# CG-3 - Beyond Classical Constraint Grammar

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

**Tino Didriksen**
GrammarSoft ApS
tino@didriksen.cc

## Abstract

This paper discusses methodological strengths and shortcomings of the Constraint Grammar paradigm (CG), showing how the classical CG formalism can be extended to achieve greater expressive power and how it can be enhanced and hybridized with techniques from other parsing paradigms. We present a new, largely theory-independent CG framework and rule compiler (CG-3), that allows the linguist to write CG rules incorporating different types of linguistic information and methodology from a wide range of parsing approaches, covering not only CG's native topological technique, but also dependency grammar, phrase structure grammar and unification grammar. In addition, we allow the integration of statistical-numerical constraints and non-discrete tag and string sets.

## 1 Introduction

Within Computational Linguistics, Constraint Grammar (CG) is more a methodological than a descriptive paradigm, designed for the robust parsing of running text (Karlsson et al., 1995). The formalism provides a framework for expressing contextual linguistic constraints allowing the grammarian to assign or disambiguate token-based, morphosyntactic readings. However, CG's primary concern is not the tag inventory itself, or the underlying linguistic theory of the categories and structures used, but rather the efficiency and accuracy of the method used to achieve a given linguistic annotation. Conceptually, a Constraint Grammar can be seen as a declarative whole of contextual possibilities and impossibilities for a language or genre, but in programming terms, it is implemented procedurally as a set of consecutively iterated rules that add, remove or select tag-encoded information. In its classical form (Karlsson, 1990; Karlsson et al., 1995), Constraint Grammar relies on a morphological analyzer providing so-called cohorts of possible readings for a given word, and uses constraints that are largely topological[1] in nature, for both part-of-speech disambiguation and the assignment of syntactic function tags. (a-c) provide examples for close context (a) and wide context (b) POS rules, and syntactic mapping (c).

*(a)* REMOVE VFIN IF (0 N) (-1 ART OR <poss> OR GEN); *remove a finite verb reading if self (0) can also be a noun (N), and if there is an article (ART), possessive (<poss>) or genitive (GEN) 1 position left (-1).*

*(b)* SELECT VFIN IF (NOT *1 VFIN) (*-1C CLB-WORD BARRIER VFIN); *select a finite verb reading, if there is no other finite verb candidate (VFIN) to the right (*1), and if there is an unambiguous (C) clause boundary word (CLB-WORD) somewhere to the left (*-1), with no (BARRIER) finite verb in between.*

*(c)* MAP (@SUBJ) TARGET N (*-1 >>> BARRIER NON-PRE-N) (1C VFIN) ; *map a subject reading (@SUBJ) on noun (N) targets if there is a sentence-boundary (>>>) left without non-prenominals (NON-PRE-N) in between, and an unambiguous (C) finite verb (VFIN) immediately to the right (1C).*

As can be seen from the examples, the original formalism refers only to the linear order of tokens, with absolute (>>>) or relative fields

[1] With "topological" we mean that grammar rules refer to relative, left/right-pointing token positions (or word fields), e.g. -2 = 2 tokens to the left, *1 = anywhere to the right.

counting tokens left (-) or right (+) from a zero/target position in the sentence. Though in principle a methodological limitation, this topological approach also has descriptive "side effects": For instance, it supports local syntactic function tags (such as the @SUBJ tag on the head noun of an NP), but it does not easily lend itself to structural-relational annotation. Thus, dependency relations or constituent brackets can neither be created or referred to by purely topological CG rules[2]. Even chunking constraints, though topologically more manageable than tree structures, have to be expressed in an indirect way (cp. the NON-PRE-N barrier condition in example rule (c), and syntactic phrases cannot be addressed as wholes, let alone subjected to rewriting rules.

A second design limitation in classical CG concerns the expression of vague, probabilistic truths about language. Thus, the formalism does not allow numerical tags or numerical feature-value pairs, and while many current main stream NLP tools are based on probabilistic methods and machine learning, classical CG is entirely rule-based, and the only way to integrate likelihoods is through lexical "Rare" tags or by ordering rules in batches with more heuristic rules applying last.

Third, classical CG tags and tokens are discrete units and are handled as string constants. While this design option facilitated efficient processing and even FST methods, it also limited the linguist, who was not allowed to use regular expressions, feature variables or unification. Another aspect of discreteness concerns tokenization: Classical CG regarded token form, number and order as fixed, so the formalism had difficulty in accommodating, for instance, the rule-based creation of a (fused) named-entitity token, the insertion or removal of tokens in spell and grammar checking, or the reordering of tokens needed for machine translation.

