future CFP's. In the same vein, it is also an interesting question, whether theoretical issues are more paper-motivating simply because they involve 100% research, while researchers with application-oriented work like machine translation will have to choose between either presenting in a CG forum or (as evidently happened this year) in a workshop or conference section for the relevant application itself.

On behalf of the workshop organizers
Eckhard Bick & Trond Trosterud

Workshop organizers

- Eckhard Bick, Research lector, Institute of Language and Communication, University of Southern Denmark
- Tino Didriksen, Developer, GrammarSoft ApS
- Kristin Hagen, Senior engineer, Tekstlaboratoriet, University of Oslo
- Inari Listenmaa, Ph.D. student, University of Gothenburg and Chalmers University of Technology
- Kaili Müürisep, Senior research fellow, Institute of Computer Science, University of Tartu
- Trond Trosterud, Assistant professor in Sámi computational linguistics, University of Tromsø

Program Committee

- Eckhard Bick (Chair)
- Kristin Hagen
- Inari Listenmaa
- Kaili Müürisep
- Anders Nøklestad
- Trond Trosterud

Workshop website

https://visl.sdu.dk/nodalida2017.html

Contents

Automatic Synthesis of Constraint Grammar Rules using a Greedy Approach and a SAT-solver
-- a work in progress report

Koen Claessen
Chalmers University of Technology,
Sweden
koen@chalmers.se

Abstract

We present a method for automatic synthesis of Constraint Grammar rules from a corpus. The method is designed to aid grammar writers interactively in coming up with new rules or improving existing ones, but also to synthesize a whole set of rules autonomously from scratch. A SAT-solver is used to compute the "best" rule at each stage, according to some measure. Initial experimental results on two corpora look promising: suggesting one rule to improve an existing set of rules typically takes seconds; synthesizing a whole set of rules takes minutes to hours.

1 Introduction

Imagine a CG grammar writer who finds herself in the following situation: There exists a gold standard development corpus, and already a set of rules. Now, the CG writer would like to add a new rule, or improve an existing one. Where to start?

This work describes a tool that can automatically find the next "best" rule to add to an existing set of rules. It deploys a SAT-solver that computes the (1) simplest, most general rule that (2) does not remove any correct reading from the corpus, and (3) maximizes the number of removed incorrect readings from the corpus.

For example, given a spanish corpus and an empty set of rules, the tool may compute the following (rather reasonable) rule:

```
SELECT (det) IF (1 (n));
```

The reason is because this is the rule that removes most ambiguities without removing any correct readings from the corpus. A human grammar writer may also have included this rule high up in the grammar, which is why we call this rule reasonable.

Asking the tool to generate one more rule and then yet another, the following two rules are generated:

```
REMOVE (vblex) IF (-1 (det));
REMOVE (sg) IF (0 (pr));
```

The first one looks again reasonable. The second rule does not. Obviously, there are sentences in which the second rule would remove the correct reading, but none of those sentences appears in our corpus.

2 Three use cases

We envision the use of our tool for grammar development in three different use cases.

A. Interactive mode - in this mode, the grammar writer can ask questions such as: "What is the most general rule I can add to my existing set of rules?" and "Is there a rule that looks like <this> that does not remove any correct readings from the corpus?" and get answers. The grammar writer is in complete control over the grammar and can add any versions of these generated rules anywhere in the grammar.

B. Meta-mode - The tool suggests rules to add to an existing grammar, but the grammar writer inspects the rules, and when the rules do not look good, *adds new sentences to the corpus*. For example, in the case of the third rule from the previous section, the corpus could be augmented with a sentence where that rule would remove the correct reading. Running the tool again would then suggest a different rule.

C. Autonomous mode - The tool starts from an empty set of rules, and greedily computes a set of rules from scratch. Typically, the resulting set of rules is perfect (or almost perfect) on the given corpus, meaning a precision and recall of 100% (or close to). The hope is then that these rules can be generalized to other corpora. Our very preliminary experiments suggest this may be the case.

3 Implementation

Our method only generates rules that satisfy the following:

- Either a SELECT or REMOVE rule
- The head is a set of tags
- The condition is a conjunction of zero or more conditions of the form (i tag_1 .. tag_n) or (iC tag_1 .. tag_n), for i chosen from a window (typically {-2,-1,0,1,2} or {-1,0,1}) and the tag_j can be any tag.

Some of these constraints can be relaxed a bit (for example adding more kinds of rules, or adding BARRIER conditions, but we have not done this yet).

We use a SAT-solver [4] to compute the "best" rule in the following way. First, we run any existing rules on the given corpus. Then, we create a SAT-problem with the following variables:

- A SAT-variable sel, indicating which kind of rule we have,
- For each possible tag t, a SAT-variable h_t, representing the head of the rule,
- For each position i in the window and tag t, a SAT-variable $c_{i,t}$, representing whether or not that tag appears in the condition with that position,
- For each position i in the window, a variable C_i, representing whether the condition for position i is careful or not,
- For each ambiguous cohort w from the corpus, a SAT-variable a_w, representing if the rule removes any reading from that cohort.

Then, we add the following constraints:

- There must be at least one head,
- The rule should never remove a correct reading,
- There must be at least one cohort for which we remove a reading.

Finally, we look for solutions to these constraints, applying an optimization strategy that optimizes, in this order:

- Maximize the number of cohorts where the rule removes a reading,
- Minimize the number of tags appearing as conditions,
- Minimize the number of C-conditions,
- Minimize the number of tags in the head,
- Prefer SELECT over REMOVE.

The solution to the SAT-problem is a model that assigns true or false to all of the variables. The actual rule can easily be constructed from that model.

4 Preliminary Results

Our preliminary results contain experiments on 3 corpora: One Spanish corpus with ~21.000 words, taken from the Apertium repositories. It contains news texts that have been hand-tagged by students. One Basque corpus with ~61.000

words, called EPEC. And one English corpus with ~29.000 words, taken from the minutes of the EU parliament meetings.

	#tags	avg. #readings
Spanish	116	1.39
Basque	203	2.84
English	78	1.48

The number of tags and average number of readings per cohort are given in the table above.

We ran the interactive mode on both corpora, and most questions could be answered within seconds to sometimes minutes.

We also ran the autonomous mode on the first 3000 words of the Spanish corpus, which took ~10 minutes (the resulting 88 rules are shown in the appendix). Although this is only 1/7th of the total corpus, the evaluation of this grammar on the rest (6/7th) of the corpus, yielded 96% correct readings after full disambiguation[1]! This fraction of correctness is on par or better than existing hand-written grammars we had for this corpus. Increasing the size of the training data to 16.800 words (80% of the corpus) yielded a grammar with 97% correct readings for the other 20%.

We also ran the autonomous mode on the first 2500 words of the Basque corpus (4%), which took a bit over 1 hour. The evaluation of the resulting grammar on the rest of the corpus yielded 79% correct readings after full disambiguation. A result that may be disappointing, but it is better than the CG we had at hand, written by humans. Basque turns out to be more computationally expensive than Spanish, mostly because it has more tags and more ambiguity in its readings. Also, word order in Basque is less strict than Spanish.

We were able to increase the size of the training corpus for Basque to 30.000 words (almost 50% of the corpus) by using non-exact optimization methods for picking the best rule. This yielded a grammar with 84% correct readings on the other half of the corpus.

For English, we reached close to 97% with 3000 words. Adding more words did not significantly improve the quality of the generated grammar.

In all our experiments, we used a window of {-1,0,1}, and no limit on the size of the conditions. For Spanish and English, we also tried a window {-2,-1,0,1,2}, which took ~3 times longer time, but did not produce a significantly better grammar. Only a handful of rules actually made use of distance -2 or 2. For Basque, it took too long time to run with a larger window.

5 Discussion and Conclusion

Our experiments are promising but far too few to draw any significant conclusion. We believe that we have enough evidence to suggest that our work may be a useful tool. One important factor that is limiting our experiments is the lack of good quality corpora to use for our experiments!

We have not studied the usefulness of the "interactive mode" or "meta mode" that would help a CG writer who is looking for suggestions on what to do next. This is left as future work.

This work is not the first to propose automatic generation of disambiguation rules from a corpus; in fact there has been work on this since the 1990s [1,2,3]. Our work differs from the ones that generate CG rules in two significant ways: (1) our greedy approach computes rules that remove as much ambiguity without also removing correct readings, as opposed to earlier work that is based on statistics, and (2) we use a SAT-solver to compute rules which avoids enumeration of rules and allows for a

[1] This precentage is both the precision and the recall after making a random choice for all ambiguities that are still left after running the rules.

constraint-based specification of what kind of rules we are looking for, which is more flexible.

We argue that an approach that avoids rules that remove correct readings is the reasonable choice in a setting where rules are generated greedily one-by-one. Allowing such rules to remove correct readings would be non-compositional; a later rule can never correct such a mistake made by an earlier rule.

6 References

[1] Samuelsson, Tapanainen, Voutilainen. Inducing Constraint Grammars. International Colloquium on Grammatical Inference. 1996.

[2] Lindberg, Eineborg. Learning Constraint-grammar style disambiguation rules using Inductive Logic Programming. COLING. 1998.

[3] Sfrent. Machine Learning of Rules for Part of Speech Tagging. MSc. thesis. Imperial College London. 2014.

[4] Een, Sörensson. An Extensible SAT-solver. SAT conference. 2003.

Appendix - Generated CG for Spanish

```
SELECT (det) IF (1 (n));
REMOVE (vblex) IF (-1 (det));
REMOVE (sg) IF (0 (pr));
REMOVE (n) IF (-1 (n)) (0 (adj));
REMOVE (imp) IF (-1 (sg));
REMOVE (p3) IF (-1 (pr));
SELECT (n) IF (-1 (det)) (-1 (m));
REMOVE (p1 prs) ;
REMOVE (p3) IF (0 (p1));
REMOVE (p2) IF (0 (n));
REMOVE (vblex) IF (1 (vblex));
REMOVE (cnjsub) IF (-1C (n));
SELECT (al) IF (0 (ant));
REMOVE (p1) IF (0 (m));
REMOVE (rel) IF (-1 (vblex));
REMOVE (sg) IF (0 (vbser));
SELECT (pri) IF (-1 (mf));
SELECT (np) IF (-1 (np));
REMOVE (adj) IF (-1 (pr)) (0 (n));
SELECT (det) IF (1 (adj));
SELECT (mf) IF (1 (p3));
REMOVE (mf) IF (0 (n));
SELECT (mf) IF (0 (vblex));
REMOVE (sp) IF (1C (adj));
SELECT (f) IF (1 (pr));
SELECT (vblex) IF (-1C (n));
SELECT (pr) IF (1 (m));
SELECT (np) IF (-1 (pr));
REMOVE (n sg) IF (1 (cm));
REMOVE (p3 sg) IF (0 (m));
SELECT (n sg) IF (-1 (f));
SELECT (np) ;
REMOVE (imp p3) ;
REMOVE (adj pl) IF (-1 (mf));
REMOVE (pl prn) IF (-1C (pr));
SELECT (ind sp) ;
SELECT (ind) IF (0C (f));
SELECT (pri) IF (1C (m));
REMOVE (sp) IF (0 (sg));
SELECT (n) IF (-1 (cnjcoo));
SELECT (adj) IF (1 (sent));
REMOVE (adj m sg) IF (0 (n));
REMOVE (m n) IF (-1C (sg));
SELECT (m) ;
REMOVE (p3) IF (-1C (sent));
SELECT (pri) IF (0C (vblex));
SELECT (pp) IF (-1C (vbser));
REMOVE (pp) IF (1 (pr));
REMOVE (mf n) ;
REMOVE (adj m) ;
REMOVE (det ind sg) ;
REMOVE (mf) IF (-1 (adv));
REMOVE (prn) IF (1 (m));
SELECT (adv) IF (1 (pr));
REMOVE (vblex) IF (-1 (pr));
SELECT (pp) ;
REMOVE (adv) IF (1C (sg));
REMOVE (vblex) IF (1C (sg));
SELECT (pri) ;
SELECT (adv) IF (1C (vblex));
SELECT (mf) IF (0 (adj));
SELECT (mf) IF (-1 (cm));
REMOVE (detnt) ;
REMOVE (al) ;
SELECT (cnjsub) IF (1 (f));
SELECT (pr) IF (1 (det));
SELECT (rel) IF (1 (pii));
SELECT (adv) IF (1 (pl));
SELECT (sp) IF (-1C (mf));
REMOVE (n) IF (-1 (sg));
REMOVE (adj) ;
REMOVE (cnjadv) IF (1 (np));
REMOVE (cnjcoo) ;
REMOVE (sent) ;
REMOVE (pr) ;
SELECT (vblex) ;
REMOVE (n) ;
SELECT (ind sg) ;
REMOVE (det) IF (1C (pri));
REMOVE (prn) IF (1 (sg));
SELECT (cnjsub) IF (1 (dem));
SELECT (an) IF (1 (det));
REMOVE (rel) IF (-1 (sg));
SELECT (an) ;
REMOVE (loc) IF (1 (rpar));
REMOVE (ant) ;
SELECT (prn) IF (-1 (p3));
SELECT (ind) ;
```

Using Constraint Grammar for Treebank Retokenization

Eckhard Bick
University of Southern Denmark
eckhard.bick@mail.dk

Abstract

This paper presents a Constraint Grammar-based method for changing the tokenization of existing annotated data, establishing standard space-based ("atomic") tokenization for corpora otherwise using MWE fusion and contraction splitting for the sake of syntactic transparency or for semantic reasons. Our method preserves ingoing and outgoing dependency arcs and allows the addition of internal tags and structure for MWEs. We discuss rule examples and evaluate the method against both a Portuguese treebank and live news text annotation.

