Turning Constraint Grammar Data into Running Dependency Treebanks

Eckhard Bick
Institute of Language and Communication, University of Southern Denmark
eckhard.bick@mail.dk, Rugbjergvej 98, DK-8260 Viby J

1. Introduction

Traditional Constraint Grammar (CG) is a methodological, rather than a descriptive paradigm, designed for robust parsing, not the implementation of a specific linguistic theory. Therefore, if used for treebank generation, it is not immediately clear, which linguistic formalism would be easiest to support, and whether the same underlying CG (in this case VISL-style CGs) can be used for different descriptive formats. In this paper, we present and evaluate a grammar-based method that attempts to bridge the gap between raw CG annotation and a full dependency structure annotation, allowing CG output to be turned into regular dependency treebanks rather than the phrase structure grammar-mediated constituent treebanks used by the VISL teaching tools.

1.1. CG dependency styles

Since CG-rules address word based tags, all levels of linguistic information have to be expressed as tags. Thus, syntactic structure is usually encoded as function tags (subject, object etc.) with or without some directional dependency information. However, since both dependency and constituent information is implicit and underspecified in classical CG, added tools, or manual revision, are necessary to create real treebanks from CG-annotated text. In one approach, Tapanainen and Järvinen (1997), describe an integrated parsing formalism (Finite Dependency Grammar, FDG) implementing full dependency structure between words or Tesnière-style multi-word nuclei, but most CG parsers, including the ones used by the author's VISL project, have been optimised for what could be called ”robust shallowness”, i.e. a maximally safe disambiguation of part of speech and syntactic function, rather than deep/complete structure. Also, there is no shared standard as to the handling of multi-word expressions, which remain lexical units rather than syntactic nuclei or ”constituents”, calling for a simple, token-based approach. The challenge, then, is a two-stage approach, where existing (CG-) tools can be exploited to the full, to be subsequently enriched with deep-structure information and adapted to a user-defined descriptive model of a given language.

2. A CG-based treebank with full dependency specification

The Danish Arboretum treebank (Bick 2003-1) at the University of Southern Denmark is maintained as a double-format treebank, where manually corrected CG output from DanGram (Bick 2001) is used as input to a specialised PSG (Bick 2003-2), and corrected once again at the constituent tree level. The conversion into dependency treebank format is handled using (”ordinary”) TIGER-XML constituent format as an intermediate format, which in turn is filtered into the Nordic Treebank Network's recommended TIGER dependency format (http://www.id.cbs.dk/~mtk/ntn/tiger-xml.html).

The problem with this method is that the cg2psg stage is somewhat less robust than CG itself, producing around 80% well-formed trees even for corrected CG input, and 50-70% on raw text. Therefore, we have examined the possibility of adding full dependency information directly to the CG format, without a PSG stage. This task has earlier been addressed with a declarative, Prolog-based method by Søren Harder[1], but for the sake of robustness, a new, procedural system *(cg2dep)*, with a compiled grammar of sequential attachment rules, has been developed by the author. From a theoretical point of view, it must be stressed that both the cg-to-psg and the cg-to-dependency methods have been implemented as linguist-written grammars and can be classified as ”structure-after-function”.

In the example below, the CG-function tag is supplemented by a dependency link specifying a target head (->) for each token (number #).

```
Når [når] KS @SUB #1->4
Sofies [Sofie] PROP GEN @>N #2->3
mor [mor] N UTR S IDF NOM @SUBJ> #3->4
var [være] V IMPF AKT @FS-ADVL> #4->9
sur [sur] ADJ UTR S IDF NOM @<SC   #5->4
over [over] PRP @A< #6->5
et=eller=andet [en=eller=anden] DET NEU S NOM @P< #7->6
$, #8->0
skete [ske] V IMPF AKT @FS-STA #9->0
det [den] PERS NEU 3S NOM @F-SUBJ #10->9
at [at] KS @SUB #11->13
hun [hun] PERS UTR 3S NOM @SUBJ> #12->13
kaldte [kalde] V IMPF AKT @FS-<SUBJ  #13->9
deres [de] PERS 3P GEN @>N #14->15
hus [hus] N NEU S IDF NOM @<ACC   #15->13
for [for] PRP @<OC   #16->13
et [en] ART NEU S IDF @>N #17->19
```

[1]Søren Harder's system, the *Depsplicator*, was developed as part of a Ph.D. project and is accessible in the Danish machine analysis section of http://visl.sdu.dk.

```
værre [dårlig] ADJ COM nG nN nD NOM @>N #18->19
menageri [menageri] N NEU S IDF NOM @P< #19->16
$. #20->0
</s>
```