Finally, when classical CG was designed, it had isolated sentences in mind. Though rule scope can be arbitrarily defined by a "window" delimiter set, and though "global" window rules clearly surpass the scope of HMM n-grams, it was not possible to span several windows at a time or to link referents across sentence, nor was it possible to contextually trigger genre variables or in other ways to make a grammar interact with a given text type. Descriptively, this limitation meant that CG as such could not be used for higher-level annotation such as anaphora or discourse relations, and that grammars were agnostic of genre and task types.

Following Karlsson's original proposal, two standards for CG rule compilers emerged in the late 90'ies. The first, CG-1, was used by Karlsson's team at Helsinki University and commercially by the spin-off company LingSoft for English (ENGCG), Swedish and German (GERCG) taggers, as well as for applied products such as Scandinavian grammar checkers (Arppe, 2000; Birn, 2000 for Swedish, Hagen et al., 2001 for Norwegian). The second compiler, CG-2, was programmed and distributed by Pasi Tapainen (1996), who made several notational improvements[3] to the rule formalisms (in particular, regarding BARRIER conditions, SET definitions and REPLACE operations), but left the basic topological interpretation of constraints unchanged. Five years later, a third company, GrammarSoft ApS, in cooperation with the University of Southern Denmark, launched an open source CG compiler, vislcg, which was backward compatible with CG-2, but also introduced a few new features[4], in particular the SUBSTITUTE and APPEND operators designed to allow system hybridization where input from a probabilistic tagger could be corrected with CG rules in preparation of a syntactic or semantic CG stage, as implemented e.g. in the earliest version of the French FrAG parser (Bick, 2004). Vislcg, too, was used in spell and grammar checkers (Bick, 2006a), but because of its open-source environment it also marked the transition to a wider spectrum of CG users and research languages.

[2] As a work-around, attachment direction markers (arrows) were introduced in the syntactic function tags, such as @>N or @N> for pre-nominal and @N< or @<N for post-nominal NP-material.

[3] Tapanainen also created a very efficient compiling and run-time interpretation algorithm for cg2, involving fintite state transducers, as well as a finite state dependency grammar, FDG (Tapanainen, 1997), for his company Conexor and its Machinese parsers.

[4] The vislcg compiler was programmed over several years by Martin Carlsen for VISL and GrammarSoft. For a technical comparison of CG-2 and vislcg, cf. http://beta.visl.sdu.dk/visl/vislcg-doc.html .

But though constraint grammars using the CG-2/VISLCG compiler standard did achieve a tag granularity and accuracy that allowed them to support external modules for both constituent and dependency tree generation, they remained topological in nature and did not permit explicit reference to linguistic relations and structure in the formalism itself. The same is true for virtually all related work outside the CG community itself, where the basic idea of CG constraints has sometimes been exploited to enhance or hybridize HMM-style probabilistic methods (e.g. Graña et al., 2003) or combined with machine learning (Lindberg & Eineborg, 1998; Lager, 1999), but always in the form of (mostly close-context) topological rather than structural-relational rules and always with discrete tag and string constants. It is only with the CG-3 compiler presented here, that these and most of the other above-mentioned design issues have been addressed in a principled way and inside the CG formalism itself. CG-3[5] (or VISL CG-3 because of its backward compatibility with VISLCG) was developed over a period of 6 years, where new features were designed and implemented continually, while existing features were tested in real-life parsing applications. In the following sections we will discuss the most important of these features and compare the finished framework with other approaches.

## 2 Expressive power: Relational tags

In CG all information is expressed as token-based tags, and this is true of CG-3 relational tags, too. Though each token can be part of any number of relations, the individual relation is binary, linking a from- and a to-token. Dependency annotation can be seen as a special type of such a relation, where each dependent (daughter, child) is assigned exactly one head (mother, parent), but where each head can have any number of dependents. In CG-3, we mark dependency relations on the daughter token with a #n->m tag, where 'n' is the token id of the dependent, and 'm' the token id of the head. Thus, dependency is a tag field, just like the ".."-marked lemma field, the upper-case POS and inflection fields or the @-marke syntactic function field[6]:

**Both** "both" <quant> DET P @>N #1->2
**companies** "company" <HH> N P @SUBJ> #2->3
**said** "say" <speak> <mv> V IMPF @FS-STA #3->0
**they** "they" <clb> PERS 3P NOM @SUBJ> #4->5
**would** "will" <aux> V IMPF @FS-<ACC #5->3
**lauch** "launch" <mv> V INF @ICL-AUX< #6->5
**an** "a" <indef> ART S @>N #7->9
**electric** "electric" <jpert> ADJ POS @>N #8->9
**car** "car" <Vground> N S NOM @<ACC #9->6
**.** "." PU @PU #10->0

Instead of the "topological" left/right-pointing position markers, CG rules with dependency contexts can refer to three types of relations: p (parent/head), c (child/dependent) and s (sibling).