1 Introduction

In an NLP framework, tokenization can be defined as the identification of the smallest meaningful lexical units in running text. Tokens can be both words, symbols or numerical expressions, but there is no general consensus on what constitutes a token boundary. For instance, are "instead of" or "Peter Madsen" 1 or 2 tokens? Should German "z. B." (for example) be 2 tokens and English "e.g." 1 token, just because the former contains a space? What about a word that allows optional space (*insofar* as vs. *in so far as*)? Far from being a merely theoretical issue, tokenization conventions strongly influence parsing schemes and results (e.g. Grefenstette & Tapanainen 1994). Thus, contextual rules become simple (and therefore safer) when faced with single-token names, conjunctions and prepostions rather than complex ones. Conversely, contractions such as Portuguese "na" (= em [in] a [the]) can only be assigned a meaningful syntactic analysis when split into multiple tokens, in this case allowing the second part (the article) to become part of a separate np.

Tokenization is often regarded as a necessary evil best treated by a preprocessor with an abbreviation list, but has also been subject to methodological research, e.g. related to finite-state transducers (Kaplan 2005). However, there is little research into changing the tokenization of a corpus once it has been annotated, limiting the comparability and alignment of corpora, or the evaluation of parsers. The simplest solution to this problem is making conflicting systems compatible by changing them into "atomic tokenization", where all spaces are treated as token boundaries, independently of syntactic or semantic concerns. This approach is widely used in the machine-learning (ML) community, e.g. for the Universal Dependencies initiative (McDonald et al. 2013). The method described in this paper can achieve such atomic tokenization of annotated treebank data without information loss, but it can also be used for grammar-based tokenization changes in ordinary annotation tasks, such as NER.

2 Retokenization challenges

What exactly atomic (space-based) retokenization implies, is language-dependent, and may involve both splitting and fusion of tokens, for fused multi-word expressions (MWEs) and split contractions, respectively. While the former, not least for NER, is a universal issue, the latter is rare in Germanic languages (e.g. *aren't, won't*), but common in Romance languages. In both cases, the retokenization method should conserve existing information, i.e. MWE boundaries in one case, and morphosyntactic tags of contraction parts in the other. Linguistically, token-splitting is the bigger problem, because it needs *added*

information: (a) partial POS tags, (b) additional internal dependency links and (c) new internal hook-up points for existing outgoing and incoming dependency links. Unlike simple tag conversion for, say, morphological features, this cannot be achieved with a conversion table.

3 CG retokenization

Our solution is based on two unique features of the CG3 compiler (Bick & Didriksen 2015). The first allows context-based insertion, deletion and substitution of cohorts (token + 1 or more readings), and was originally intended for spell- and grammar-checking. Thus, we implemented token fusion by either inserting a (new) fused token and then removing all original tokens, or by substituting a token with a larger, fused one containing the subsequent token (rules 1), then removing the latter (rule 2). The other feature introduces cohort splitting rules and was added specifically for retokenization. Such a rule can specify how to split a target token and manipulate its parts using regular expression matching (rule 3). In a separate rule field, a dependency chain is stipulated across the split token.

3.1 Multi-word expressions

How an MWE is to be split, obviously depends on its POS and composition. A simple case are name chains entirely made up of proper nouns. Here, (part) lemmas equal (part) tokens, and internal structure is simply a left- (or right-) leaning dependency chain. With other word classes, however, there may be inflection and complex internal structure. The Portuguese proper noun-splitting rule (1a), for instance, breaks up TARGET named entities (NE) of the type PROP+PRP+PROP (e.g. *"(Presidente do) Conselho de Administração" [Administrative Council President]*) - if necessary, iteratively. The asterisk for part 1 means that the first part inherits all tags (pos, edge label, features) from the NE as a whole, while c->p means that it also inherits incoming child (c) and outgoing parent (p) dependencies. For parts 2 and 3, independent new POS tags (PRP, PROP) and syntactic function labels (@N<, @P<) are provided. All parts receive a numbered MWE id (<MWE1>, <MWE2> etc.), and the original MWE token is retained in a separate tag (<MWE:...>. Note that the new parts may themselves be MWEs, needing further splits. Contractions contained in a NE (*do* [of the_sg_m], *pelas* .. [by the_pl_f]) need to be split (1c), in order to be treated like other, "free" contractions in the corpus. (1c) starts with a default male singular reading which is "corrected" by (1d) into female and/or plural where necessary.

Rule (1b) targets a complex adverb, *dali para diante [from here onward, from now on]*, performing not only a 3-way split on space, but also splitting the contraction *dali* (de+ali PRP+ADV). The '*' on the first part means that it will inherit form and function tags from the MWE as a whole, and "c->p" means it will also inherit both incoming (child) and outgoing (parent) dependencies.

```
(1a) SPLITCOHORT:multipart-prop (
"<$1>"v "$1"v <MWE1><MWE:$1=$2=$3>v * c->p
"<$2>"v "$2"v <MWE2> PRP @N< 2->3
"<$3>"v "$3"v <MWE3> PROP @P< 3->1)
TARGET ("<(.+?)=(aos?|às?|com|contra|d[eao]s?|em|
n[ao]s?|para)=(.*)>"r PROP /\(@.*\)/r) ;

(1b) SPLITCOHORT:three->fourpart-adv(
"<$1e>"v "de"v <sam-> <MWE1> <MWE:
$1$2=$3=$4>v PRP VSTR:$5 1->p
"<$2>"v "$2"v <-sam> <MWE2> ADV @P< 2->1
"<$3>"v "$3"v <MWE3> PRP VSTR:$5 @P< 3->1
"<$4>"v "$4"v <MWE4> ADV @P< c->3)
TARGET ("<([dD])(ali)=(para)=([^=]+?)>"r ADV \
(@.*\)/r) ;

(1c) SPLITCOHORT (
"<por>" "por" <sam-> <MWEprp> PRP @N< c->p
"<$1>"v "o" <-sam> <artd> <MWEdet> DET M S
@>N 2->p)
        TARGET ("pel([ao]s?)"r)
        (0 PRP OR N/PROP) ;

(1d) SUBSTITUTE (M) (F)
TARGET ("<.*[aà]s?>"r <MWEdet>) ;
```

3.2 Contractions

Fusion of tokens does not add linguistic information, and a function tag can simply be inherited from the head token of the to-be-fused words. Still, CG rules like (2-3) are an effective option for this purpose, too, because the formalism will automatically handle the resulting dependency number adjustments for the rest of the tree, and morphophonetic changes can be addressed where necessary. Here, we use fusion rules to reassemble Portuguese contractions that were split into lemma parts in the original

treebank (marked <sam-> for first part and <-sam> for second part). Thus, (2a) creates a compound POS for the contraction, substituting it for the preposition POS of the contraction's first part. (2b-c) then fuse the tokens "por" and "as" into "pelas", and (2d) creates a compound lemma for the contraction. (3), finally, removes the now-superfluous second part token.

```
(2a) SUBSTITUTE (PRP) (PRP_DET)
        TARGET (<sam->)
        (1 (<-sam> DET)) ;

(2b) SUBSTITUTE
        ("<$1>"v) (VSTR:"<$1$2>")
        TARGET ("<(.*)>"r PRP_DET)
        (1 ("<(.*)>"r <-sam>)) ;

(2c) SUBSTITUTE
        ("<por$1>"v) (VSTR:"<pel$1>")
        TARGET ("<por(.*)>"r PRP_DET) ;

(2d) SUBSTITUTE ("$1"v) (VSTR:"$1+$2")
        TARGET ("([^<]+)"r PRP_DET)
        (1 ("([^<]+)"r <-sam>)) ;

(3) REMCOHORT REPEAT (<-sam>)
        (-1 (/^PRP_.*$/r) OR (PERS_PERS)) ;
```

4 Evaluation and statistics

We evaluated the CG-based retokenization method on the Portuguese Floresta Sintá(c)tica treebank (Afonso et al. 2002), a 239,899 token treebank covering the European as well as the Brazilian varieties of Portuguese. The treebank is available in both constituent and dependency formats, both adhering to the cross-language VISL annotation standard[1]. Since the treebank's native format does not adhere to atomic tokenization, as advocated by the Universal Dependency initiative, retokenization has become an issue for ML-users of the Floresta Sintá(c)tica. Our retokenizer proved capable of addressing this problem, resolving all 8779 MWEs in the treebank into their 21,954 parts (2.50 per MWE), and reestablishing all 15,912 contractions. The process took 33.6 seconds on a 2-core laptop, amounting to a processing speed of 7,140 words/sec.

In combination with a live parser run, on a Portuguese newstext corpus with ~ 1.1 million tokens, the method handled 44,826 MWEs of similar complexity (109,320 parts, 2.44 per MWE), missing out on only 273 (0.6%) MWEs. The failure rate for contractions was a negligible 0.01% (with 76,610 successful fusions).

5 Conclusions and outlook

We have shown that (cg3-level) Constraint Grammar can be used for retokenization, and that our method can establish space-based tokenization for treebanks or parser output that for syntactic or semantic reasons use different tokenization strategies. Thus, both splitting of fused multi-word-expressions and fusion of split contractions can be handled with a high degree of accuracy. In addition to retokenization itself, the method specifically supports tree strucctures in dependency treebanks, preserving ingoing and outgoing dependency arcs and allowing the addition of internal tags and dependency structure for MWEs.

Apart from the treebank conversion discussed here, we hope that the technique will prove useful at various stages of NLP pipelines, supporting grammar- and context-driven tokenization *after* the preprocessing stage, or as post-processing for named entities, numerical expressions or compound-related spelling errors. As a specialized application, we are currently experimenting with target-language retokenization in machine translation.

References

Afonso, Susana & Eckhard Bick & Renato Haber & Diana Santos. 2002. Floresta sintá(c)tica: A treebank for Portuguese. In Proceedings of LREC'2002, Las Palmas. pp. 1698-1703, Paris: ELRA

Bick, Eckhard & Tino Didriksen. 2015. CG-3 - Beyond Classical Constraint Grammar. In: Beáta Megyesi: Proceedings of NODALIDA 2015, May 11-13, 2015, Vilnius, Lithuania. pp. 31-39. Linköping: LiU Electronic Press. ISBN 978-91-7519-098-3

Grefenstette, Gregory & Pasi Tapanainen. 1994. What is a word, what is a sentence? Problems of tokenization. Proceedings of the 3rd Conference on Computational Lexicography and Text Research (COMPLEX'94), Budapest. pp. 79-87

Kaplan, Ronald M. 2005. A method for tokenizing text. In: Festschrift in Honor of Kimmo

[1] http://visl.sdu.dk

Koskenniemi's 60th anniversary. CSLI Publications, Stanford, CA. pp. 55-64

McDonald, Ryan et al. 2013. Universal dependency annotation for multilingual parsing. In: Proceedings of ACL 2013, Sofia. pp. 92-98

Cleaning up the Basque grammar: a work in progress

Inari Listenmaa
University of Gothenburg
inari@chalmers.se

Jose Maria Arriola
University of the Basque Country
josemaria.arriola@ehu.eus

Itziar Aduriz
University of Barcelona
itziar.aduriz@ub.edu

Eckhard Bick
University of Southern Denmark
eckhard.bick@mail.dk

1 Introduction

The first version of the Basque Constraint Grammar (BCG) was developed in 1995–1997 by two linguists (Aduriz et al., 1997) based on the Constraint Grammar theory of Karlsson (1990; Karlsson et al. (1995). Since then, it has undergone many changes, by many grammarians. During the two decades of development, the Basque morphological analyser has also been updated several times, and not always synchronised with the CG. As a result, the Basque grammar needs serious attention.

In the present paper, we describe the ongoing process of cleaning up the Basque grammar. We use a variety of tools and methods, ranging from simple string replacements to SAT-based symbolic evaluation, introduced in Listenmaa and Claessen (2016), and grammar tuning by Bick (2013). We present our experiences in combining all these tools, along with a few modest additions to the simpler end of the scale.

2 Previous work

Bick (2013) presents a method for automatically tuning a grammar, and reports an error reduction between 7–15 % when tested on the Danish tagging grammar. Listenmaa and Claessen (2016) present a method for detecting contradictions in a grammar, using SAT-based symbolic evaluation. They report detecting rule conflicts in a few small grammars, but provide no further evaluation on the grammars after fixing the rule conflicts. In our experiments, we use both of these tools for different purposes, complementing each other.

3 Pipeline

As a first step, we run a series of simple, mostly off-the-shelf tools. The next step is to group the rules and order them by their contextual tests. These sets are checked both by the SAT-based tool, and grammarians. After these steps, we give the grammar as an input for ML-tuning.

3.1 Simple tools

String operations Fix typos: O for 0, and various mismatched `"<>"` in word forms: e.g. `"<zuen">`, `<argi>"`. Transform word forms into case-insensitive, remove duplicates. There were many occurrences of identical rules, of the form `REMOVE ("<x>")` and `REMOVE ("<X>")`. We changed those rules into the form `REMOVE ("<x>"i)`, and removed duplicate rules after that.

