Formalising the Swedish Constructicon in Grammatical Framework

Normunds Gruzitis[1] **Dana Dannélls**[2] **Benjamin Lyngfelt**[2] **Aarne Ranta**[1]
[1]Department of Computer Science and Engineering
[2]Department of Swedish
University of Gothenburg
[1]{name.surname}@cse.gu.se, [2]{name.surname}@svenska.gu.se

Abstract

This paper presents a semi-automatic approach to acquire a computational construction grammar from the semi-formal Swedish Constructicon. The implementation is based on the resource grammar library provided by Grammatical Framework and can be seen as an extension to the existing Swedish resource grammar. An important consequence of this work is that it generates feedback, explicit and implicit, on how to improve the annotation consistency and adequacy of the original construction resource.

1 Introduction

Constructicon is a collection of conventionalized pairings of form and meaning (or function), typically based on principles of Construction Grammar (Goldberg, 2013).

The formalisation and implementation of a wide coverage construction grammar is a highly relevant task. From the linguistic point of view, it leads to new insights on the interaction between the lexicon and the grammar, as well as it allows for testing the linguistic descriptions of constructions. From the language technology point of view, the account of constructions facilitates language processing in both monolingual and multilingual settings, e.g. in information extraction and machine translation.

Several approaches to Construction Grammar have been proposed. Remarkable examples include Sign-Based Construction Grammar (Boas and Sag, 2012) that uses Head-Driven Phrase Structure Grammar (Pollard and Sag, 1994) as the underlying formalism, Fluid Construction Grammar (Steels, 2013) and Embodied Construction Grammar (Bergen and Chang, 2013).

While the previous work has been mainly focused on English, our work is currently focused on Swedish. However, the main difference is that we test Grammatical Framework, GF (Ranta, 2004), as a formalism and a toolkit for implementing computational construction grammars. GF provides a built-in support for multilingual grammars, which has a great potential for implementing, unifying and interlinking constructions of different languages, which, in turn, would be particularly beneficial for the use in machine translation and second-language learning.

In this paper we describe a methodology on how to systematically formalise the semi-formal representation of the Swedish Constructicon in GF, showing that a GF construction grammar can be, to a large extent, acquired automatically. A side result of our work is that it has also helps to improve the original construction resource.

2 Background

2.1 Swedish Constructicon (SweCcn)

SweCcn[1] is a comparatively large open database of Swedish constructions – partially schematic multi-word units having both fixed and variable parts (Lyngfelt et al., 2012). It particularly addresses constructions of relevance for second-language learning, but also covers argument structure constructions, which concern matters of transitivity, voice, and event structure. Construction descriptions are manually derived from corpus examples, and some of the examples are manually annotated and added to each SweCcn entry. A simplified example of how a construction is described in SweCcn is given in Table 1.

Construction elements (CE) are either internal or external. The internal CEs are a part of the construction while the external CEs are a part of the valency of the construction. In the structure sketches, the internal CEs are bounded by brackets. CEs are described in more detail by attribute-

[1] http://spraakbanken.gu.se/eng/sweccn

Proceedings of the Grammar Engineering Across Frameworks (GEAF) Workshop, 53rd Annual Meeting of the ACL and 7th IJCNLP, pages 49–56,
Beijing, China, July 26-31, 2015. ©2015 Association for Computational Linguistics

Name	REFLEXIV_RESULTATIV
Category	VP
Frame	CAUSATION
Defintion	[Someone/something]$_{\text{NP}}$ performs/undergoes [an action]$_{\text{Activity}}$ that leads (or is supposed to lead) the [actor/theme]$_{\text{Pn}}$, expressed by reflexive, to [a state]$_{\text{Result}}$.
Structure	NP [V Pn$_{\texttt{refl}}$ AP]
Internal	Activity: {cat=V, role=Activity} Pn: {cat=Pn$_{\texttt{refl}}$, role=Actor|Theme} Result: {cat=AP, role=Result}
External	NP: {cat=NP, role=Actor|Theme}
Example	*Peter*$_{\text{NP}}$ [*äter*$_{\text{Activity}}$ *sig*$_{\text{Pn}}$ *mätt*$_{\text{Result}}$]

Table 1: A simplified description of the Swedish construction REFLEXIV_RESULTATIV. The example literally translates as 'Peter eats himself full'.

value matrices that specify their syntactic and semantic features.