ADD (§AG) TARGET @SUBJ (p V-HUM LINK c @ACC LINK 0 N-NON-HUM) ;

*(Add an AGENT tag to a **subject** reading if its parent verb is a human verb that in turn has a child accusative **object** that is a non-human noun. E.g. "**BMW** launched an electric **car**.")*

In order to add dependency annotation to ”virgin” input, the operators SETPARENT and SETCHILD are used together with a TO target. Thus, for the sentence *"We **know** for a fact **that** the flat **had** not been used in months."*

SETPARENT @FS-<ACC (*-1 (”that” KS) BARRIER CLB TO (**-1 <mv> LINK 0 V-COGNITIVE) (NOT 1 @<ACC);

will link a finite object clause (underlined, marked @FS-<ACC on *'had'*) with a that-conjunction to a main verb (<mv>) anywhere to the left (**-1) if the latter is a cognitive verb (V-COG) and is not followed by an ordinary direct object (@<ACC). Both the SET-target and the TO-target can have their own independent context conditions, and that these can either be traditional positional contexts, or exploit already established dependency relations. CG-3 has a built-in check against circularity, preventing attachments that would create a dependency loop[7]. Dependency operators can be combined with a number of options:

[5] For detailed technical documentation on CG-3, cf. http://beta.visl.sdu.dk/cg3.html, for tutorials, associated tools, parser demos and resources, see http://visl.sdu.dk/constraint_grammar.html. Cagetories and tag abbreviations are explained at http://visl.sdu.dk/tagset_cg_general.html .

[6] All of these fields can easily be converted into xml-encoded feature-value pairs for compatibility. The authors provide scripts for conversion into e.g. MALT xml and TIGER xml.

1.) * *(Deep scan)* allows a child- or parent-test to continue searching along a straight line of descendants and ancestors, respectively, until the test condition is matched or until the end of a relation chain is reached.

2.) *C (All scan)* requires a child- or sibling-relation to match *all children* or *all* siblings, respectively. Note that this is different from the ordinary C (= safe) option which applies to readings. Thus 'cC ADJ' means 'only adjectives as children' – e.g. no articles or PP's, while 'c (*) LINK 0C ADJ' means 'any one daughter with an unambiguous adjective reading'.

3.) *S (Self)* can be combined with c, p or s to look at the current target as well. For example, 'c @SUBJ LINK cS HUM' looks for a human subject NP – where either the head noun (@SUBJ) itself is human, or where it has a modifier that is tagged as human.

Apart from dependency relations, we also allow general *named* relations in CG-3, that can be used for arbitrary relation types, such as secondary dependencies between object and object complement, anaphora (Bick, 2010), discourse relations etc. Thus, the following establishes an identity relation between a relative pronoun and its noun antecedent:

```
SETRELATION (identity) TARGET (<rel>)
    TO (*-1 N)
```

Where matched, this will add a relational tag on the pronoun token: *ID:n R:identity:m*, where R: specifies the relation, and n and m are token id's for the pronoun and noun, respectively.

It is even possible to set bidirectional relations with separate labels, to be tag-marked at both ends of the relation arc. Thus, the example rule sets a relation between a human noun subject and a sense-verb object, labelling the former as "experiencer" and the latter as "stimulus":

```
SETRELATIONS (experiencer) (stimulus)
    TARGET N-HUM + @SUBJ TO (p V-SENSE
    LINK c @ACC) ;
```

[7] Though descriptively undesirable, loops can be explicitly allowed with the ALLOWLOOP and NEAREST options (cf. visl.sdu.dk/cg3.html)

# 3 Constituent structure: Inspiration from the generative paradigm

Because dependency syntax bases its structural description on tokens (words), it is inherently closer to the native CG approach than the competing generative family of syntactic formalisms, which operate with non-terminal nodes and constituent brackets.

## 3.1 Tree transformation

Classical CG does not support constituent brackets in any form, be it flat chunks or nested constituents, so external modules had to be used to create constituent trees. The oldest example are PSGs with CG functions as terminals (Bick, 2003), used for CALL applications within the VISL project, followed by dependency-to-constituent tree transformation employing an external dependency grammar (Bick, 2005; Bick, 2006b). Of course the same transformation could be used with our new, native CG dependency (cp. previous section), but CG-3 does offer more direct ways to express linguistic structure in generative terms, allowing linguists used to *think* along PSG lines to directly translate generative descriptions and constraints into the CG formalism.