Tagset operations The VISL CG-3 compiler offers useful features, such as `--show-unused-sets` and `--show-tags`. With the former, we could eliminate 255 unused tagsets, and with the latter, we detected 15 obsolete or misspelled tags in the remaining used tagsets, by comparing against an up-to-date lexical database (Aldezabal et al., 2001).

Human readability For improving the readability of the grammar, we wrote a tool that finds repetitive set definitions, and suggests ways to compact them. An example is shown below:

Original

```
("ageri" ADJ ABS MG)
("bizi" ADJ ABS MG) ...
("haizu" ADJ ABS MG) ;
```

Compact

```
("ageri"|"bizi"|"haizu") +
(ADJ ABS MG) ;
```

In addition, the grammar contains many rules that specify an inline set, when there is already the same or a very similar set definition. For instance, the rule `REMOVE (ADL) IF (0 ADT) (1 ("<.>") OR ("<;>") OR ("<,>") OR ("<:>") OR ("<?>") OR ("<!>"))` lists different punctuation marks as word forms, instead

```
SELECT ADOIN IF (1 ARAZI) ;                          # line 7412
REMOVE ADOIN IF (0 IZE) (1C ADJ) ;                   # line 6423
REMOVE ADOIN IF (0 IZE) (1 DET | ADJ | IZE) ;        # line 6433
REMOVE ADOIN IF (0 EZEZAG + ADJ + GEN) (-2C IZE) ;   # line 6319
REMOVE ADOIN IF (0 IZE) (-1C IZE) (1C ADJ) ;         # line 6422
```

Figure 1: Rules grouped by target, and ordered by their contextual tests.

of using the list PUNTUAZIOA, which contains all these tokens.

The standard tools did not provide this type of suggestions, so we wrote these tools ourselves. Neither of these transformations is applied automatically, they are just suggestions for the grammar writers.

3.2 Group by target, sort by conditions

After the simple checks and transformations, we group the rules by their targets, and sort them by the complexity of their contextual texts. For instance, the 5 rules that target ADOIN will be in the order shown in Figure 1: from fewest to most contextual tests, and in the case of same number of tests, preferring those with fewer tagsets.

3.3 Check for conflicts and redundancies

When the rules are grouped and sorted as described, we run SAT-based symbolic evaluation (Listenmaa and Claessen, 2016) on each group. If it says that some rule with a more complex condition is superfluous because of another rule earlier in the list[1], then that is a hint for the grammar writer: why are there two similar rules in the grammar, if the simpler one would do? At the moment of writing, we are still looking for better ways to adapt the conflict check to the Basque grammar; due to the large number of tags, we cannot use the system straight out of the box. We describe the adaptations we have done so far in Section 4.2, as well as some preliminary results.

3.4 Manual cleanup

Even if the program wouldn't find any conflicts, we give the rules to a grammarian in any case. The grammarian works with this list, having the original grammar on the side to see the comments, or other original context of any given rule. On the one hand, seeing all the rules grouped helps with the situation where different grammarians have written rules independent of each other. On the other hand, working with the ordered grammar makes it difficult to compare the precision, recall and F-score to the original grammar in the intermediate stages of the cleanup.

In any case, the sorting and grouping is not meant to be the final order, it is only to help a human grammarian to make decisions regarding all the rules that target the same tagsets. An easy alternative would be to work on the original grammar instead, only keeping the sorted list as a help and generating new ones as the cleanup proceeds. However, we found it easier to work directly on the sorted list. To solve the problem of intermediate evaluation, we decided to compare the results by ML-tuning both the original and the ordered grammar.

3.5 ML-tuning

Once the grammar has gone through the previous steps, we give it to the ML-tuning tool (Bick, 2013), with the purpose of finding an optimal order. Then we can run the newly ordered grammar through the conflict check, to detect if the ML-tuning has introduced new conflicts or superfluous rules.

Our initial hypothesis is that the human-cleaned version will benefit more from tuning than the original grammar. Some bad rules may have only a minor problem, such as a single tag name having changed meaning, and they would be better fixed by updating the obsolete tag, instead of the whole rule being demoted or killed. To test our assumptions, we tune both the original grammar and the human-cleaned versions, continuously comparing the new versions to the original.

3.6 Final order

After checking the conflicts and redundancies of the grammar, we will proceed to reorder the grammar by defining the sections of the grammar corresponding to each level of granularity of the Basque tag set.

[1] For example, the latter rule of the following is superfluous: `REMOVE Verb IF (-1 Det)` and `REMOVE Verb IF (-1 Det) (1 Verb)`

4 Evaluation

We evaluate the grammars with a manually disambiguated corpus of 65,153 tokens/53,429 words, compiled from different sources (Aduriz et al., 2006). We report the original score, and the result from ML-tuning the original grammar, as well as the result of preliminary cleanup. The scores are given using two metrics, differing on the granularity of the tagset.

4.1 Evaluation criteria

The Basque tag set is structured in four levels of granularity. As explained in Ezeiza et al. (1998), the first level contains only the main POS, 20 distinct tags, and the fourth level contains several hundreds of tags with fine-grained distinctions, including semantic tags such as animacy. Table 1 shows a simplified example of the levels for nouns. On the 4th level, the initial ambiguity is very high: the test corpus has, on average, 3.96 readings per cohort. On the 2nd level, when readings that differ only in higher-level tags are collapsed into one, the initial ambiguity is 2.41 readings per cohort. We follow the scheme for evaluation: assume that we are left with two readings, "Common noun, singular" and "Common noun, plural", and one of them is correct. Evaluation on levels 3 and 4 reports 100 % recall and 50 % precision. Evaluation on levels 1 and 2 ignores the tags from the higher levels, and regards any common noun or noun as correct, hence 100 % for both measures.

It should be noted that the linguistic revision has been targeted towards improving the 2nd level.

4.2 Analysis of the results

The results of the preliminary evaluation are in Table 2. The drop in performance after the preliminary cleanup is most certainly due to ordering—we found it easier to work on the grammar directly after grouping and sorting the rules, as shown in Figure 1. ML-tuning the cleaned grammar brings the precision up, indicating that more rules get to fire in the tuned order. The difference is most dramatic in the sorted and grouped grammar on the 4th level: the original precision drops from 62 % to 56 %, and goes up to 68 % with the ML-tuning.

As explained in Section 3.4, the fairest test at this stage is to compare the ML-tuned results of the original and the cleaned grammar. We see the cleaned and tuned grammar slightly outperforming the tuned original; the difference is not large, but we see it as a promising start.

Conflict check Our main problem is the size of the tag set: all possible combinations of tags on level 4 amount to millions of readings, and that would make the SAT-problems too big. We cannot just ignore all tags beyond level 2 or 3, because many of the rules rely on them as contexts.

As a first approximation, we have created a reduced set of 21000 readings, which allows the program to check up to 200 rules at a time before running out of memory. We are still developing better solutions, and have not run the whole grammar with this setup. Among the first rules we tested, it has found a few redundancies, such as the following:

```
SELECT ADB IF
  (0 POSTPOSIZIOAK-9)
  (-1 IZE-DET-IOR-ADJ-ELI-SIG + INE) ;
SELECT ADB IF
  (0 ("<barrena>")) (-1 INE) ;
```

The problem is that the set `POSTPOSIZIOAK-9` contains the word form "barrena", and the other set contains the tag `INE`; in other words, the latter rule is fully contained in the first rule and hence redundant.

Our second strategy is to reduce the rules themselves: from a rule such as `SELECT Verb + Sg IF (1 Noun + Sg)`, we just remove all tags higher than level 2, resulting in `SELECT Verb IF (1 Noun)`. We also keep all lexical tags intact, but unlike in Listenmaa and Claessen (2016), we allow them to attach to any morphological tags; this may lead to further false negatives, but reduces the size of the SAT-problem. This setup analyses the whole grammar, in the given order, in approximately 1 hour. With the reduced rules, the program would not find the redundancy described earlier, because the problem lies in the 3rd-level tag `INE`. But this approximation found successfully 11 duplicates or near-duplicates in the whole grammar, such as the following:

```
SELECT IZE IF   # line 817
  (0 POSTPOSIZIOAK-10IZE LINK 0 IZE_ABS_MG)
  (-1 IZE-DET-IOR-ADJ-ELI-SIG + GEN) ;
SELECT IZE IF   # line 829
  (0 POSTPOSIZIOAK-10IZE + IZE_ABS_MG)
  (-1 IZE-DET-IOR-ADJ-ELI-SIG + GEN) ;
```

Both of the contextual tests contain 3rd-level tags (`ABS, MG, GEN`), but removing them keeps the

Level 1	Level 2	Level 3	Level 4
Noun	Common noun Proper noun	Common noun, plural absolutive Common noun, singular ergative Proper noun, plural absolutive Proper noun, singular ergative	Common noun, plural absolutive, animate Common noun, plural absolutive, inanimate ... Proper noun, singular ergative, animate Proper noun, singular ergative, inanimate

Table 1: Levels of granularity

	All tags (Level 4)			**48 main categories** (Level 2)		
	Rec.	*Prec.*	*F-score*	*Rec.*	*Prec.*	*F-score*
Original grammar	**95.61**	62.99	75.94	**97.48**	84.37	90.45
ML-tuned original	93.87	68.06	78.91	96.66	86.82	91.48
Preliminary cleanup	94.81	56.56	70.85	96.82	84.13	90.03
ML-tuned prel.cl.	93.41	**68.61**	**79.11**	96.41	**87.19**	**91.57**

Table 2: Preliminary evaluation on words, excluding punctuation, for levels 4 and 2 of granularity.

sets identical, hence it is not a problem for the conflict check.

Finally, all setups have found some internal conflicts. In order to get a more reliable account, we would need more accurate tagset, beyond the 21000. To be fair, many internal conflicts can be detected by simpler means: using STRICT-TAGS would reveal illegal tags, which are the reason for a large number of internal conflicts. But some cases are due to a mistake in logic, rather than a typo; examples such as the following were easily found by the tool.

```
REMOVE ADI IF (NOT 0 ADI) (1 BAT) ;
SELECT ADI IF (0C ADI LINK 0 IZE) ;
```

The first rule is clearly an error; it is impossible to remove an ADI from a reading that does not have one. The conflict likely stems from a confusion between `NOT X` and `(*) - X`. The second rule is not obvious to the eye; the interplay of `0C` and `LINK 0` requires `ADI` and `IZE` in the same reading, which is not possible[2].

ML-tuning So far, the most important use of the ML-tuning has been to overcome the differences in ordering. Given the preliminary nature of the work, we have not tried multiple variations. We used a development corpus of 61,524 tokens and a test corpus of 65,153 tokens; the same which we used to obtain the scores in Table 2. We stopped the tuning after 5 iterations, and used an error threshold of 25 % to consider a rule as "good" or "bad".

[2]ADI is a verb, IZE is a noun.

In the future, as the grammar cleanup advances, we are interested in trying out different settings. Already in our current stage, ML-tuning has clearly improved the precision, for both original and preliminarily cleaned grammars, and for both levels of granularity; it is likely that experimenting with different parameters, we would find a combination that would also improve the recall, like Bick (2013) and Bick et al. (2015) report. However, while ML-tuning improves the grammar's performance, it makes it less readable for human eyes, and continuing the development is harder. Thus we might settle to two versions of the grammar: one for maintenance, and other for running.

5 Future work

After checking the soundness of the grammar by means of some simple tools, we are aware that in the near future we will need more complex utilities for helping the grammar writing. The following items are on our wish list:

Flexible rule ordering We would like the option to view the grammar in a variety of orders, possibly implemented as a feature in the CG IDE. The base order would be one that is easily maintainable and linguistically motivated, and any other orders can be generated from the base order.

Deeper connections between the rules So far we have used the SAT-based conflict check to run the grammar in order, but we would like to develop this further: take any given rule, and give a list of

all the rules, anywhere in the grammar, that potentially feed to it or block it. The biggest problem in developing such a method is the size of the tagset; this leads us to the next item on our wishlist.

Tagset minimisation This feature may be specific to the Basque grammar; for a language with a smaller tagset, there is no reason to restrict the number of tags used in the rules. We propose this idea, because we think it would help to make the SAT-encoding of the Basque grammar more manageable.

The grammar is written to optimize the recall and precision on level 2 tags. It is possible that some of the level 4 or 3 tags used in the rules could be removed without it affecting the functionality of the grammar. Using a development corpus, we could find the minimal set of tags that discriminate between correct and incorrect readings. The following example illustrates the idea:

```
"<lurtarraren>"
  "lurtar" ADJ ARR IZAUR+ GEN
           NUMS MUGM ZERO <Correct!>
  "lurtar" IZE ARR GEN NUMS MUGM ZERO
  "lurtar" ADJ ARR IZAUR+ ABS MG
```

For the given cohort, tags that are only in the correct are `GEN` and only in incorrect are `IZE, ABS, MG`. In other words, we learn that a rule that would target e.g. `ZERO` or `IZAUR+` would not remove all ambiguity. We can compute these tags for all cohorts/ambiguity classes, and see if some tags don't contribute to the disambiguation as much as the others. In such a case, we could simplify the rules in the grammar.

6 Conclusions

We have set out to improve the readability and performance of the Basque CG. The work is in progress, and the improvements on the performance are so far quite minor, but we feel this as a promising start, and a useful case study, for trying out the resources developed within the CG community.