3. The rule formalism

The basic idea of the formalism is that a grammar of ordered rules tells CG tags (especially function tags) which other tags they can attach to, and in which direction, left or right:

```
(a1) @<ACC -> (<mv>) IF (L)
(a2) @SUBJ> -> (PR,IMPF) IF (R)
```

The first rule attaches left-pointing direct object tags (@<ACC) to the next free main verb head target (<mv>), if it can be found to the left (L) without creating circular dependencies. As the second, subject attaching, rule shows, targets can be sets, like verb tenses (PR,IMPF). The grammar contains a set definition section, where set names can be defined:

```
(b) ¤LEFT-NPHEAD =
        @P<,@<SUBJ,@<ACC,@<DAT,@<SC,@<OC,@APP,@N<PRED
```

It is important to note that both the to-be-attached dependent candidate and the to-be-found head candidates allow what in other formalisms might be called subcategorisation or selection restrictions, the distinction simply being a question of including different types of tags in the match-string. Thus, a verb tense tag like PR (present) can be combined with a morphological tag like PAS (passive) or a valency-potential tag like <vt> (monotransitive verb). Likewise, a function tag like @SUBJ can be semantically restricted as human by prepending <H.*> (i.e. <H.*> @SUBJ), matching the categories of <Hprof> (professions), <Hfam> (family relations), <Hideo> (supporter of ideology, e.g. 'kommunist') etc.

3.1. Context conditions

Apart from the direction condition (L,R), some other types of context conditions have been implemented:

```
(c1) @FS-@N< -> (¤NPHEAD, N.*@N<)
   IF (L) TRANS:(@SUBJ>,@F-SUBJ>,@S-SUBJ>)

(c2) @ADVL> -> (<mv>)
   IF (R) BARRIER (@SUBJ>,@F-SUBJ>,@S-SUBJ>
(c3) <np-close> -> (DET)
    IF (L) HEADCHILD=(@>N)

(c4) @N< -> (N,PROP,PERS,INDP,¤NPHEAD)
    IF (L) NOTHEAD=(<clb>) NOTTARGET=(@FS-@N<)
```

The **TRANS** condition identifies tokens that have to be ”crossed” before attachment to a legal head is allowed. In the example (c1), a relative clause (@FS-N<) is prevented from attaching to its own subject (while still being allowed to attach to the *next* subject). The **BARRIER** condition is the opposite of the TRANS-condition: It will stop the search for a suitable head in the direction given. In the example (c2), right adverbial attachment to main verbs is blocked by subject. The **HEADCHILD** condition in the third rule (c3), finally, allows postnominal (pp-) attachment to the non-standard head candidate determiner (DET), *if* the determiner has itself a prenominal modifier (@>N) and thus, status of np-head. Finally, conditions can be negated. In the most general postnominal attachment rule (c4), for instance, there is a condition (**NOTHEAD**) preventing attachment to e.g. relative pronoun subjects (<clb>), and another one (**NOTTARGET**) excluding verbal heads of relative clauses (@FS-@N<) as rule targets altogether.

3.2. Forced and inverted attachment

```
(d1) <quote> -> (@FS-@STA,<v-quote>)IF (R) (F)

(d2) (PR|IMPF).*@FS-@N< -> (@SUBJ>,@[FS]-SUBJ>)
    IF (L) (D) BARRIER:(@>>P,PR,IMPF)
```

The formalism also allows to ”force” **(F)** or ”invert” rules **(D)**. Forced attachments cannot be undone, and win over possible competitors in circularity contests, while D-rules search from head to dependent rather than the other way around. Thus, a forced dependency connection (d1) is established between quoted clauses <quote> and the quoting verb <v-quote> or, if un-annotated, the top node verb (@STA) to the right, preventing closer, but wrong attachment to an intervening subclause verb.

Since attachment targets are ”tested” on a sentence' words working left to right, a finite verb, when it finds an unattached subject to the left, can safely assume this to be the correct subject daughter, while the inverse is not true - a subject may have to attach to a verb mother several verbs further to the right, in the case of embedded relative clauses. This fact is exploited in (d2), where reverse attachment is used to find the subject of a relative clause (@FS-@N<)[2].

[2]Note the BARRIER condition disallowing fronted arguments of prepositions (@>>P), which may indicate a subject-less clause as in *”et råstof, der skal kæles for”*.

3.3. Segment delimiters

In principle, semantic-syntactic dependency-relations could be established across a whole text. However, in practice, our corpus annotation employs a segmentation into sentences or verb-less utterances as part of the CG-annotation, and these segments will be respected as window limits for the dependency annotation, too. However, in preformatted corpora, as the Korpus90/2000, or the paragraph-divided Europarl corpus, more than one sentence may be present in one corpus chunk. To handle these cases, while at the same time allowing list annotation or utterance chaining where desired, the dependency grammar formalism provides for a list of delimiters (e1), which may be used to block unwanted attachment across sentence boundaries, as in rule (e2) which attaches statements (@STA) to verbless nominal (@NPHR) or adverbial (@ADVL) utterances.