## 3.2 Chunking

There are at least two distinct methods in CG-3 to perform chunking, using either (a) cohort insertion or (b) relation-adding. For traditional, shallow chunking, without overlaps and nesting, only about 20 rules are needed (Bick, 2013), inserting opening (a) and closing (b) edge marker tokens.

```
(a) ADDCOHORT ("<$np>" "CHUNK" NP)
BEFORE @>N OR N/PROP/PRON OR
DET/NUM/PERS - @ATTR (NOT -1 @>A OR
@>N) (NEGATE -1 IT LINK -1 @>N) ;

(b) ADDCOHORT ("<$/np>" "ENDCHUNK"
NP) AFTER N/PROP/PRON OR DET/NUM/PERS
- @ATTR (NOT 0 @>N) (*-1 CHUNK-NP
BARRIER CHUNK) ;
```

NP opening markers (a) are inserted before prenominal noun dependents (@>N) or NP heads (N/PROP/PRON), accepting even determiners and numerals if they have no attributive function (@ATTR). Likewise, NP closing markers (b) are inserted *after* the above NP head candidates, in the presence of the left-

hand (*-1) NP-chunk opener. The NOT contexts in (a) make sure that the triggering prenominal is in fact the *first* element in the NP, and not preceded by an adverbial dependent of its own, or part of a coordination. The inserted chunk-opening and -closing tokens can then be interpreted as labelled brackets: **(np** *We_PRON* ***/np)*** *had* ***(np*** *very_@>A delicious_@>N icecream_N* ***/np)*** *with* ***(np*** *strawberries_N* ***/np)***.

The second method is better suited for layered, deep chunking, because it uses relational tags to individually link chunk edges to each other or to the chunk head. With full layering, this approach can create complete xml-formated constituent trees from CG dependency-tagged input without the need of an external converter, if chunk brackets are expressed as xml opening/closing markers. However, using relations to delimit topological units such as chunks, introduces certain complexities in the face of crossing branches and needs to specify the "handedness" (left/right) and "outermostness" of dependency arcs, features that are normally left underspecified in dependency annotation. In CG3, we support these features as l/r- (left/right) and ll/rr- (leftmost/rightmost) additions:

(a) ADDRELATIONS (np-head-l) (np-start) TARGET (*) (c @>N OR @N<&) TO (llScc (*)) ;

(b) ADDRELATIONS (np-head-r) (np-stop) TARGET (*) (c @>N OR @N<&) (r:np-head-l (*)) TO (rrScc (*)) ;

Both rules are bidirectional and mark both chunk head and chunk edges. The head target is any word (*) with an adnominal dependent (c @>N OR @N<), and the TO-edge is the leftmost (ll) resp. rightmost (rr) descendant (cc) or self (S). This second method will yield complete, nested structures, including adjective phrases (adjp) and prepositional phrases (pp) in the NPs: ***(np-start (adjp-start*** *very_@>A delicious_@>N* ***adjp-stop)*** *icecream_N* ***(pp-start*** *with_PRP_@N< strawberries_@P<* ***pp-stop) np-stop)***[8].

### 3.3 Phrase templates

Both of the above chunking methods are intended to be used late in the annotation pipe, and exploit existing morphosyntactic markup or even dependencies, so the chunking cannot itself be seen as methodological part of parsing *per se*. However, CG-3 also offers another way of expressing chunks, the *template,* which can be integrated into CG rules also at early tagging stages. A template is basically a pre-defined sequence of tokens, POS or functions that can be referred to as a whole in rule contexts, or even in other templates. The basic idea goes back to Karlsson et al. (1995), but was not implemented in either CG-1 or CG-2.

For instance, an NP could be defined as

(a) TEMPLATE np = ([ART, N]) OR ([ART,ADJ,N])

(b) TEMPLATE np = (? ART LINK 1 N) OR (? ART LINK 1 ADJ LINK 1 N)

(c) TEMPLATE np = ? ART LINK *1 N BARRIER NON-PRE-N

and then used in ordinary rules with a T:-prefix

(*1 VFIN LINK *1 T:np).