Acknowledgments

This work has been supported by the project UPV/EHU taldea. UPV/EHU (GIU16/16)

References

Itziar Aduriz, José María Arriola, Xabier Artola, Arantza Diaz de Ilarraza, Koldo Gojenola, and Montse Maritxalar. 1997. Morphosyntactic disambiguation for basque based on the constraint grammar formalism. In *Proceedings of Recent Advances in NLP (RANLP97)*.

Itziar Aduriz, Maria Jess Aranzabe, Jose Maria Arriola, Aitziber Atutxa, Arantza Diaz de Ilarraza, Nerea Ezeiza, Koldo Gojenola, Maite Oronoz, Aitor Soroa, and Ruben Urizar. 2006. Methodology and steps towards the construction of EPEC, a corpus of written Basque tagged at morphological and syntactic levels for the automatic processing. In *Corpus Linguistics Around the World*, volume 56 of *Language and Computers*, pages 1–15. Rodopi, Netherlands.

Izaskun Aldezabal, Olatz Ansa, Bertol Arrieta, Xabier Artola, Aitzol Ezeiza, Gregorio Hernndez, and Mikel Lersundi. 2001. Edbl: a general lexical basis for the automatic processing of basque. In *IRCS Workshop on linguistic databases. Philadelphia (USA)*.

Eckhard Bick, Kristin Hagen, and Anders Nklestad, 2015. *Optimizing the Oslo-Bergen Tagger*, pages 11–19. Linkping University Electronic Press.

Eckhard Bick. 2013. ML-Tuned Constraint Grammars. In *Proceedings of the 27th Pacific Asia Conference on Language, Information and Computation (PACLIC 2013)*, pages 440–449.

Nerea Ezeiza, Itziar Aduriz, Iñaki Alegria, Jose Mari Arriola, and Ruben Urizar. 1998. Combining stochastic and rule-based methods for disambiguation in agglutinative languages. In *COLING-ACL'98. Pgs. 380 - 384. Vol 1. Montreal (Canada). August 10-14, 1998*.

Fred Karlsson, Atro Voutilainen, Juha Heikkilä, and Arto Anttila. 1995. *Constraint Grammar: a language-independent system for parsing unrestricted text*, volume 4. Walter de Gruyter.

Fred Karlsson. 1990. Constraint grammar as a framework for parsing running text. In *Proceedings of 13th International Conference on Computational Linguistics (COLING 1990)*, volume 3, pages 168–173, Stroudsburg, PA, USA. Association for Computational Linguistics.

Inari Listenmaa and Koen Claessen. 2016. Analysing Constraint Grammars with a SAT-solver. In *Proceedings of the 10th edition of the Language Resources and Evaluation Conference (LREC 2016)*.

Exploring the Expressivity of Constraint Grammar

Wen Kokke
University of Edinburgh
wen.kokke@ed.ac.uk

Inari Listenmaa
University of Gothenburg
inari.listenmaa@cse.gu.se

Abstract

We believe that for any formalism which has its roots in linguistics, it is a natural question to ask "how expressive is it?" Therefore, in this paper, we begin to address the question of the expressivity of CG. Aside from the obvious theoretical interest, we envision also practical benefits. Understanding what CG can and cannot express makes it possible to transform other formalisms to corresponding or approximate CGs, thus making way for new ways of grammar writing, and better reuse of existing language resources.

1 Introduction

For any formalism with its root in linguistics, it is natural to ask questions such as "How expressive is it?" or "Where does it sit in the Chomsky hierarchy?" (Chomsky, 1956) In this paper, we begin addressing some of these questions for constraint grammar (Karlsson et al., 1995, CG).

Before we can even consider such a question, there is a problem we must solve. CG was never meant to be a grammar in the generative sense. Instead, it is a tool for analysing and disambiguating strings. This, we believe, explains why the question of the expressivity of CG went unasked and unanswered for a long time. It also gives us our first problem: How do we view CGs generatively? We address this in section 2.

2 Generative Constraint Grammar

We view a constraint grammar CG as generating a formal language $\mathscr{L}$ over an alphabet Σ as follows. We encode words $w \in \Sigma^\star$ as a sequence of cohorts, each of which has one of the symbols of w as a reading. A constraint grammar CG rejects a word if, when we pass its encoding through the CG, we get back the cohort `"<REJECT>"`. A constraint grammar CG accepts a word if it does not reject it. We generate the language $\mathscr{L}$ by passing every $w \in \Sigma^\star$ through the CG, and keeping those which are accepted.

As an example, consider the language $a^\star$ over $\Sigma = \{a, b\}$. This language is encoded by the following constraint grammar:

```
LIST A = "a";
LIST B = "b";
SET LETTER = A OR B;
SELECT A;
ADDCOHORT ("<REJECT>")
    BEFORE LETTER
    IF (-1 (>>>) LINK 1* B);
REMCOHORT LETTER
```

We then encode the input words as a series of letter cohorts with readings (e.g. `"<l>" "a"`, `"<l>" "b"`), and run the grammar. For instance, if we wished to know whether either word in $\{aaa, aab\}$ is part of the language $a^\star$, we would run the following queries:

Input	Output
"<l>" "a"	"<l>" "a"
"<l>" "a"	"<l>" "a"
"<l>" "a"	"<l>" "a"
"<l>" "a"	"<REJECT>"
"<l>" "a"	
"<l>" "b"	

As CG is a tool meant for disambiguation, we can leverage its power to run both queries at once:

Input	Output
"<l>" "a"	"<l>" "a"
"<l>" "a"	"<l>" "a"
"<l>" "a" "b"	"<l>" "a"

This is a powerful feature, because it allows us disambiguate based on some formal language $\mathscr{L}$ if we can find the CG which generates it. However, the limitations of this style become apparent when we look at a run of a CG for the language $\{ab, ba\}$:

Input	Output
"<l>" "a" "b"	"<l>" "a" "b"
"<l>" "a" "b"	"<l>" "a" "b"

While the output contains the interpretations *ab* and *ba*, it also includes *aa* and *bb*. Therefore, while this style is useful for disambiguating using CGs based on formal languages, it is too limited to be used in defining the language which a CG generates.

In light of the idea of using CGs based on formal languages for disambiguating, it seems at odds with the philosophy of CG to reject by replacing the entire input with a single "<REJECT>" cohort. CG generally refuses to remove the last possible reading of a cohort, under the philosophy that *some* information is certainly better than none. However, for the definition of CG as a formal language, we need some sort of distinctive output for rejections. Hence, we arrive at *two* distinct ways to run generative CGs: the method in which we input unambiguous strings, and output "<REJECT>", which is used in the definition of CG as a formal language; and the method in which we input ambiguous strings, and simply disambiguate as far as possible.

It should be noted that VISL CG-3 (Bick and Didriksen, 2015; Didriksen, 2014) supports commands such as EXTERNAL, which runs an external executable. It should therefore be obvious that the complete set of VISL CG-3 commands, at least theoretically, can generate any recursively enumerable language. For this reason, we will investigate particular subsets of the commands permitted by CG. In sections 3 and 4, we will restrict ourselves to the subset of CG which only uses the REMOVE command with sections, and show this to at least cover all regular languages and some context-free and context-sensitive languages. In section 5, we will restrict ourselves to the subset of CG which only uses the ADDCOHORT and REMCOHORT commands with sections, and show this to be Turing complete.

3 A lower bound for CG

In this section, we will only use the REMOVE command with sections, in addition to a single use of the ADDCOHORT command to add the special cohort "<REJECT>", and a single use of the REMCOHORT command to clean up afterwards. We show that, using only these commands, CG is capable of generating some context-free and context-sensitive languages, which establishes a lower bound on the expressivity of CG (see Figure 1).

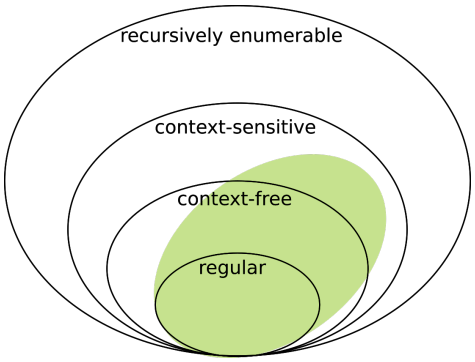

Figure 1: Lower bound on the expressivity of the subset of CG using only REMOVE.

3.1 Example grammar: $a^n b^n$

Below, we briefly describe the CG which generates the language $a^n b^n$. This CG is defined over the alphabet Σ, in addition to a hidden alphabet Σ'. These hidden symbols are meant to serve as a simple form of memory. When we encode our input words, we tag each cohort with *every* symbol in the hidden alphabet[1], e.g. for some symbol $\ell \in \Sigma$ and $\Sigma' = \{h_1, \ldots, h_n\}$ we would create the cohort "<ℓ>" "h$_1$" ... "h$_n$".

The CG for $a^n b^n$ uses the hidden alphabet {odd, even, opt_a, opt_b}. These symbols mean that the cohort they are attached to is in an even or odd position, and that a or b is a legal option for this cohort, respectively. The CG operates as follows:

1. Is the number of characters even? We know the first cohort is odd, and the rest is handled with rules of the form REMOVE even IF (NOT -1 odd). If the last cohort is odd, then discard the sentence. Otherwise continue...

2. The first cohort is certainly a and last is certainly b, so we can disambiguate the edges: REMOVE opt_b IF (NOT -1 (*)), and REMOVE opt_a IF (NOT 1 (*)).

3. Disambiguate the second cohort as a and second-to-last as b, the third as a and third-to-last as b, etc, until the two ends meet in the middle. If every "<a>" is marked with opt_a, and every "<b>" with opt_b, we accept. Otherwise, we reject.

[1] We can automatically add these hidden symbols to our cohorts using a single application of the ADD command.

The language $a^n b^n$ is context-free, and therefore CG must at least partly overlap with the context-free languages.

3.2 Example grammar: $a^n b^n c^n$

We can extend the approach used in the previous grammar to write a grammar which accepts $a^n b^n c^n$. Essentially, we can adapt the above grammar to find the middle of any input string. Once we have the middle, we can "grow" *a*s from the top and *b*s up from the middle, and *b*s down from the middle and *c*s up from the bottom, until we divide the input into three even chunks. If this ends with all `"<a>"`s marked with `opt_a`, all `"<b>"`s marked with `opt_b`, and all `"<c>"`s marked with `opt_c`, we accept. Otherwise, we reject.

The language $a^n b^n c^n$ is context-sensitive, and therefore CG must at least partly overlap with the context-sensitive languages.

4 Are all regular languages in CG?

In the present section, we propose a method to transform any finite-state automata into CG. The translation is implemented in Haskell, and can be found on GitHub[2].

4.1 Finite-state automata

Formally, a finite-state automaton is a 5-tuple

$$\langle \Sigma, S, s_0, \delta, F \rangle.$$

Σ is the alphabet of the automaton, S is a set of states, including a starting state s_0 and a set F of final states. δ is a transition function, which takes one state and one symbol from the alphabet, and returns the state(s) where we can get from the original state with that symbol. The automaton in Figure 2 is presented as follows:

$$\begin{aligned} S &= \{\texttt{s1}, \texttt{s2}\} & \Sigma &= \{\textit{det, adj, n}\} \\ s_0 &= \texttt{s1} & \delta &= \{\texttt{s1} \xrightarrow{\textit{det}} \{\texttt{s2}\}, \\ F &= \{\texttt{s1}\} & & \quad \texttt{s2} \xrightarrow{\textit{adj}} \{\texttt{s2}\}, \\ & & & \quad \texttt{s2} \xrightarrow{\textit{noun}} \{\texttt{s1}\}\} \end{aligned}$$

Informally, the automaton describes a simple set of possible noun phrases: there must be one determiner, one noun, and 0 or more adjectives in between. We implement a corresponding CG in the following sections.

[2] See `https://github.com/inariksit/cgexp`

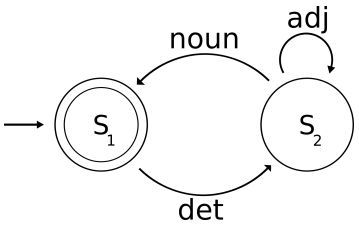

Figure 2: A finite-state automaton describing the regular language `det (adj)* noun`.