(e1) ¤DELIMITER = \$\. , \$\! , \$\? , \$:, \$;

(e2) @STA -> (@NPHR,@ADVL) IF (L) BARRIER:(¤DELIMITER)

4. The rule compiler

4.1. Compiler principles

Sets and rules from the grammar are compiled and implemented on CG-input by a perl-program (cg2dep) that creates dependency links between tokens, the final output format being an additional tag in the list of word based CG tags, containing a number relation (e.g. #7->4, meaning token number 7 is a dependent of token number 4). In an actual run, word tokens are matched against targets from left to right, first in an ordered sequence of target batches, then once more, in simple token-driven order. Target ordering and iteration help to avoid or detect circular attachment, i.e. attachment hierarchies, where a daughter ultimately has itself as ancestor.

Though this program is, in principle, language and grammar-independent, it is not entirely theory-free, since it has to make certain assumptions/decisions about dependency grammar theory. First, more generally, it is assumed that each dependent has one and only one head, that dependencies can cross (non-projectivity), and that heads can be ”saturated” with certain types of dependents (clause function uniqueness principle, verb chains). Second, certain more theory-dependent descriptive issues had to be normalized, most important coordination, where both the coordinator and all following conjuncts were attached to the first conjunct. This solution has the advantage of being able to handle coordination without coordinators and of maintaining both the link between what is coordinated (sister-relation) and, through first-conjunct-inheritance, the semantically important link between mother and all coordinated daughters.

4.2. Coordination and tag insertion

Coordination is notoriously one of the more difficult tasks in syntactic annotation. To prevent over-generation in psg-grammars and spurious attachment i the dependency grammar, a special CG module marks coordinators for what they coordinate. These tags are then exploited to first attach coordinators to the correct (right hand) conjunct (f1-2), and subsequently, to make right hand conjuncts to look for left hand matches (f3), i.e. tokens of the same function, to the left of the flagged coordinator.

(f1) <co-acc> -> (@ACC>,@<ACC) IF (R)

(f2) <co-inf> -> (INF) IF (R)

(f3) <cjt>.* @FS-ADVL> -> (@FS-@ADVL>) IF (L)

The <cjt> tag is automatically added to a token, when it either receives a coordinator daughter or a dependency link to/from another token of the same function. Thus, the <cjt> tag in (f3) stems from rule (f2) and allows right pointing adverbial clauses to attach *left* to a conjunct-head rather than *right* to a verb. Left-pointing tokens may also find their conjunct head simply by *not* encountering a legitimate head to the left before another same-function token blocks the path. Here, the uniqueness principle will be used to only let the leftmost token attach to the (joint) head, and conjunct-attach all other (right-hand) conjuncts to the first, marking both the former and the latter with the <cjt> tag. Though this principle is hard-wired into the compiler program executing the dependency grammar, other ways of expressing conjunct dependencies can easily by achieved by post-filtering. Thus, we employ ”flat” attachment of conjuncts to a common head for semantic reasons in our machine translation application. Finally, <cjt> tags may be added in adirect, rule-governed way by using the ADD convention:

(g) @SUBJ> -> (PR,IMPF) IF (L) (ADD:<cjt>)

5. Evaluation

To evaluate the performance of the cg2dep compiler and the coverage of its grammar, a small random text sample was extracted from Korpus90[3], consisting of 1437 words (1663 tokens, 124 sentences) of news text. A gold-standard corrected annotation was built for both the CG and dependency

[3]Korpus90 is part of a Danish corpus compiled by DSL for lexicographical work (www.dsl.dk), and constitutes one half of the Korpus90/2000 project (www.korpus2000.dk and corp.hum.sdu.dk).

levels. In a complete run on raw text, where complete disambiguation was forced, the combination of DanGram and the cg2dep stage achieved 95% accuracy for edge labels, 99.4% for PoS, and 93% for dependency attachments.

1437 words 1663 tokens	*errors*	*accuracy (words, not tokens, out of all)*
Part of speech - on raw text	10	99.4 %
Syntactic function (edge label) - on raw text	73	95 %
Dependency (attachment) - on raw text	102	93 %
Dependency - on function-corrected input	20	98.7 %

If the dependency stage was run on CG input with corrected function labels, attachment accuracy was 98.7%. This error rate of 1.3% is a third lower than the difference in percentage points (2%) between edge label errors (95%) and attachment errors (93%) in the full run, indicating the importance of a good syntactic CG stage. The percentage of sentences without any attachment errors was 64.8% in a full run, 90.4% for cg-corrected input.