(a) is closest to the original idea, and reminiscent of generative rewriting rules, while (b) and (c) are shorthand for ordinary CG contexts and harness the full power of the latter. Independently of the format, however, the linguistic motivation behind templates is to allow direct reference to constituent units, to *think* in terms of phrase structure and to subsume aspects of generative grammar into CG. Thus, constituent templates allow a direct conceptual transfer from generative rules, and a simple generative NP grammar for the NP *"a very delicious icecream with strawberries"*:

np = adjp? n pp? ;
adjp = adv? adj ;
pp = prp np ;

could be expressed in CG3 as:

```
TEMPLATE np = (N)
        OR (T:adjp LINK 1 N)
        OR (T:adjp LINK 1 N LINK 1 T:pp)
        OR (N LINK 1 T:pp) ;
TEMPLATE adjp = (ADJ) OR (ADV LINK 1
ADJ) ;
TEMPLATE pp = PRP LINK 1 T:np ;
```

In the example, "*very_ADV delicious_ADJ"* matches T:adjp, and *"with_PRP icecream_N"* matches T:pp, and the whole expression could then be referred to as a T:np context by CG rules. CG-internally, templates could also simply be interpreted as shorthand (variables) for context parentheses, so-called *context templates*. As such, they logically need to allow *internal*, predefined

[8] For clarity, only phrases with 2 or more constituents were bracketed in the 2nd method.

positions, as in the following example for a human verb-template, where the motivation is not a constituent definition, but simply to integrate two context alternatives into one[9], and to label the result with one simple variable.

TEMPLATE v-hum = ((c @SUBJ + HUM) OR (*1 (”that” KS) BARRIER V)) ; *"human verb" defined as either having a subject (@SUBJ) child (c) that is human (HUM), or having a subordinating conjunction (KS) anywhere to the right (*1) without another verb (V) in between.*

Compiler-internally, both template types are processed in a similar way, which is why constituent templates have question marks or 0-positions as place holders for an external position marker, which will be inserted into the template by the compiler at run-time.

When using templates together with (external) BARRIER's or LINKed conditions, the template can be thought of as one token – meaning that right-looking contexts with a template (*1 T:x BARRIER) will be interpreted against the left edge of the template, while left-looking contexts (*-1 T:x) will be interpreted against the right edge of the template so as to avoid internal, unpredictable parts of the template itself to trigger the BARRIER condition.

## 4 Beyond discrete tags and string constants: Regular expressions, variables and unification

A formal grammar has to strike a balance between computational efficiency on the one hand, and linguistic ease and rule writing efficiency on the other. Thus, the "classical" CG compilers treated tags and strings (lemma & word form) as constants and CG-2, in particular, achieved very high processing speeds exploiting this fact in its finite state implementation. Some flexibility was introduced through set definitions, and vislcg went on to allow sets as targets, too, as well as multiple conditions for the same position, but many rules had still be to be written in multiple versions because of expressive limitations in the formalism:

(a) OR'ing only for tags/sets, not contexts
(b) no nesting of NOT conditions
(c) no C-restriction for BARRIERs

CG-3 adds all of the above[10], but while increasing rule-writing efficiency, these changes to not affect the discreteness of tags and strings. Methodologically more important, therefore, is our introduction of regular expressions and variables. The former can be used instead of sets for open-class items, primarily lemma and word class, e.g. ".*i[zs]e"r V in a transitivity set or ".*ist" N as a heuristic candidate for the <Hprof> or <Hideo> classes ("professional" or "ideological" humans). But the feature is useful even with a closed-class semantic set such as +HUM, and <H.*>r will work across grammars and languages leaving grammarians the option to introduce ad hoc sub-distinctions (e.g. <Hsick> for words like 'diabetic'). Finally, regular expressions can be used to substitute for, or enhance, morphological analysis, for instance in stemming or affix recognition, supporting the creation of so-called "barebones" Constraint Grammars without lexical resources (Bick, 2011).

Variables can be used in connection with regular expressions, when appending readings (a) or for instantiating valency conditions (b):

APPEND ("$1"v ADJ) TARGET ("<(.*(ic|oid|ous))>"r) ; # recognizing adjective endings

REMOVE (N) (0 (<(.+)^vp>r INF)) (-1 INFM) (1 ("$1"v PRP)) ; # e.g. *to minister to the tribe*

With the example given, the second rule can remove the noun reading for 'minister' because the 'to' in the valency marker <to^vp> of the verb 'minister' matches the lemma "to" of the following preposition, even if the infinitive marker is still unsafe and potentially a preposition itself (-1 rather than -1C).

The methodologically most important use of variables, however, resides in feature unification. Thus, CG3 allows the use of sets as to-be-unified variables by prefixing $$ before the set name. Set unification integrates yet another methodological feature, used in other parsing paradigms, such as HPSG, but so far accessible in CG only at the cost of considerable "rule explosion". Apart from the obvious gender/number/case-disambiguation of noun phrases, unification is also useful in for

[9] In traditional CG, this OR'ed expression could not even be expressed in one rule, let alone be referenced as one label.