4.2 Cohorts and sentences

We encode our input as a sequence of *state cohorts* and *transition cohorts*. Initially, a state cohort contains the full set $S = \{\texttt{s1}, \texttt{s2}\}$ as its readings, and a transition cohort contains the alphabet $\Sigma = \{\textit{det, adj, noun}\}$, or some subset of it. As an example, we generate all 2-letter words recognised by the automaton in Figure 2. The initial maximally ambiguous input for length 2 looks as follows:

```
"<s>" "<w>" "<s>" "<w>" "<s>"
  s1    det   s1    det   s1
  s2    adj   s2    adj   s2
        noun        noun
```

The grammar disambiguates both transition cohorts and state cohorts. Thus the desired result shows both the accepted sequence(s)—*det noun* in this case—and their path(s) in the automaton.

```
"<s>" "<w>" "<s>" "<w>" "<s>"
  s1    det   s2   noun   s1
```

We can easily adapt the disambiguation scheme for real-world ambiguities, such as "the present". The state cohorts are identical, but the transition cohorts contain now some actual word form, and the initial ambiguity is not over the whole Σ, but some subset of it.

```
"<s>" "<the>" "<s>" "<present>" "<s>"
  s1    det     s1      adj       s1
  s2            s2      noun      s2
```

The disambiguation process goes exactly like in the first version, with full Σ in the transition cohorts. Depending on how much the initial input contains ambiguity, the result may be the same, or more disambiguated. For our example, the output is identical.

```
"<s>" "<the>" "<s>" "<present>" "<s>"
  s1    det     s2      noun      s1
```

4.3 Rules

Given that every transition happens between two states, and every state has an incoming and out-

going transition, every rule needs only positions -1 and 1 in its contextual tests. The semantics of the rules are "remove a transition, if it is *not* surrounded by allowed states", and "remove a state, if it is *not* surrounded by allowed transitions". For the example automaton, the rules are as follows:

```
REMOVE Det          # Transition rules
    IF (NEGATE -1 S1 LINK 2 S2) ;
REMOVE Adj
    IF (NEGATE -1 S2 LINK 2 S2) ;
REMOVE Noun
    IF (NEGATE -1 S2 LINK 2 S1) ;

REMOVE S1                # State rules
    IF (NEGATE -1 >>> OR Noun
           LINK 2 Det) ;
REMOVE S2
    IF (NEGATE -1 Det OR Adj
           LINK 2 Adj OR Noun) ;
```

The start and end states naturally correspond to the first and last state cohort, and can be trivially disambiguated, in this case both into `s1`. Once we remove a reading from either side of a cohort, some more rules can take action—the context "`s2` on the left side and `s1` on the right side" may be broken by removing either `s2` or `s1`. One by one, these rules disambiguate the input, removing impossible states and transitions from the cohorts.

4.4 Result

For the final result of the disambiguation, we consider three options: the cohorts may contain the whole alphabet, a well-formed subset or a malformed subset.

Full Σ If there is only one allowed word of length n in the language, then the result will contain only fully disambiguated transition cohorts. Furthermore, if there is only path in the automaton that leads to this word, then also the state cohorts are fully disambiguated.

If there are multiple words of the same length in the language, then we have to relax our criteria: every transition cohort and state cohort in the result may contain multiple readings, but all of them must contribute to some valid word of length n, and its path in the automaton.

Well-formed subset of Σ With well-formed subset, we mean that each cohort contains at least one of the correct readings: {`det`} for "the", and {`adj`,`noun`} for "present". If the initial input is well-formed, then the result will be correct, and may even be disambiguated further than with the full Σ in the transition cohorts.

Malformed subset of Σ Malformed subset has at least one cohort without any correct readings, for example, "the" is missing a `det` reading. This will lead to arbitrary disambiguations, which do not correspond to the automaton. Without a `det` reading in "the", the rule which removes `s2` would trigger in the middle state, leaving us with three `s1` states. `s1-s1-s1` is an impossible path in the automaton, so it would trigger all of the transition rules, and stop only when there is one, arbitrary, reading left in the transition cohorts.

5 Turing Machines in CG?

In the previous sections, we have assumed that CG refers to the subset of VISL CG-3 which uses only the `REMOVE` command. In this section, we will take CG to refer to the subset of VISL CG-3 which uses only the `ADDCOHORT` and `REMCOHORT` commands, and show that this subset is Turing complete. We will do this by implementing a procedure which translates arbitrary Turing machines to CG, taking VISL CG-3 itself as sufficient evidence of the fact that Turing machines can simulate constraint grammars.

The translation we present in this section has been implemented in Haskell, and can be found on GitHub[3]

5.1 A sample Turing machine

We will discuss our translation by means of an example Turing machine. Before we delve into this, however, we will briefly remind the reader of the definition of a Turing machine. A Turing machine is a 7-tuple

$$M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle.$$

Q is a finite, non-empty set of states, with a designated starting state $q_0 \in Q$, and a subset $F \subseteq Q$ of accepting states. Γ is a set of tape symbols, with a designated blank symbol b and a subset $\Sigma \subseteq \Gamma \setminus \{b\}$ of input symbols. Lastly, δ is a transition function of the type

$$(Q \setminus F) \times \Gamma \to Q \times \Gamma \times \{\text{Left}, \text{Right}\}.$$

For the remainder of this section, we will use the Turing machine which computes the successors of

[3] See `https://github.com/wenkokke/cgtm`.

binary numbers as an example. This machine is given as follows:

$$Q = \{\texttt{S0}, \texttt{S1}, \texttt{S2}, \text{Halt}\} \quad \Sigma = \{0, 1\}$$
$$\Gamma = \{\_, 0, 1\} \quad q_0 = \texttt{S0}$$
$$b = \_ \quad F = \{\text{Halt}\}$$

The transition function δ is described in table 1. What do these various states do? `S0` and `S2` both move the head of the Turing machine to the start of the number. This leaves `S1` for the actual computation. While in state `S1`, the head will move rightwards, overwriting any `1` it encounters with a `0`, until it reaches either a `0` or the end of the number. It then overwrites this final symbol with a `1`. Table 2 shows the execution trace of our sample Turing machine for the input `1101`, writing the current state *before* the current position of the head.

5.2 Representing the tape and state

We will represent the tape of the Turing machine using the sequence of cell cohorts (written `"<c>"`):

```
"<c>" "<c>" "<c>" "<c>"
 "1"   "1"   "0"   "1"
```

We will store the current state in a special cohort (written `"<s>"`) which we insert right before the cell the Turing machine is currently reading. This means that, e.g. the middle row in table 2 is represented by the following cohorts:

```
"<c>" "<c>" "<c>" "<c>" "<s>" "<c>" "<c>"
 "_"   "_"   "0"   "0"  "S1"  "0"   "1"
```

5.3 Simulating the Turing machine

We start the Turing machine by inserting a cohort with the starting state at the beginning of our input. The starting state for our sample machine is `S0`, so we add the following code to our CG:

```
BEFORE-SECTIONS
ADDCOHORT ("<s>" "S0")
   BEFORE ("<c>") IF (-1 (>>>));
```

Now for the main portion of the Turing machine—simulating the transition function. Since this function is applied iteratively, we will wrap our code in a `SECTION`. We need some way to simulate an infinite tape. Therefore, the first thing we do in each section is check if the current head is near the edge of the tape. If it is, we simply add a new, blank cell:

```
ADDCOHORT ("<c>" "_")
   BEFORE ("<s>")
      IF (-1 (>>>));
ADDCOHORT ("<c>" "_")
    AFTER ("<c>")
      IF (0 (<<<) LINK -1 ("<s>"));
```

We also need some way to distinguish input from output, so before we apply our transition rules, we mark the old state and the old input symbol with the tag `"OLD"`:

```
ADD ("<s>" "OLD") ("<s>");
ADD ("<c>" "OLD") ("<c>")
 IF (-1 ("<s>" "OLD"));
```

We are using an `ADD` command here for clarity, though it is possible to encode this usage of `ADD` using `ADDCOHORT` by simply inserting a specialized cohort (e.g. `"<old>"`) after the cohort we wish to mark, and adjusting all indices and ranges accordingly.

Next, we encode our transition rules. We will translate every entry in our transition function to a pair of rules. The first of these inserts the new state, and the second of these inserts a *new* cell, with whatever we wish to write, after the old cell. For instance, the sixth rule in table 1, which says that "if we are in state 1, and we read a 1, then we write a 0, move the tape to the right, and continue in state 1," is compiled to the following two rules:

```
ADDCOHORT ("<s>" "S1")
   BEFORE ("<c>")
      IF (-2 ("<s>" "S1" "OLD") LINK
           1 ("<c>" "1" "OLD"));
ADDCOHORT ("<c>" "0")
    AFTER ("<c>" "1" "OLD")
      IF (-1 ("<s>" "S1" "OLD"));
```

Note that the first of these rules is in effect responsible for moving the head over the tape. Because of this, a rule for left movement will look slightly different. For instance, the rule which says that "if we are in state 1, and we read a blank, then we write a 1, move the tape to the left, and change to state 2" is compiled to the following two rules:

```
ADDCOHORT ("<s>" "S2")
   BEFORE ("<c>")
      IF (1 ("<s>" "S1" "OLD") LINK
          1 ("<c>" "_" "OLD"));
ADDCOHORT ("<c>" "1")
    AFTER ("<c>" "_" "OLD")
      IF (-1 ("<s>" "S1" "OLD"));
```

We run such a pair of rules for each element in the transition function, and then we finish the SECTION by removing the old state and input cell:

```
REMCOHORT ("<s>" "OLD");
REMCOHORT ("<c>" "OLD");
```

If we wish to know where the head of the machine was located when the program terminated, we can alter these lines to remove any state *except* for the halting states. However, in this instance, we will opt instead to truncate the tape after the execution finishes, by removing any leading or trailing blank cells:

```
AFTER-SECTIONS
REMCOHORT ("<c>" "_") IF (NOT -1* SYM);
REMCOHORT ("<c>" "_") IF (NOT  1* SYM);
```

6 Linear-Bounded Automata in CG?

Linear-bounded automata (LBA) are important, because they accept exactly the class of context-sensitive languages. They are defined as Turing machines whose tape is restricted to the portion containing the input. It therefore seems obvious that we can simulate an LBA by removing the two rules which expand the tape from the transformation outlined in section 5. However, this is not incredibly interesting, as we already know the subset of CG using only ADDCOHORT and REMCOHORT is Turing complete. In this section, we will discuss a different subset of CG which we believe to be sufficiently expressive to cover all context-sensitive grammars. This is the subsets using only ADD and REPLACE.

6.1 LBAs using ADD and REPLACE

In the encoding for Turing machines in section 5 we use ADDCOHORT, as it is the most obvious way to simulate an infinite tape. However, for LBAs, we no longer need an infinite tape. We can require the machine to do all its work with the limited number of cohorts it has been given as its input. This means we can do all the computation by adding and removing tags. We can retain much of the structure we set up for simulating Turing machines:

1. we start by marking the cohort we are currently reading—i.e. the only cohort with a state tag—with "OLD"; then
2. we ADD the next state tag to the cohort to which we are moving; then
3. we REPLACE all tags on the cohort which we left with the output symbol.

And we repeat the above steps until we reach a halting state. This way, we can implement any linear-bounded automaton as a constraint grammar using only ADD and REPLACE.

We can take this idea one step further by replacing any usage of ADD with a usage of REPLACE. We can do this, because LBAs use a finite set of states and a finite alphabet. For instance, we can mark the cohort we are currently reading as "OLD" using a series of REPLACE rules,

$$\forall q \in Q, \forall a \in \Gamma,$$
$$\texttt{REPLACE ("<c>" "}q\texttt{" "}a\texttt{" "OLD")}$$
$$\texttt{("<c>" "}q\texttt{" "}a\texttt{");}$$

Similarly for tagging the next state. However, this does result in a huge blowup in the number of rules, as instead of writing a single rule for each of these uses of ADD, we now write $|Q| \cdot |\Gamma|$ rules, to test every single combination of state and symbol.

Note that we can set up a similar construction using only the commands APPEND and REMOVE, by using readings instead of tags.

7 Discussion

We have shown several different constructions, using different subsets of CG. The resulting grammars are not very readable: they include extra cohorts and symbols, and the logic is spread across rules in a rather obscure way—in contrast to a human-written grammar, where each rule is a self-contained piece of truth about a language. Therefore we do not envision the generated grammars being used as is, but rather as compilation targets. Such CGs could be used as a part of a larger constraint grammar: some sections can be written manually, and others derived from existing grammars. This could serve as an alternative to learning grammars from a corpus. So far we only have a working conversion tool for finite-state automata, but we are hoping to develop this further, to also include context-free or even mildly context-sensitive grammars.

Another question is, even if we had a working conversion system for CFGs, would the result be correct? As Lager and Nivre (2001) point out, CG has no way of expressing disjunction. Unlike its close cousin FSIG (Koskenniemi, 1990), which

would represent a language such as $\{ab, ba\}$ faithfully, CG substitutes uncertainty on the sentence level ("either *ab* or *ba*") with uncertainty in the cohorts: "the first character may be either *a* or *b*, and the second character may be either *a* or *b*". If we use such a CG to generate, by feeding it maximally ambiguous cohorts, the result will be overly permissive. We acknowledge that this is a limitation in the expressive power: many languages can only be approximated by CG, not reproduced exactly. Nevertheless, this limitation may not matter so much when disambiguating real-world text, because the cohorts are initially less ambiguous, and leaving genuine ambiguity intact is desired behaviour for CG.

8 Related Work

Tapanainen (1999) gives an account of the expressivity of the contextual tests for 4 different constraint formalisms, including CG. In addition, parsing complexity can be easily defined for a given variant and implementation of CG; see for instance Nemeskey et al. (2014). Yli-Jyrä (2017) relates CG to early formal language theory, and provides an independent proof of non-monotonic[4] CG being Turing-complete.

[4] A monotonic variant of CG may only remove readings from cohorts, whereas a non-monotonic variant may add readings or cohorts.

References

Eckhard Bick and Tino Didriksen. 2015. CG-3 – Beyond Classical Constraint Grammar. In *Proceedings of the 20th Nordic Conference of Computational Linguistics (NODALIDA 2015)*.

Noam Chomsky. 1956. Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124, September.

Tino Didriksen, 2014. *Constraint Grammar Manual*. Institute of Language and Communication, University of Southern Denmark.

Fred Karlsson, Atro Voutilainen, Juha Heikkilä, and Arto Anttila. 1995. *Constraint Grammar: a language-independent system for parsing unrestricted text*, volume 4. Walter de Gruyter.

Kimmo Koskenniemi. 1990. Finite-state parsing and disambiguation. In *Proceedings of 13th International Conference on Computational Linguistics (COLING 1990)*, volume 2, pages 229–232, Stroudsburg, PA, USA. Association for Computational Linguistics.