Fixed CEs are represented by lexical units (LU), and they refer to entries in SALDO, the Swedish Associative Thesaurus (Borin et al., 2013), which is the core lexicon of a large macro-resource for Swedish, developed within the Swedish FrameNet++ project (Borin et al., 2010).

Many constructions have a referential meaning, more specifically, they are frame-bearing and are thus linked to FrameNet frames. There is also an ongoing work to link, when possible, the SweCcn constructions with constructions in Berkeley Constructicon (Bäckström et al., 2014) as well as other constructicons, notably the one for Brazilian Portuguese (Torrent et al., 2014).

It should be noted that a central part of construction descriptions in SweCcn is the free text definitions. For example, the construction REFLEXIV_RESULTATIV roughly means 'become AP by V-ing'. Hence, *äta sig mätt* 'eat himself full' and *skrika sig hes* 'shouting himself hoarse' are instances of the construction, whereas *känna sig trött* 'feel himself tired' and *skratta sig lycklig* 'laugh himself lucky' are not. The difference is captured by the free text definition, but not by the formal features, therefore it unfortunately gets lost in the automatic translation to GF.

In this experiment, we use a recent version of SweCcn (a snapshot taken on June 9, 2015) which contains 374 entries describing constructions of different grammatical categories such as VP, NP and S (see Table 2).

Category	Total	Ratio	FrameNet
VP	105	28%	77
NP	85	23%	54
S	77	21%	50
PP	26	7%	22
AdvP	23	6%	19
XP	16	4%	4
AP	14	4%	13
other	28	7%	19

Table 2: The number of constructions in SweCcn. The category XP represents any phrase type. The column FrameNet shows the number of constructions linked to the Swedish FrameNet.

2.2 Grammatical Framework (GF)

GF (Ranta, 2004) is a grammar formalism characterized by its two-level approach to natural language representation. One level, the abstract syntax, accounts for the language-independent aspects, and the other level, the concrete syntax, accounts for the language-specific aspects. The same abstract syntax can be equipped with many concrete syntaxes – reversible mappings from abstract syntax trees to records (feature structures) and strings – making the grammar multilingual.

Most importantly, GF provides a general-purpose resource grammar library, RGL (Ranta, 2009), for currently 30 languages, all implementing the same abstract syntax.

In order to hide the low-level details, RGL has a high-level interface that provides constructors like `mkCl: NP -> VP -> Cl` for building a clause from a NP and a VP.[2] The resource grammars take care of agreement and word order.

One of the most developed languages in RGL, in terms of syntactic and lexical coverage, is Swedish. Its resource grammar also includes over 100,000 lexical entries from SALDO.[3]

3 Preprocessing of SweCcn

In the current experiment, we consider only the 105 constructions of type VP (verb phrase) from which we exclude 9 whose status is 'suggestion'. Descriptions of the suggested constructions are too immature to be processed. Currently we also

[2] http://www.grammaticalframework.org/lib/doc/synopsis.html

[3] http://www.grammaticalframework.org/lib/src/swedish/DictSwe.gf

do not include the 16 XP constructions which are relevant to any phrase type, including VP.

We have chosen to begin with VP constructions because they are dominating in SweCcn, and they have the most complex internal structure – if our approach can handle these constructions then it should also be applicable for the rest.

According to the SweCcn annotation manual,[4] constructions are described at two levels of detail:

1. A flat structure sketch that lists the formal elements in the construction (see *Structure* in Table 1). Each CE is represented in terms of grammatical category (either word class or phrase type), LU or just word form. The list of CEs follows the expected word order. A structure sketch may specify alternative realisation patterns of the same construction.