[10] The nesting of NOT conditions is achieved by making a distinction between ordinary NOT, that only negates its immediate position, and NEGATE, which a scope over the whole context bracket - including other NOTs or NEGATEs.

instance coordination, as with the following LIST set of semantic roles (agent, patient, theme and location):

```
LIST ROLE = §AG §PAT §TH §LOC ;

SELECT $$ROLE (-1 KC) (-2C $$ROLE) ;
```

Sometimes unification has to be vague in order to work. This is the case when underspecified "Portmanteau" tags are used (e.g. nC - nocase unified with NOM or ACC cases), or in the face of very finegrained semantic distinctions. We therefore make a distinction between list unification ($$-prefix) and set unification (&&-prefix), where the former unifies "terminal" set members, while the latter unifies subsets belonging to a superset. Two contexts will set-unify if they have tags sharing the same subset. In the example below, N-SEMS is defined as a superset, with N-SEM as one of the subsets.

```
LIST N-SEM = <sem> <sem-l> <sem-r> <sem-
    w> <sem-c> <sem-s> <sem-e> <coll-sem>
    <sem-nons> <system> <system-h> ;

SET N-SEMS = N-HUM OR N-LOC  ... OR N-
    SEM ... OR N-SUBSTANCE ;

REMOVE @SUBJ>
    (0 $$@<ARG LINK 0 &&N-SEMS)
    (*-1 KC BARRIER NON-PRE-N/ADV LINK
    *-1C $$@<ARG BARRIER CLB-ORD OR
    &MV OR @ARG/ADVL> LINK 0 &&N-
    SEMS) ; # ... offered the reader detailed notes
    and instructions on most of the prayers ...
```

The example sentence has an ambiguous coordination, where it is not clear if 'and' starts a new clause, and the task of the REMOVE rule is to exclude a subject reading for 'instructions' (tagged <sem-s>) by semantically aligning it with 'notes' (tagged <sem-r>) because both <sem-r> and <sem-s> are part of the N-SEM subset of the &&N-SEMS superset, - and by checking if both nouns also have matching left-pointing argument readings ($$@<ARG), in this case @<ACC (direct objects).

## 5 Integrating statistical data: Numerical tags

CG-3 moves beyond traditional <Rare> sets and heuristic rule batching by allowing rules to make reference to statistical information. This is achieved by introducing numerical secondary tags of the type <LABEL:number>, which can be used to encode and use corpus-harvested frequencies. The simplified example rule (a) exploits relative lexical POS frequencies for bigram disambiguation in a way reminiscent of hidden Markov models (HMMs), while (b) is a spell checker fall-back rule selecting the word with the highest phonetical similarity value

```
REMOVE (<fr<10> N) (0 (<fr>60> V)) (1 N)

SELECT (<PHONSIM=MAX>)
```

A more complex example is the use of CG-annotated data to boot-strap statistical "wordnets" or "framenets", containing the likelihood of semantic types or roles given an established syntactic function. Thus, the Portuguese PALAVRAS parser (Bick, 2014) assigns and exploits tags like <fSUBJ/H:41>, <fSUBJ/org:27> and <fACC/deverbal:53> for the verb *"propor"* (suggest), meaning that *"propor"* has a 53% probability of having a deverbal direct object (action/activity/process/ event), and subject likelihoods of 41% and 27% for person and organization, respectively[11].

Obviously, numerical tags could be used for other ends than statistics, for instance to assign confidence values to mapped syntactic tags or semantic roles, or for similarity degrees in spell-checking. Finally, using only the equal-operator, numerical tags can be seen as a special case of (numerical) global variables, e.g. for numbered genre types or Wordnet synset id's.

## 6 Grammar-text interaction

The fourth and last design limitation of classical CG to be treated here concerns ways to let a constraint grammar mold itself on the fly and to adapt to the text (or speech transcript) it is used to annotate. In CG-3 we introduce 3 types of such self-organizing behaviour:

(a) scope control
(b) rule or section triggering
(c) parameter variables

Scope control is achieved by allowing the grammarian not only to define window (read: sentence) delimiters, but also a spanning width of n windows left or right of the rule focus. Unbounded context conditions can breach

[11] Simplifying, we here only list high-percentage semantic types for subjects and objects.

window boundaries by adding a 'W', e.g. *-1W for scanning left across the window boundary. This feature is especially useful for higher-order relations such as anaphora (Bick, 2010) or discourse relations. Another scope-related innovation are (definable) paired brackets that allow rules to scan across brackets in a first pass, and make reference to them in a second pass. Like templates, bracket eclipsing is meant to help reduce CG's topological complexity problem, i.e. allow syntactic function carriers to "see" each other more easily across intervening tokens.