Torbjörn Lager and Joakim Nivre. 2001. Part of speech tagging from a logical point of view. In *Logical Aspects of Computational Linguistics, 4th International Conference (LACL 2001)*, pages 212–227.

Dávid Márk Nemeskey, Francis Tyers, and Mans Hulden. 2014. Why implementation matters: Evaluation of an open-source constraint grammar parser. In *Proceedings of the 25th International Conference on Computational Linguistics (COLING 2014)*, pages 772–780, Dublin, Ireland, August.

Pasi Tapanainen. 1999. *Parsing in two frameworks: Finite-state and Functional dependency grammar*. Ph.D. thesis, University of Helsinki.

Anssi Yli-Jyrä. 2017. The Power of Constraint Grammars Revisited. In *Proceedings of the Constraint Grammar workshop at the 21th Nordic Conference of Computational Linguistics (NODALIDA 2017)*.

State In	Symbol In	Symbol Out	State Out	Move
	Read "_"	Write "_"	"S1"	Right
"S0"	Read "0"	Write "0"	"S0"	Left
	Read "1"	Write "1"	"S0"	Left
	Read "_"	Write "1"	"S2"	Left
"S1"	Read "0"	Write "1"	"S2"	Left
	Read "1"	Write "0"	"S1"	Right
	Read "_"	Write "_"	Halt	Right
"S2"	Read "0"	Write "0"	"S2"	Left
	Read "1"	Write "1"	"S2"	Left

Table 1: Sample Turing machine (binary successor function)

"<c>"	"<s>"	"<c>"	"<s>"	"<c>"	"<s>"	"<c>"	"<s>"	"<c>"	"<s>"	"<c>"
"_"		"_"	"S0"	"1"		"1"		"0"		"1"
"_"	"S0"	"_"		"1"		"1"		"0"		"1"
"_"		"_"	"S1"	"1"		"1"		"0"		"1"
"_"		"_"		"0"	"S1"	"1"		"0"		"1"
"_"		"_"		"0"		"0"	"S1"	"0"		"1"
"_"		"_"		"0"	"S2"	"0"		"1"		"1"
"_"		"_"	"S2"	"0"		"0"		"1"		"1"
"_"	"S2"	"_"		"0"		"0"		"1"		"1"
"_"		"_"	"S2"	"0"		"0"		"1"		"1"

Table 2: Execution trace of a Turing machine (see table 1) for input 1101

The Power of Constraint Grammars Revisited

Anssi Yli-Jyrä
University of Helsinki, Finland
anssi.yli-jyra@helsinki.fi

Abstract

Sequential Constraint Grammar (SCG) (Karlsson, 1990) and its extensions have lacked clear connections to formal language theory. The purpose of this article is to lay a foundation for these connections by simplifying the definition of strings processed by the grammar and by showing that Nonmonotonic SCG is undecidable and that derivations similar to the Generative Phonology exist. The current investigations propose resource bounds that restrict the generative power of SCG to a subset of context sensitive languages and present a strong finite-state condition for grammars as wholes. We show that a grammar is equivalent to a finite-state transducer if it is implemented with a Turing machine that runs in $o(n \log n)$ time. This condition opens new finite-state hypotheses and avenues for deeper analysis of SCG instances in the way inspired by Finite-State Phonology.

1 Introduction

Lindberg and Eineborg (1998), Lager and Nivre (2001) and Listenmaa (2016) have analyzed the Sequential Constraint Grammar (SCG) (Karlsson, 1990) from the logical point of view, proposing that the rules can be expressed in first-order Horn clauses, first-order predicate logic or propositional logic. However, many first-order logical formalisms are themselves quite expressive as Horn-clauses are only semi-decidable and first-order logic is undecidable, thus at least as powerful as SCG itself. Propositional logic is more restricted but does not help us to analyse the expressive power of SCGs and to prove the finite-stateness of grammars.

Instead of just reducing SCG to undecidable or otherwise powerful formalisms, we are interested in the ultimate challenge that tries to prove that a *practical* grammar is actually reducible to a strictly weaker formalism. This goal is interesting because this kind of narrowing reductions have been proven extremely valuable. For example, the proof that practical grammars in Generative Phonology are actually equivalent to finite-state transducers has turned out to be a game-changing result. In fact, the reduction gave birth to the influential field of Finite-State Phonology.

It is noteworthy that prior efforts to analyse SCG in finite-state terms have focused on the finite-state nature of individual and parallel rules (Peltonen, 2011; Hulden, 2011; Yli-Jyrä, 2011). The efforts have mostly ignored the generative power of the grammar system as a whole and that of practical grammar instances.

In this paper, we are aiming to Finite-State Syntax through reductions of practical SCGs. To set the formal framework, we have to start, however, from the total opposite: we show first that the simplified formalism for Nonmonotonic SCGs is Turing equivalent and thus similar to Generative Phonology (Chomsky and Halle, 1968; Ristad, 1990) and Transformational Grammar (Chomsky, 1965; Peters and Ritchie, 1973). This foundational result gives access to the large body of literature of bounded Turing machines and especially to Hennie machines that run in $O(n)$ time and are equivalent o finite-state machines. Then the Gap Theorem (Trakhtenbrot, 1964) gives us access to a looser bound $o(n \log n)$ whose reasonable approximations are sufficient and decidable conditions for finite-state equivalence. We present some ways in which these bounds can be related to SCG parsing.

The article is structured as follows. Section 2 describes the alphabets, the strings and the derivation steps in SCG parsing. In Section 3, these are used to show Turing equivalence of SCGs. In next two sections, simple bounds are introduced and

elaborated further to obtain specific conditions for finite-state equivalence of grammars. Further links to formal language theory and two important open problems are presented in Section 6. Then the paper is concluded.

2 SCG as a "Phonological" Grammar

In the SCG literature, morphosyntactic readings of tokens are usually represented as tag strings like `"<went>" "go" V PAST`. The tag strings are now viewed as a compressed representation for a huge binary vector $(f_0, f_1, f_2, ..., f_k, ...)$. The semantics of the grammar ignores some tags and considers only k tags declared in advance in the grammar. These k tags or features distinguish readings from each other and define the *reading alphabet* $\Sigma = 2^k$.

An ambiguous token has more than one reading associated to it. The elements of the *cohort alphabet* $\mathscr{P}(\Sigma)$ are called cohorts. This alphabet is the powerset of the reading alphabet. Only a small subset of all possible cohorts occur in practice.

The input of an SCG is produced by a deterministic finite-state function, $Lexicon^* : T^* \rightarrow (\mathscr{P}(\Sigma))^*$, that maps token strings to *lexical cohort strings* of the same length. This function is the concatenation closure of the function $Lexicon : T \rightarrow (\mathscr{P}(\Sigma))$ that maps every token to a cohort.

Since the image of each token is a set of strings, *Lexicon* is internally a nondeterministic lexical transducer (Karttunen, 1994; Chanod and Tapanainen, 1995), but the image of each token is viewed externally as a *symbol* in $\mathscr{P}(\Sigma)$, making *Lexicon* a one-valued function.

An SCG processes the lexical cohort string by iterated application of a derivation step $\Rightarrow$: $(\mathscr{P}(\Sigma))^* \rightarrow (\mathscr{P}(\Sigma))^*$ that affects one cohort at a time. The contexts conditions of each derivation step are normally defined using an existing SCG formalism for contextual tests. Monadic Second Order Logic (Büchi, 1960; Elgot, 1961; Trakhtenbrot, 1961) provides an alternative formalism that can express all finite state languages over $\mathscr{P}(\Sigma)$.

The parser defines the parsing strategy that resolves the conflicts between rules that could be applied simultaneously. A typical strategy chooses always the most reliable rule and the leftmost target position. When the plain contextual tests are combined with the application strategy, we obtain a total functional transducer (Skut et al., 2004; Yli-Jyrä, 2008; Hulden, 2009). E.g., the transducer in

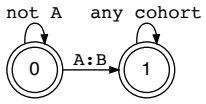

Figure 1: A simple $\Rightarrow$ relation as an FST

Fig. 1 is total and replaces A by B in the first possible occurrence position.

The semantics of an SCG grammar G is defined as the relation

$$[[G]] = \{(i,o) \mid i \in \mathscr{P}(\Sigma)^*, o \in I, i \Rightarrow^* o\}$$

where $I \subseteq \mathscr{P}(\Sigma)^*$ as $\{x \mid (x,x) \in \Rightarrow\}$. This semantics makes SCG grammars similar to grammars in Generative Phonology (Chomsky and Halle, 1968) as both grammars relate the lexical string into some kind of output string by applying a sequence of alternation rules.

3 Nonmonotonic SCG

Two recent SCG implementations (Tapanainen, 1996; Didriksen, 2017) are *nonmonotonic*: they do not always reduce the input but they can insert tags, readings and even cohorts. In this section, we study the expressive power of such SCGs.

3.1 Minimal Definition

For the sake of minimality, we define the Nonmonotonic SCG (NM-SCG) as a rule system that supports the following kinds of local transformation rules:

- `REPLACE (old) (new) (cond)`$^+$
- `INSCOHORT (targ) (cond)`$^+$
- `REMCOHORT (targ) (cond)`$^+$

The first rule template in the above replaces the leftmost cohort containing the reading `old` with a cohort that contains the reading `new` if the relative context condition `cond` is satisfied. The familiar `SELECT` and `DELETE` rules are seen as shorthands for sets of `REPLACE` rules. The second and the third rule templates are used to insert or remove a target cohort matching the pattern `targ` when the condition `cond` is satisfied. The plus ($^+$) indicates that more than one condition can be present.

Our simplified context conditions are of the form `(`d `tags)` or `(`d `NOT tags)` where the first tests the presence of the pattern `tags` in the relative cohort location d. The second is true when the location does not contain the pattern.

3.2 One-Tape Turing Machine

A one-tape deterministic *Turing machine* (TM) has a finite control unit and an infinite rewritable tape with a pointer (Fig. 2). A configuration of the machine consists of the current state q, the current pointer value and the contents of the working tape.

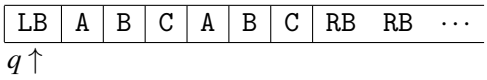

Figure 2: A one-tape Turing machine

The tape is divided into squares that hold a left boundary LB, a right boundary RB, or a symbol from the tape alphabet Ω. Given the input string $x \in \Omega^*$, the first square of the tape is pointed and the tape is initialized with the prefix LBxRB that is followed by an infinite number of right boundary symbols.

The control unit is a deterministic finite automaton where each transition $s \overset{(A,B,d)}{\rightarrow} t$ specifies the source state s, the target state t, the input symbol A, the output symbol B, and a head move $d \in \{-1,0,1\}$. On each transition, the machine overwrites the symbol A in the pointed square with the symbol B, changes its state from s to t and then moves the pointer d steps to the right. All but the leftmost square are over-writable ($B = A$ if $A =$ LB), but the machine never moves beyond the first right boundary without overwriting it with a tape symbol and never writes RB between two tape symbols.

The computation of the machine starts from state q_0. At each step, the machine takes the next transition based on the current state and the currently pointed symbol on the memory tape. The computation continues as long as the next transition is defined and then halts by reaching a state from which there is no transition on the current input. If the halting state is among the final states F, the machine accepts the input contents and relates it with the string $x' \in \Omega$ stored to the memory tape. Otherwise, the machine either gets stuck to an infinite computation or gives up, leaving some ill-formed string to the memory tape.

3.3 Reduction to Nonmonotonic SCG

Now we show that any one-tape Turing machine can be simulated with a nonmonotonic SCG.

In our simulation, each square in the initial portion of the memory tape corresponds to a cohort in the input. Each cohort is a singleton set in $\mathscr{P}(\Sigma)$ i.e. represents just one reading in Σ. Each reading is a collection of positive features from Φ. These features include the tape symbols Ω, the boundary symbols {LB, RB}, and the markers that that we need to keep track of the computation steps.

The pointed square corresponds to a cohort that contains a marker. Since SCG can change only one cohort at a time, movement of the pointer involves two temporarily marked positions and markers: the first indicates the previously pointed square and the second indicates the new pointed square. One marker represent the source state and the other represents the transition in progress.

A transition $q \overset{A,B,d}{\rightarrow} r$, RB $\notin \{A,B\}$, corresponds to a sequence of three rule applications that change one cohort at a time. Since the set of transitions, the sets of states Q and the tape alphabet Ω are finite, each step is described with a finite set of non-monotonic SCG rules:

1. Given the state marker $\mathtt{Q}_q \in \{\mathtt{Q}_s \mid s \in Q\} \subseteq \Phi$ in cohort i and no other marked cohorts, add a transition marker T-q-$A \in \Phi$ to cohort $i+d$ that previously contains a tape symbol $C \in \Omega$:

   REPLACE (C) (T-q-A C) ($-d$ Q$_q$ A)

2. Given a transition marker T-q-A in cohort $i+d$, overwrite, in cohort i, the reading containing the tape symbol A and the state marker Q$_q$ with a reading containing the tape symbol B:

   REPLACE (Q$_q$ A) (B) (d T-q-A)

3. When no state marker is present, replace the transition marker T-q-A with the marker for the target state Q$_r$ while keeping the remainder $C \in \Sigma$ in the changed cohort:

   REPLACE (T-q-A C) (Q$_r$ C) ($-d$ NOT Q$_q$)

A transition $q \overset{\mathrm{RB},A,0}{\rightarrow} r$, $A \in \Omega$, corresponds to the application of rules:

```
ADDCOHORT (T-q-RB A) (1 Q_q RB)
REPLACE (Q_q RB) (RB) (-1 T-q-RB A)
REPLACE (T-q-RB A) (Q_r A) (1 NOT Q_q RB)
```

When the previous cohort contains tape symbol $C \in \Omega$, a transition $q \overset{A,\mathrm{RB},-1}{\rightarrow} r$, where $A \in \Omega$, corresponds to the application of rules:

```
REPLACE (C) (T-q-A C) (1 Q_q A) (2 RB)
     REMCOHORT (Q_q A) (1 T-q-A)
REPLACE (T-q-A C) (Q_r C) (1 NOT Q_q A)
```

Transitions $q \overset{\texttt{RB},A,-1}{\rightarrow} r$, $q \overset{\texttt{RB},A,1}{\rightarrow} r$ and $q \overset{A,\texttt{RB},0}{\rightarrow} r$, $A \in \Omega$, reduce to a sequence of two transitions.