2. A set of feature matrices, one per CE (see *Internal* and *External* in Table 1), that specify additional morphosyntactic constraints which may be omitted in the more general sketch for the sake of simplicity to a human reader. Additionally, the feature matrices often specify the semantic roles and grammatical functions, but we do not take this information into account in the current work.
   The word order is encoded only by the structure sketches; it is not reflected by the corresponding feature matrices as they can be potentially reused by alternative patterns of the same construction. Because the linking between the sketches and matrices is not explicit, and the implicit links (matching categories, LUs etc.) are not unique in general, the automatic mapping can be ambiguous. In practice, however, it happens rarely.

Constructions may have optional CEs, alternative types of CEs or alternative LUs, and even alternative word order. In the structure sketches, optional CEs are delimited by parentheses, and alternative types/LUs are separated by a bar:

`[V av`^{1} `Pn`$_{\texttt{refl}}$ `(NP)]`

`[behöva`^{1} `NP`$_{1}$ `till`^{1} `NP`$_{2}$`|VP]`

`[snacka`^{1}`|prata`^{1}`|tala`^{1} `NP`$_{\texttt{indef}}$`]`

`[N|Adj+städa`^{1}`]`

[4] https://svn.spraakdata.gu.se/sb/fnplusplus/pub/constructicon/manual/sweccnmanual.pdf

Note that the variable CEs (represented by grammatical categories) may have indices denoting difference, formal identity (repetition), coreference, etc. In the case of a lexical construction that represents a compound word, its internal CEs are delimited by the plus sign indicating the concatenation. Suffixation is indicated by the hyphen.

The automatic preprocessing of SweCcn entries consists of four steps:

1. Normalization of the structure sketches and attribute values in the feature matrices. SweCcn entries have been annotated manually, therefore inconsistently used spaces, inconsistently used delimiters of alternative CE types as well as inconsistent representation of auxiliary or function CEs (e.g. sig^{1} vs. Pn_{refl} vs. *refl*) is common.

2. In case of optional CEs and alternative types of CEs, there are formally several constructions compressed in one. The original structures are rewritten so that for each combination there is a separate alternative structure. For instance, `[V av`^{1} `Pn`$_{\texttt{refl}}$ `(NP)]` is rewritten to `[V av`^{1} `Pn`$_{\texttt{refl}}$ `NP] | [V av`^{1} `Pn`$_{\texttt{refl}}$`]`.
   This however does not apply to alternative LUs. If a CE is represented by a fixed set of LUs, we assume that they are interchangeable (synonymous). Otherwise they should be either split into alternative constructions (separate entries), or the CE should be made more general.[5]

3. The rewritten structure sketches are enriched with additional morphosyntactic information from the feature matrices, so that a complete description is at hand. The mapping of CEs between the two layers of annotation is based on values of the grammatical category and LU attributes in the feature matrices (see Table 1). Although such mapping in general is based on a partial comparison as well as it can be ambiguous, it has not led to incorrect results in the selected dataset,[6] because we do not consider the semantic roles.

[5] If a list of non-interchangeable but frame-evoking LUs is replaced by a general grammatical category, the set of possible target words is still implicitly restricted by the FrameNet frame which is evoked by the whole construction.

[6] Provided that the specifications are consistent across the two layers.

4. The grammatical categories used in SweCcn are converted to GF RGL categories. In specific cases, the conversion may lead into a more general or more specific description as well as it may include the morphosyntactic tags and may depend on CEs in the context. For instance, categories Adv, AdvP and PP are all generalized to Adv while the specification NP_{indef} is elaborated in three alternative substructures: `[aSg_Det CN] | [aPl_Det CN] | CN`, where *aSg_Det* is a function representing the indefinite article and requiring the singular agreement, *aPl_Det* requires the plural agreement, and CN is a category that represents common nouns (including modifiers, except determiners). This requires a subsequent rewriting of the whole construction as described in Step 2. Few categories are not converted at this step; their conversion is postponed to the generation of the GF grammar. For instance, Pc (participle) and PcP (participle phrase) are not converted to V and VP respectively, as they have to be treated differently in the concrete syntax: PcP is a VP that is further converted to AP or Adv as illustrated by FÅ_RESULTATIV.AGENTIV in Sections 4.1 and 4.2.