CG-3, unlike earlier CG compilers, applies rules strictly sequentially, and each rule is run on *all* cohorts in a window before the next rule is tried. This makes rule tracing more predictable, but also facilitates grammar self-organisation. Thus, we allow context-triggered JUMPs to rule ANCHORs, to INCLUDE additional rules from a file or to call EXTERNAL programs. For instance, an early rule can scan the window for verbo-nominal ambiguities, and if there are none, bypass the rule section in question.

Because CG does not depend on training data, it is generally assumed to be more genre-robust than machine-learning systems[12], and a few manual rule changes will often have a great effect on genre tuning (e.g. allowing/forbidding imperative readings for recipes or science articles, respectively). In CG-3, we further enhance this methodological advantage by introducing parameter variables, that can be set or unset either in the data stream (e.g. corpus section headers) or dynamically-contextually by the grammar itself. The example rule below assigns the value "recipe" to a "genre" variable, when encountering imperatives followed by quantified food nouns.

SETVARIABLE (genre) (recipe) TARGET (IMP) (*1 N-FOOD LINK *-1 NUM OR N-UNIT BARRIER (*) - ("of"))

Finally, grammar-text interaction may take the form of rule-governed changes to the text itself. Thus, the ADDCOHORT feature used for chunking in section 3.2., and its REMCOHORT counterpart can be used for adding or removing commas in grammar checking, and the MOVE BEFORE, MOVE AFTER and SWITCH WITH operators can be used to express syntactic movement rules in machine translation. The example rule will change Danish VS into English SV in the presence of a fronted adverb: MOVE WITHCHILD[13] (*) @<SUBJ BEFORE (*-1 VFIN) (-1 ADV LINK -1 >>>).

Applied to the Danish sentence "I går *så jeg* et rensdyr", this will turn the literal translation "Yesterday *saw I* a reindeer" into the correctly ordered "Yesterday *I saw* a reindeer".

## 7 Efficiency and hybridization options

This paper is primarily concerned with design aspects and a linguistic discussion of the CG-3 formalism, and advances in expressive power have been the main focus of innovation during development. That said, the CG-3 rule parser compiles mature grammars with thousands of rules in fractions of a second and maintains the processing speed of VISLCG inspite of the added complexity caused by regular expressions, variables, templates and numerical tags. For a mature morphosyntactic core grammar with 6000 rules, on a single machine, this amounts to ~1000 words (cohorts) per second for each of the morphological and syntactic levels. However, Yli-Jyrä (2011) has shown that much higher speeds (by about 1 order of magnitude[14] on a comparable machine) are possible, at least for VISLCG-compatible rules without the above complexities, when using a double finite-state representation, where rule conditions are matched against a string of feature vectors that summarize compact representations of local ambiguity. Future work should therefore explore the possibility of sectioned grammars, where a distinction is made between FST-compatible rule sections on the one hand, and smaller specialized rule sections on the other hand, which for their part would allow the complete range of CG-3 features. This way, simpel "traditional" rules would run at the higher FST speed, and the current procedural compiler architecture would only be used where necessary, greatly reducing overall processing time.

[12] The rationale for this is that an ML system basically is a snapshot of the linguistic knowledge contained in its training data, and therefore will need new training data for each new genre in order to perform optimally.

[13] The WITHCHILD option means that heads are moved together with their dependents, in this case "reindeer" together with "a".

[14] The reported speed is 110,000 cohorts for FINCG, an open morphological CG with ~ 950 low-complexity CG-1 rules, originally developed by Fred Karlsson for Finnish.

## References


Arppe, Antti. 2000. Developing a grammar checker for Swedish. In: Torbjørn Nordgård (ed.). *Proceedings of NODALIDA '99.* pp. 28-40. Trondheim: Department of Linguistics, University of Trondheim.

Bick, Eckhard. 2003. A CG & PSG Hybrid Approach to Automatic Corpus Annotation. In: Kiril Simow & Petya Osenova (eds.). *Proceedings of SProLaC2003* (at Corpus Linguistics 2003, Lancaster), pp. 1-12

Bick, Eckhard. 2004. Parsing and evaluating the French Europarl corpus, In: Patrick Paroubek, Isabelle Robba & Anne Vilnat (red.): *Méthodes et outils pour l'évaluation des analyseurs syntaxiques* (Journée ATALA, May 15, 2004). pp. 4-9. Paris: ATALA.