More generally, our results indicate that the depencency stage is somewhat better at building complete structures from cg-annotated input than VISL's traditional PSG-stages, while at the same time being about 30 times faster. Thus, the latter produced well-formed psg structures from cg-corrected input in 81.6 % of all sentences in the test-sample, while the former achieved 90.4 % complete *and* correct dependency structures. Obviously, ”well-formed” does not necessarily mean ”correct”, so in order to perform a direct comparison of dependencies, psg-output was converted into TIGER-format dependency trees, using VISL's various format filters (http://visl.sdu.dk/visl2/treebanks.html). The psg-derived dependency-trees for function-corrected input were complete in 93.6 % of sentences and correct in 75.1 %[4]. For raw text, the difference between the direct cg2dep method and the intermediate-psg approach was less marked, with 64.8 % and 58.4 % correct sentences, respectively. Individual attachments with the intermediate psg were correct in 86.2% for raw text, and in 92.3% for function-corrected cg-input, i.e. about half as good as in the direct approach (93% ad 98.7%, respectively).

[4] Individual attachments were correct in 86.2% for raw text, and in 92.3% for function-corrected cg-input.

percentage of sentences with:	*cg2dep*	*cg -> psg -> dep*	*cg -> psg*
complete structures *- from raw text*	88 %	89.6 %	72.8 %
complete structures *- on function-corrected cg*	96 %	93.6 %	81.6 %
complete & correct *- from raw text*	64.8 %	58.4 %	
complete & correct *- on function-corrected cg*	90.4 %	75.1 %	

6. Exchange formats

Bypassing the PSG- and TIGER-XML constituent formats, we have written two new format filter programs for VISL's token based dependency tag format, both maintaining full information-equivalence (http://beta.visl.sdu.dk/treebanks.html). One exports to the Malt-XML format (http://w3.msi.vxu.se/~nivre/research/MaltXML.html) by simply creating an xml-attribute structure around the VISL-tags. The other exports to TIGER-xml, as recommend by the Nordic Treebank Network (http://w3.msi.vxu.se/~nivre/research/nt.html), creating ”non-terminals” by re-writing heads as ”self-dependents”, and daughters as constituents, while using CG function labels as edge labels, and part of speech as category label. For both MALT and TIGER formats, the original CG function tag of second or later conjuncts is replaced by 'CJT'.

7. Outlook

The full dependency format is used for the dependency version of the Danish Arboretum treebank (now 423.656 tokens, 21.757 sentences). At the time of writing, automatic adding of full dependencies leads to circularity problems in about 1 percent of sentences, a problem which will have to be addressed either by improving the dependency grammar itself, or by linguistic revision of its output.

The numbered dependency format has also proven useful, in an ongoing Danish-English MT-project, for grammar based polysemy resolution and translation equivalent differentiation. Here, lexical transfer rules make use of head-, daughter-, sister- and higher level dependency relations in order to express context-dependent function- and semantic prototype class restrictions on possible target language translation equivalents.

Finally, a pilot study has shown that the numbered dependency format contains the necessary information for a filter program, without external grammatical rules, to create PSG-style constituent-tree structures from such data. A new round of evaluation should therefore add a comparison of direct and indirect[5] PSG-formats to the comparison of direct and indirect dependency-formats discussed in this paper.

References

Bick, Eckhard (2001). ”En Constraint Grammar Parser for Dansk”. In: Widell, Peter & Kunøe, Mette (ed.): *8. Møde om Udforskningen af Dansk Sprog.* Århus: Århus Universitet 2001.

Bick, Eckhard (2003-1), “Arboretum, a Hybrid Treebank for Danish”. In: Joakim Nivre & Erhard Hinrich (eds.), *Proceedings of TLT 2003 (2nd Workshop on Treebanks and Linguistic Theory, Växjö, November 14-15, 2003)*, pp.9-20. Växjö University Press

Bick, Eckhard (2003-2). “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

Tapanainen, Pasi (1999). *Parsing in two frameworks: finite-state and functional dependency grammar*. University of Helsinki, Deparment of General Linguistics

Tapanainen, Pasi and Timo Järvinen. (1997). ”A non-projective dependency parser”. In: *Proceedings of the 5th Conference on Applied Natural Language Processing*, pages 64–71, Washington, D.C., April. Association for Computational Linguistics.

[5] *Indirect* here means the use of an intermediate PSG stage in dependency annotation or the use of an intermediate (full) dependency stage in PSG annotation. Hower, both the direct and indirect methods depart, in our approach, from a CG stage expressing shallow dependency syntax.