Out of the 96 VP constructions that were processed, only 43 turned out to be consistent in the first attempt. For more than a half of constructions, various inconsistencies were detected and reported to SweCcn developers for manual inspection and correction. After several iterations, the number of consistent VP constructions increased to 93. The remaining 3 are different corner cases that are actually consistent but are not yet supported by the preprocessor and are thus skipped.

The following is a list of representative VP constructions with their original and rewritten structure descriptions that we use in Section 4 to illustrate the automatic generation of the GF grammar:

BEHÖVA_NÅGOT_TILL_NÅGOT:
behöva mat till festen 'need food to the party'
$\texttt{behöva}^1\ \texttt{NP}_1\ \texttt{till}^1\ \texttt{NP}_2|\texttt{VP} \rightarrow$
$\texttt{behöva}_\texttt{V}\ \texttt{NP}_1\ \texttt{till}_\texttt{Prep}\ \texttt{NP}_2$
$|\ \texttt{behöva}_\texttt{V}\ \texttt{NP}\ \texttt{till}_\texttt{Prep}\ \texttt{VP}$

FÅ_RESULTATIV.AGENTIV:
få gräsmattan klippt 'get the lawn trimmed'
$\texttt{få}^0\ \texttt{NP}\ \texttt{PcP} \rightarrow \texttt{få}_\texttt{V}\ \texttt{NP}\ \texttt{PcP}_\texttt{perf}$

GÖRA_SIG_ADVP:
gör sig bra 'does himself well'
$\texttt{göra}^1\ \texttt{Pn}_\texttt{refl}\ \texttt{AdvP} \rightarrow \texttt{göra}_\texttt{V}\ \texttt{refl}_\texttt{Pron}\ \texttt{Adv}$

SNACKA_NP:
prata skolminnen 'talk school memories'
$\texttt{snacka}^1|\texttt{prata}^1|\texttt{tala}^1\ \texttt{NP}_\texttt{indef} \rightarrow$
$\texttt{snacka}|\texttt{prata}|\texttt{tala}_\texttt{V}\ \texttt{aSg\_Det}\ \texttt{CN}$
$|\ \texttt{snacka}|\texttt{prata}|\texttt{tala}_\texttt{V}\ \texttt{aPl\_Det}\ \texttt{CN}$
$|\ \texttt{snacka}|\texttt{prata}|\texttt{tala}_\texttt{V}\ \texttt{CN}$

VERBA_AV_SIG.TRANSITIV:
ta av mig skorna 'take off myself shoes'
$\texttt{V}\ \texttt{av}^1\ \texttt{Pn}_\texttt{refl}\ \texttt{(NP)} \rightarrow$
$\texttt{V}\ \texttt{av}_\texttt{Prep}\ \texttt{refl}_\texttt{Pron}\ \texttt{NP}\ |\ \texttt{V}\ \texttt{av}_\texttt{Prep}\ \texttt{refl}_\texttt{Pron}$

X-STÄDA:
storstäda 'bigclean'
$\texttt{N}|\texttt{Adj+städa}^1 \rightarrow \texttt{N + städa}_\texttt{V}\ |\ \texttt{A + städa}_\texttt{V}$

Note that we ignore the SALDO sense identifiers. We ignore the external CEs in the current approach as well, as they should be attached to constructions by the general syntactic rules already provided by GF RGL. It is satisfactory also from the future translation point of view, as the translation of external CEs should be compositional.