The SCG parser halts when the tape contents does not trigger any of these rules that simulate transitions. The simulation accepts the input if some cohort contains a marker $\texttt{Q}_q$ such that $q \in F$.

Proposition 1. *NM-SCGs can simulate TMs.*

Since NM-SCG is itself an algorithm, we have:

Proposition 2. *There is a one-tape deterministic TM that implements the NM-SCG parser.*

Proposition 3. *NM-SCGs are equivalent to TMs.*

4 Bounded Nonmonotonic SCGs

The undecidability of Nonmonotonic SCG creates a need to restrict the formalism in ways that ensure decidability. In this section, we propose two parameters that set important bounds on the resources available to grammars.

4.1 The $O(n)$ Space Bound

The *fertility* $f \in \mathbb{N} \cup \{\infty\}$ of a nonmonotonic SCG grammar is the maximum number of new cohorts that each the grammar inserts before any of the n cohorts in the original sentence (with RB). Note that fertility $f > 0$ implies nonmonotonicity.

Proposition 4. *In finite-fertility SCGs, the length ℓ of the output string is linearly bounded.*

The bounded length of the cohort string is an important restriction to Nonmonotonic SCGs because it ensures that any infinite loop in the computation can be detected after a bounded number of computation steps because the number of distinct tape contents is bounded.

Proposition 5. *The termination of a finite-fertility SCG is decidable.*

We also know that the preconditions of each rule can tested with a finite automaton and that the actual effect on the target cohort is a functional finite-state computation that can be implemented in linear space according to the length of the cohort string.

Proposition 6. *The space requirement of a finite-fertility SCG is linear to the maximum length of the cohort string during the derivation.*

A deterministic linear-bounded automaton (DLBA) (Myhill, 1960) is a special case of Turing machines with the restriction that the right boundary is fixed and cannot be overwritten. The LBA computations can be initialized so that the space available for storing the cohort string is linearly bounded by the length of the initial cohort string.

Proposition 7. *A nonmonotonic SCG with finite fertility is simulated by an DLBA.*

The power of DLBAs is restricted to a strict subset of context-sensitive languages (Kuroda, 1964).

Proposition 8. *The cohort language accepted by a finite-fertility SCG is context sensitive.*

4.2 The $O(n^2)$ Time Bound

By studying only monotonic SCGs with the reading count r in cohorts, and the sentence length n (including RB), Tapanainen (1999) has given a lower bound for the parsing time:

Proposition 9 (Tapanainen 1999)**.** *Any monotonic SCG performs $O(nr)$ rule applications.*

The *volume* $v \in \{1, 2, ...\} \cup \{\infty\}$ of cohorts is a parameter that tells the maximum number of operations that can be applied to any cohort. This new notion is a nonmonotonic generalization of the maximum number of readings in one cohort. Finite volume basically turns every finite fertility SCG into a monotonic SCG.

Finite fertility helps us to generalize the above proposition to nonmonotonic SCGs.

Proposition 10. *Any NM-SCG performs $O((1 + f)nv)$ rule applications.*

Assuming again that any rule of the grammar can be applied in linear time according to the number of cohorts, we obtain a time complexity result:

Proposition 11. *Any NM-SCG runs in $O((1 + f)^2 n^2 v)$ time.*

5 Finite-State Hypotheses

A deterministic linear bounded automaton is a special case of one-tape deterministic Turing machines that gives us a context where many interesting conditions for finite-stateness aka regularity become applicable.

5.1 The $o(n \log n)$ Time Bound

Hennie (1965) showed that a deterministic one-tape TM running in $O(n)$ is equivalent to a finite automaton. By defining the relation between the initial and final tape contents, we can extend Hennie's result to regular relations:

Proposition 12 (Hennie 1965)**.** *A one-tape deterministic TM running in $O(n)$ time is equivalent to a functional finite-state transducer.*

The Borodin-Trakhtenbrot Gap Theorem (Trakhtenbrot, 1964) states that expanded resources do not always expand the set of computable functions. In other words, it is possible that $O(n)$ is unnecessarily tight time bound for finite-state equivalence. A less tight time bound is now expressed with the little-o notation: $t(n) \in o(f(n))$ means that the upper bound $f(n)$ grows much faster than the running time $t(n)$ when n tends to infinity: $\lim_{n\to\infty} t(n)/f(n) = 0$.

Hartmanis (1968) and Trakhtenbrot (1964) showed independently that the time resource of a finite-state equivalent deterministic one-tape TM can be expanded from $O(n)$ to $o(n\log n)$ without expanding the characterized languages. More recently, Tadaki et al. (2010) showed that the bound $o(n\log n)$ applies also to nondeterministic one-tape TMs that explore all accepting computations.

Proposition 13 (Tadaki et al. 2010)**.** *A one-tape TM running in $o(n\log n)$ time is equivalent to a finite automaton/transducer.*

A sufficient condition for finite-state equivalence of a TM is satisfied if the running time of the machine is bounded by a function $t(n)$ that is in $o(n\log n)$. For any reasonable function $t(n)$, this sufficient condition is decidable (Gajser, 2015). However, to decide finite-state equivalence of any TM, it would be necessary to consider all functions $t(n) \in o(n\log n)$.

We will assume a one-tape TM implementation for finite-fertility SCGs. The tape is initialized in such a way that f empty squares are reserved for latent cohorts at every cohort boundary.

We assume the representation of the grammar rules and the related application strategy by a functional transducer such as in Figure 1. Its optimization via the inward deterministic *bimachine* constructions (Yli-Jyrä, 2011; Hulden, 2011) optimizes the tape moves between derivation steps.

The parallel testing of all context conditions involves (i) the initialization step and (ii) a number of maintenance steps. The initialization step computes the validity of all context conditions at every tape squares in amortised $O(n)$ time. After this, the total amortised time needed to maintain the contexts is then bounded by the total number of moves needed to perform the subsequent rule applications.

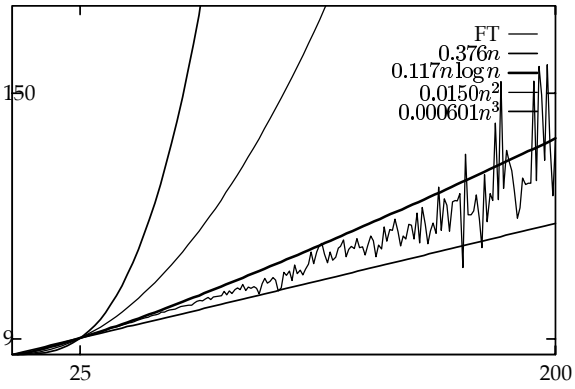

Figure 3: Average running time of CG-2 in Financial Times (according to Tapanainen 1999) seems to follow the curve $O(n\log n)$

Proposition 14. *The time used to maintain the context conditions is dominated by the time used to move between target cohorts.*

NM-SCGs based on a one-tape TM have now a regularity condition:

Proposition 15. *An NM-SCG is equivalent to a finite automaton/transducer if its one-tape TM implementation runs in $o(n\log n)$ time.*

This proposition can be compared to an interesting empirical observation by Tapanainen (1999) who reports experiments with a practical SCG (CG-2) system. According to the experiments, the average running time of the system follows closely the $O(n\log n)$ curve (Fig. 3).

On the basis of the experiments by Tapanainen, we cannot exclude the hypothesis that the asymptotic running time is actually in $o(n\log n)$. Whether the used grammar is actually equivalent to a finite-state transducer is not known.

If the given NM-SCG instance would be equivalent to a finite-state transducer, there would be a possibility to carry out monotonic SCG parsing in *linear time* and thus improve the parser's efficiency considerably. In case that the transducer is extremely large, the improvement remains solely as a theoretical possibility but the discovered regularity may still give valuable insight.

5.2 The $O(n)$ Time Bound

Hennie's finite-stateness condition (Hennie, 1965) for deterministic one-tape TMs and its generalization to nondeterministic one-tape TMs (Tadaki et al., 2010) are insightful and provide a method to construct the equivalent finite-state transducer when the finite-stateness condition is met.

LB	W	O	R	D	I	N	G	RB	⋯
q_0	q_1	q_2	$\boxed{q_3}$	q_4					
	q_7	$\boxed{q_6}$	q_5	$\hookleftarrow$					
	$\hookrightarrow$	q_8	$\boxed{q_9}$	q_{10}	q_{11}	$\cdots$			

Figure 4: A crossing sequence between squares

A *Hennie machine* refers to a one-tape TM whose running time is $O(n)$. Hennie analysed the expressive power of such machines using the concept of *crossing sequence*, aka *schema* (Rabin, 1963; Trakhtenbrot, 1964). This concept is a powerful tool in the analysis of the behaviour of two-way automata and one-tape TMs.

A *crossing sequence* is the sequence of target states $s_1, s_2, \ldots$ visited by a TM when its pointer crosses the boundary between a pair of adjacent tape squares. States $s_1, s_3, \ldots$ are reached when the pointer moves forward and states $s_2, s_4, \ldots$ are reached when pointer moves backwards. Figure 4 shows how states are visited during a computation. The crossing sequence between the 3rd and the 4th squares is $(s_1, s_2, s_3) = (q_3, q_6, q_9)$.

Every Hennie machine satisfies the property that the length of its crossing sequences is bounded by an integer $k \in \mathbb{N}$. The finiteness of the crossing sequences of a given TM is undecidable (Průša, 2014) but if a finite upper bound k exists, this constant is computable (Kobayashi, 1985; Tadaki et al., 2010).

Finiteness of crossing sequences implies that the TM is equivalent to a finite-state automaton/transducer. Furthermore, the bound lets us construct this finite-state device. Unfortunately, the size complexity of the constructed machine is large in comparison to the original TM:

Proposition 16 (Průša 2014)**.** *Each $|Q|$-state, $|\Omega|$-symbol deterministic Hennie machine can be simulated by a nondeterministic finite automaton with $2^{O(|\Omega| \log |Q|)}$ states.*

Testing the finite-stateness of already constructed TMs requires more effort than to design and construct machines that are immediately known to be Hennie machines. We will now mention a few immediate constructions.

Průša (2014)'s construction is based on a finite weight $w \in \mathbb{N}$ of the tape squares. Every time when a square is visited or passed, the weight associated with the square is reduced. Once the weight is zero, further visits to the square are blocked.

Proposition 17 (Průša 2014)**.** *A weight-reducing one-tape TM is a Hennie machine.*

Analogously, we can define an NM-SCG whose cohorts has a weight w that is reduced whenever the pointer of the associated TM implementation visits the corresponding square. The cohorts of such an NM-SCG have obviously a finite volume $v \leq w$ and can be changed at most w times.

Proposition 18. *A finite-fertility NM-SCG implemented by a weight-reducing one-tape TM is equivalent to a finite-state transducer.*

The second way to construct a Hennie-machine based NM-SCG is to set the maximum distance $m \in \mathbb{N} \cup \{\infty\}$ between adjacent rule applications.[1] When combined with the linear bound for rule applications, we obtain the $O(n)$ bound and finite-state equivalence:

Proposition 19. *A finite-fertility NM-SCG runs in $O(m(f+1)vn)$ time and is equivalent to a finite-state transducer if $m, f, v \in \mathbb{N}$.*

The third way is to assume fertility $f \in \mathbb{N}$ and $w = 1$. Since no square can be revisited, this forces the SCG to move constantly into one direction after all rule applications. This special case resembles the rewriting rules in finite-state phonology whose fundamental theorem (Johnson, 1972; Kaplan and Kay, 1994) states that if a phonological rule does not reapply to its own output (but instead moves on), it is regular.

The fourth way to construct a Hennie machine from an SCG is based on the number of times the context conditions for a cohort has to be updated. A monotonic SCG reduces the ambiguity of the sentence at every rule application. The reduced ambiguity causes occasional updates in context conditions of cohorts. Depending on the context conditions, such updates at a cohort boundary may have an infinite or finite bound. Due to functionality and inward determinism of the $\Rightarrow$-transducer, the pointer moves from one cohort to another only if the context conditions of the latter have changed as a result of a rule application. Thus, the number of context updates bound the number of moves:

Proposition 20. *If the context conditions can be updated only finitely often at every cohort, then the SCG is equivalent to a finite-state transducer.*

[1] This approach was pursued and developed further by the current author in an earlier manuscript (Yli-Jyrä, unpublished) that is available on request.