Bick, Eckhard. 2005. Turning Constraint Grammar Data into Running Dependency Treebanks. In: Civit, Montserrat & Kübler, Sandra & Martí, Ma. Antònia (red.). *Proceedings of TLT 2005* (4th Workshop on Treebanks and Linguistic Theory, Barcelona), pp.19-27

Bick, Eckhard. 2006a. A Constraint Grammar Based Spellchecker for Danish with a Special Focus on Dyslexics". In: Suominen, Mickael et.al. (ed.) *A Man of Measure: Festschrift in Honour of Fred Karlsson on his 60th Birthday*. Special Supplement to SKY Jounal of Linguistics, Vol. 19. pp. 387-396. Turku: The Linguistic Association of Finland

Bick, Eckhard. 2006b. Turning a Dependency Treebank into a PSG-style Constituent Treebank. In: Calzolari, Nicoletta et al. (eds.). *Proceedings of LREC 2000.* pp. 1961-1964

Bick, Eckhard. 2010. A Dependency-based Approach to Anaphora Annotation. In: (eds.) *Extended Activities Proceedings, 9th International Conference on Computational Processing of the Portuguese Language* (Porto Alegre, Brazil). ISSN 2177-3580

Bick, Eckhard. 2011. A Barebones Constraint Grammar. In: Helena Hong Gao & Minghui Dong (eds), *Proceedings of the 25th Pacific Asia Conference on Language, Information and Computation* (Singapore). pp. 226-235

Bick, Eckhard. 2013. Using Constraint Grammar for Chunking. In: S. Oepen, K. Hagen & J. B. Joannessen (Eds). *Proceedings of the 19th Nordic Conference of Computational Linguistics (NODALIDA 2013).* Linköping Electronic Conference Proceedings Vol. 85, pp. 13-26. Linköping: Linköping University Electronic Press.

Bick, Eckhard. 2014. PALAVRAS, a Constraint Grammar-based Parsing System for Portuguese. In: Tony Berber Sardinha & Thelma de Lurdes São Bento Ferreira (eds.), *Working with Portuguese Corpora*, pp 279-302. London/New York: Bloomsburry Academic.

Birn, Jussi. 2000. Detecting grammar errors with Lingsoft's Swedish grammar checker. In: Torbjørn Nordgård (ed.). *Proceedings of NODALIDA '99.* p. 28-40. Trondheim: Deparment of Linguistics, University of Trondheim.

Graña, Jorge & Gloria Andrade & Jesús Vilares. 2003. Compilation of constraint-based contextual rules for part-of-speech tagging into finite state transducers. In: *Proceedings of the 7th Conference on Implementation and Application of Automata (CIAA 2002)*. pp. 128–137. Springer-Verlag, Berlin.

Hagen, Kristin & Pia Lane & Trond Trosterud. 2001. En grammatikkontrol for bokmål. In: Kjell Ivar VAnnebo & Helge Sandøy (eds). *Språkknyt 3-2001, pp.* 6-9. Oslo: Norsk Språkråd

Karlsson, Fred. 1990. Constraint Grammar as a Framwork for Parsing Running Text. In: Hans Karlgren (ed.). Proceedings of COLING-90, Vol. 3, pp. 168-173

Karlsson, Fred & Atro Voutilainen & Juha Heikkilä & Arto Anttila. 1995. *Constraint Grammar: A language-independent system for parsing unrestricted text.* Natural Language Processing 4. Berlin & New York: Mouton de Gruyter.

Lager, Torbjörn. 1999. The μ-TBL System: Logic Programming Tools for Transformation-Based Learning. In: *Proceedings of CoNLL'99 (Bergen)*.

Lindberg, Nikolaj & Martin Eineborg. 1998. Learning Constraint Grammar-style Disambiguation Rules Using Inductive Logic Programming. In: *Proceedings of the 36th ACL / 17th COLING (Montreal, Canada)*. volume 2, pages 775–779

Tapanainen, Pasi. 1996. The Constraint Grammar Parser CG-2. No 27, Publications of the Department of Linguistics, University of Helsinki.

Tapanainen, Pasi. 1997. A Dependency Parser for English. Technical Reports No TR1. Department of Linguistics, University of Helsinki.

Yli-Jyrä, Anssi Mikael. 2011. An Efficient Constraint Grammar Parser based on Inward Deterministic Automata. In: *Proceedings of the NODALIDA 2011 Workshop ConstraintGrammar Applications*, pp. 50-60 NEALT ProceedingsSeries , vol. 14