4 Generation of a GF Grammar

The rewritten structural descriptions of constructions, as described in Section 3, provide sufficient information to generate both the abstract and the concrete syntax of a SweCcn-based construction grammar, an extension to the Swedish GF resource grammar.[7]

4.1 Abstract Syntax

The generation of the abstract syntax is rather straight forward. Each construction is represented by one or more functions depending on how many alternative structure descriptions are produced in the preprocessing steps. The name of a function corresponds to the name of the construction suffixed by an index if there is more than one function per construction. For the current input data, the 93 VP constructions resulted in 127 functions. The maximum and average numbers are respectively 6 and 1.4 functions per construction.[8]

[7] https://github.com/GrammaticalFramework/gf-contrib/tree/master/SweCcn

[8] The max number is produced by SNACKA_NP.EMFAS: $[\texttt{snacka}^1|\texttt{prata}^1\ \texttt{(AP)}\ \texttt{NP}_\texttt{indef}]$.

Each function takes one or more arguments that correspond to the variable CEs of the respective alternative construction description. In the rewritten structure descriptions, the variable CEs can be formally distinguished from fixed CEs (LUs and structural words) by the first letter of each CE: the variable CEs always start with an upper case letter while the fixed CEs start with a lower case letter. The fixed CEs are not represented by the abstract syntax. The variable CEs are represented only by their grammatical categories; other morphosyntactic constraints (if any) are handled by the concrete syntax.

Constructions listed at the end of Section 3 are represented by the following abstract functions:

```
behöva_något_till_något1: NP -> NP -> VP
behöva_något_till_något2: NP -> VP -> VP

få_resultativ_agentiv: NP -> VP -> VP

göra_sig_AdvP: Adv -> VP

snacka_NP1: CN -> VP
snacka_NP2: CN -> VP
snacka_NP3: CN -> VP

verba_av_sig_transitiv1: V -> NP -> VP
verba_av_sig_transitiv2: V -> VP

x_städa1: N -> VP
x_städa2: A -> VP
```

4.2 Concrete Syntax

As our initial investigation unveiled, many constructions can be implemented in GF by systematically applying the high-level RGL constructors. For instance, `behöva_något_till_något1` can be implemented as shown in Figure 1 by first making a two-place verb (V2) from the V element and then combining it with the first NP element into a VP. The preposition can be combined with the second NP element into a prepositional phrase (Adv) which can then be attached to the VP. The question is how to make such constructor applications systematically given the various construction descriptions.

Essentially, this is a parsing problem itself. We can look at CEs as words in the construction description language for which we need a grammar to combine the lists of CEs into trees of RGL constructors and their arguments.

In order to address this issue, we have defined an auxiliary GF grammar to generate the

```
behöva_något_till_något1 np1 np2 =
 mkVP
  (mkVP (mkV2 (mkV "behöver")) np1)
  (mkAdv (mkPrep "till") np2)
```

Figure 1: The expected implementation for the function `behöva_något_till_något1`.

implementation of functions in the GF construction grammar. To keep the code-generating grammar simple, it accepts only the categories of CEs, some additional constraints and certain structural words. The preprocessed construction descriptions are generalized before parsing; LUs are inserted back in a post-processing step. For instance, `behöva`$_{\texttt{V}}$ `NP1 till`$_{\texttt{Prep}}$ `NP2` is generalised to {V} NP {Prep} NP, where the curly brackets indicate fixed CEs. Fragments of the code-generating grammar related to this structure are listed in Figure 2 and Figure 3.

```
fun mkV2: V -> V2

fun mkVP__V2_NP: V2 -> NP -> VP
fun mkVP__VP_Adv: VP -> Adv -> VP

fun mkAdv: Prep -> NP -> Adv

fun _mkV_: V

fun _mkPrep_: Prep

fun _NP_: NP
```

Figure 2: A simplified fragment of the abstract syntax of the auxiliary code-generating grammar.