6 Open Problems

6.1 Aperiodic Context Conditions

Yli-Jyrä (2003) showed that the context conditions used in a realistic Finite-State Intersection Grammar (FSIG) are not only regular but star-free. Since context conditions of SCG rules are strictly weaker than those of FSIG (Tapanainen, 1999), we have a strong conjecture that contexts in practical SCG are also star-free.

Star-free languages are definable in the monadic first-order logic of order, FO$[<]$, a decidable logic that is equivalent to LTL (Pnueli, 1977) and loop-free alternating finite automata (LF-AFA) (Salomaa and Yu, 2000). The states in an LF-AFA are totally ordered in such a way that every state is independent from all the preceding states in this order. This is a major restriction to the structure and expressive power of alternating finite automata.

While preserving possible star-freeness has led improvements in fundamental algorithms (Yli-Jyrä and Koskenniemi, 2004), we have not been able to solve the following open problem:

Open Problem 1. *Determine whether the construction of Hennie machines could benefit from star-freeness of the context conditions, possibly in combination with other conditions.*

6.2 Full Parsing

Reductionistic parsing (Koskenniemi, 1990; Maruyama, 1990; Voutilainen and Tapanainen, 1993; Gross, 1997; Eisner and Smith, 2005) is closely related to the consistency enforcing methods used in image recognition (Huffman, 1971; Clowes, 1971) and to the satisfiability in logic (Listenmaa, 2016). All these methods use some idea of domains that are then constrained.

Karlsson (1990) introduced the term *cohort* for ambiguity domains or lists of readings associated with tokens. Lauri Karttunen has then proposed (p.c., see also Voutilainen 1994) that the cohorts can be treated as strings and processed by finite-state transducers. This idea has been implemented later by others (Peltonen, 2011; Hulden, 2011).

Interestingly, the idea of processing ambiguity domains, i.e. cohorts, as strings is actually older than the SCG tradition. In the context of formal language theory, it dates back to Greibach (1973) and has been appreciated recently, e.g. by Okhotin (2013). What is interesting in Greibach's original use of cohorts is that these cohorts are used to represent parse trees instead of just morphological ambiguity. The decomposition of trees and digraphs into local trees in the lexicon is actually due to the tradition of Categorial Grammar (Ajdukiewicz, 1935; Bar-Hillel, 1953; Lambek, 1958). This suggests an avenue for future SCG-related research.

Open Problem 2. *Develop an SCG grammar that performs full parsing on the basis of the structural ambiguity encoded into lexical categories.*

7 Conclusions

In this paper, the author has laid foundations for the analysis of the generative power of SCGs.

- The parsing is viewed as a derivation that resembles that of Generative Phonology.
- The equivalence between Nonmonotonic SCG and Turing machines is established, thus linking Constraint Grammar to Undecidability and the Chomsky hierarchy.
- Finite-fertility SCGs are shown to be context sensitive and running in quadratic time.
- A loose time bound $o(n \log n)$ for finite-state equivalent SCG instances (running on a TM) is provided and related to prior experiments.
- Specific conditions for constructing finite-state equivalent SCGs are given.
- Two open problems related to the potential of the star-freeness restriction of context conditions and the structural categories in the lexicon are presented.

The current work has demonstrated that the SCG formalism is not just a programming language for text linguistics but a formal framework that lends itself to connections to the richness of formal language theory and rigorous formal analysis of the related parsing complexities, culminating to attempts to reduce grammars into finite transducers.

Acknowledgements

The author has received funding as Research Fellow from the Academy of Finland (dec. No 270354 - A Usable Finite-State Model for Adequate Syntactic Complexity) and Clare Hall Fellow from the University of Helsinki (dec. RP 137/2013). The distance-based restriction of SCG has been studied by the author (Yli-Jyrä, unpublished) under earlier funding from the first agency (dec. 128536).

References

Kazimierz Ajdukiewicz. 1935. Die syntaktische konnexität. In Storrs McCall, editor, *Polish Logic 1920-1939*, page 207231. Oxford University Press, Oxford. Translated from Studia Philosophica, 1, 1-27.

Yehoshua Bar-Hillel. 1953. A quasi-arithmetical notation for syntactic description. *Language*, 29:4758.

J. R. Büchi. 1960. Weak second-order arithmetic and finite automata. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 6:66–92.

Jean-Pierre Chanod and Pasi Tapanainen. 1995. A lexical interface for finite-state syntax. MLTT technical report, Rank Xerox Research Centre, Grenoble Laboratory, Grenoble, France, February 9.

Noam Chomsky and Morris Halle. 1968. *The Sound Pattern of English*. Harper & Row, New York.

Noam Chomsky. 1965. *Aspects of the Theory of Syntax*. MIT Press, Cambridge, Massachusetts.

M. B. Clowes. 1971. On seeing things. *Artificiul Intelligence*, 2:79–116.

Tino Didriksen, 2017. *Constraint Grammar Manual: 3rd version of the CG formalism variant*. GrammarSoft ApS, Denmark.

Jason Eisner and Noah A. Smith. 2005. Parsing with soft and hard constraints on dependency length. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 30–41, Vancouver, British Columbia, October. Association for Computational Linguistics.

Calvin C. Elgot. 1961. Decision problems of finite automata design and related arithmetics. *Transactions of the American Mathematical Society*, 98(1):21–51.

David Gajser. 2015. *Verifying Time Complexity of Turing Machines*. Ph.D. thesis, University of Ljubljana, Department of Mathematics, Ljubljana, Slovenia.

Sheila Greibach. 1973. The hardest context-free language. *SIAM Journal on Computing*, 2(4):304–310.

Maurice Gross. 1997. The construction of local grammars. In Emmanuel Roche and Yves Schabes, editors, *Finite-State Language Processing*, chapter 11, pages 329–354. A Bradford Book, the MIT Press, Cambridge, MA, USA.

Juri Hartmanis. 1968. Computational complexity of one-tape Turing machine computations. *J. ACM*, 15(2):325–339, April.

Frederick C. Hennie. 1965. One-tape, off-line Turing machine computations. *Information and Control*, 8(6):553–578.

D. A. Huffman. 1971. Impossible objects as nonsense sentences. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 6, pages 295–323. Edinburgh University Press, Edinburgh, Scotland.

Måns Hulden. 2009. *Finite-State Machine Construction Methods and Algorithms for Phonology and Morphology*. Ph.D. thesis, Department of Linguistics, The University of Arizona.

Mans Hulden. 2011. Constraint Grammar parsing with left and right sequential finite transducers. In *Proceedings of the 9th International Workshop on Finite State Methods and Natural Language Processing (FSMNLP 2011)*, pages 39–47, Blois, France, July. Association for Computational Linguistics.

C. Douglas Johnson. 1972. *Formal Aspects of Phonological Description*. Number 3 in Monographs on linguistic analysis. Mouton, The Hague.

Ronald M. Kaplan and Martin Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378, September.

Fred Karlsson. 1990. Constraint Grammar as a framework for parsing unrestricted text. In H. Karlgren, editor, *Proceedings of the 13th International Conference of Computational Linguistics*, volume 3, pages 168–173, Helsinki.

Lauri Karttunen. 1994. Constructing lexical transducers. In *15th COLING 1994, Proceedings of the Conference*, volume 1, pages 406–411, Kyoto, Japan.

K. Kobayashi. 1985. On the structure of one-tape nondeterministic Turing machine time hierarchy. *Theoretical Computer Science*, 40(2–3):175–193.

Kimmo Koskenniemi. 1990. Finite-state parsing and disambiguation. In Hans Karlgren, editor, *13th COLING 1990, Proceedings of the Conference*, volume 2, pages 229–232, Helsinki, Finland, August.

Sige-Yuki Kuroda. 1964. Classes of languages and linear-bounded automata. *Information and Control*, 7(2):207–223.

Torbjörn Lager and Joakim Nivre. 2001. Part of speech tagging from a logical point of view. In P. de Groote, G. Morrill, and C. Retoré, editors, *Logical Aspects of Computational Linguistics*, volume 2099 of *Lecture Notes in Artificial Intelligence*, pages 212–227. Springer-Verlag.

Joachim Lambek. 1958. The mathematics of sentence structure. *American Mathematical Monthly*, 65:154–170.

Nikolaj Lindberg and Martin Eineborg. 1998. Learning Constraint Grammar-style disambiguation rules using Inductive Logic Programming. In *36th ACL 1998, 17th COLING 1998, Proceedings of the Conference*, Montréal, Quebec, Canada, August 10-14.

Inari Listenmaa. 2016. *Analysing Constraint Grammar with SAT*. Licentiate thesis, Chalmers University of Technology and University of Gothenburg, Gothenburg, Sweden.

Hiroshi Maruyama. 1990. Structural disambiguation with contraint propagation. In *28th ACL 1989, Proceedings of the Conference*, pages 31–38, Pittsburgh, Pennsylvania, June 6-9.

John Myhill. 1960. Linear bounded automata. Wadd technical note, Wright Patterson AFB, Wright Air Development Division, Ohio, June.

Alexander Okhotin. 2013. Inverse homomorphic characterizations of Conjunctive and Boolean Grammars. Technical Report 1080, Turku Centre for Computer Science, Turku.

Janne Peltonen. 2011. Finite state Constraint Grammar parser. In *Proceedings of the NODALIDA 2011 workshop Constraint Grammar Applications, May 11, 2011*, volume 14 of *NEALT Proceedings Series*, Riga, Latvia.

P. S. Peters and R. W. Ritchie. 1973. On the generative power of transformational grammars. *Information Sciences*, 6:49–83.

Amir Pnueli. 1977. The temporal logic of programs. In *Proceedings of the IEEE 18th Annual Symposium on Foundations Computer Science*, pages 46–57, New York.

Daniel Průša. 2014. Weight-reducing Hennie machines and their descriptional complexity. In *Language and Automata Theory and Applications: 8th International Conference, LATA 2014, Madrid, Spain, March 10-14, 2014. Proceedings*, pages 553–564, Cham. Springer International Publishing.

Michael O. Rabin. 1963. Real time computation. *Israel Journal of Mathematics*, 1(4):203–211.

Eric Sven Ristad. 1990. Computational structure of generative phonology and its relation to language comprehension. In *Proceedings of the 28th Annual Meeting on Association for Computational Linguistics*, ACL '90, pages 235–242, Pittsburgh, Pennsylvania.

Kai Salomaa and Sheng Yu. 2000. Alternating finite automata and star-free languages. *Theoretical Computer Science*, 234:167–176.

Wojciech Skut, Stefan Ulrich, and Kathrine Hammervold. 2004. A bimachine compiler for ranked tagging rules. In *Proc. 20th Int'l Conf. on Computational Linguistics*, COLING '04, Stroudsburg, PA, USA.

Kohtaro Tadaki, Tomoyuki Yamakami, and Jack C. H. Lin. 2010. Theory of one-tape linear-time Turing machines. *Theoretical Computer Science*, 411(1):22–43.

Pasi Tapanainen. 1996. *The Constraint Grammar Parser CG-2*, volume 27 of *Publications*. Department of General Linguistics, University of Helsinki.

Pasi Tapanainen. 1999. *Parsing in two frameworks: finite-state and functional dependency grammar*. Ph.D. thesis, University of Helsinki, Finland, 1 December.

B. A. Trakhtenbrot. 1961. Finite automata and logic of monadic predicates. *Doklady Akademii Nauk SSSR*, 140:326–329. In Russian.

Boris A. Trakhtenbrot. 1964. Turing computations with logarithmic delay (in Russian). *Albegra i Logica*, pages 33–34. English translation in U. of California Computing Center, Tech. Report. No. 5, Berkeley, CA, 1966.

Atro Voutilainen and Pasi Tapanainen. 1993. Ambiguity resolution in a reductionistic parser. In *6th EACL 1993, Proceedings of the Conference*, pages 394–403, Utrecht, The Netherlands.

Atro Voutilainen. 1994. *Designing a Parsing Grammar*. Number 22 in Publications of the Department of General Linguistics, University of Helsinki. Yliopistopaino, Helsinki.

Anssi Mikael Yli-Jyrä and Kimmo Koskenniemi. 2004. Compiling contextual restrictions on strings into finite-state automata. In Loek Cleophas and Bruce W. Watson, editors, *The Eindhoven FASTAR Days, Proceedings*, number 04/40 in Computer Science Reports, Eindhoven, The Netherlands, December. Technische Universiteit Eindhoven.

Anssi Mikael Yli-Jyrä. 2003. Describing syntax with star-free regular expressions. In *11th EACL 2003, Proceedings of the Conference*, pages 379–386, Agro Hotel, Budapest, Hungary, April 12–17.

Anssi Yli-Jyrä. 2008. Transducers from parallel replace rules and modes with generalized lenient composition. In *Finite-State Methods and Natural Language Processing, 6th International Workshop, FSMNL-2007*, pages 197–212, Potsdam. Potsdam University Press.

Anssi Yli-Jyrä. 2011. An efficient constraint grammar parser based on inward deterministic automata. In *Proceedings of the NODALIDA 2011 workshop Constraint Grammar Applications, May 11, 2011*, volume 14 of *NEALT Proceedings Series*, Riga, Latvia.

Anssi Yli-Jyrä. unpublished. Efficient context-sensitive rewriting with inward deterministic transducers. Manuscript, 11 pages. Archived to EasyChair as a submission to PSC 2010 (Prague Stringology Conference 2010).