According to the auxiliary grammar, the parse tree for "{V} NP {Prep} NP" is

```
mkVP__VP_Adv
  (mkVP__V2_NP (mkV2 _mkV_) _NP_)
  (mkAdv _mkPrep_ _NP_)
```

which corresponds to the expected implementation as shown in Figure 1 after the post-processing is done. The post-processing involves three steps:

1. Remove all suffixes delimited by the double underscore. The suffixes are used just to make the function names unique in the auxiliary grammar.

2. Sequentially replace all placeholders of the fixed CEs, annotated as *_mkX_*, by the actual lexical constructors. In case of verbs, constructors (inflectional paradigms) specified in

```
param Voice = Act | Pass

lincat
  V, V2 = Voice => Str
  VP, NP, Adv, Prep = Str

lin
  mkV2 v = \\voice => v ! voice

  mkVP__V2_NP v2 np = v2 ! Act ++ np
  mkVP__VP_Adv vp adv = vp ++ adv

  mkAdv prep np = prep ++ np

  _mkV_ = table {
    Act => "{V}"
    Pass => "{V_pass}"
  }

  _mkPrep_ = "{Prep}"

  _NP_ = "NP"
```

Figure 3: A simplified fragment of the concrete syntax of the auxiliary code-generating grammar.

the GF implementation of SALDO (see Section 2.2) are reused.

3. Sequentially replace all placeholders of the variable CEs, annotated as *_X_*, by the actual variable names, e.g. replace the first *_NP_* by *np1* and the second *_NP_* by *np2*.

Note that the auxiliary code-generating grammar, in general, is ambiguous – it can return several alternative code skeletons for a given CE list. However, it should hold that all alternatives accept and linearise the same strings. Our heuristics is to take the shortest implementation, which is supported by the intuition that the shortest ones correlate with the simplest ones.

If we consider the alternative realization of BEHÖVA_NÅGOT_TILL_NÅGOT represented by the function `behöva_något_till_något2` , the parsing with the auxiliary grammar fails at the element *VP*. Indeed, there is no straightforward constructor provided by RGL that would combine a Prep with a VP or an Adv (as the *in-order-to*-VP should be first converted to Adv). Thus, a lower level means have to be applied to implement this function.

The implementation generated for the rest of functions listed in Section 4.1 is given below (in a slightly simplified form):

```
få_resultativ_agentiv np vp = mkVP
  (mkV2A (mkV "få"))
  np (PresPartAP vp)

göra_sig_AdvP adv = mkVP
  (mkVP (reflV (mkV "göra"))) adv

snacka_NP1 cn = mkVP
  (mkV2 (mkV ("snacka"|"prata"|..)))
  (mkNP aSg_Det cn)

snacka_NP2 cn = mkVP
  (mkV2 (mkV ("snacka"|"prata"|..)))
  (mkNP aPl_Det cn)

snacka_NP3 cn = mkVP
  (mkV2 (mkV ("snacka"|"prata"|..)))
  (mkNP cn)

verba_av_sig_transitiv1 v np = mkVP
  (mkV2 (reflV
    (partV v (toStr (mkPrep "av")))))
  np

verba_av_sig_transitiv2 v = mkVP
  (reflV
    (partV v (toStr (mkPrep "av"))))

x_städa1 n = mkVP
  (prefixV (toStr n) (mkV "städar"))

x_städa2 a = mkVP
  (prefixV (toStr a) (mkV "städar"))
```

As it was already mentioned, for some functions the implementation has to be based not only on the high-level language-independent interface of RGL but also on low-level language-specific parameters. To keep the GF code generation flexible and functional, we have defined some helper functions (in the construction grammar) that wrap the low-level code and make it reusable. For instance, the helper function *toStr* takes a preposition, adjective or noun and returns its base form as a plain string which can then be passed, for instance, to the RGL function *partV* to make a particle verb, or to another helper function *prefixV* to make a compound verb.

As for LUs, note that they are implemented, in general, as free alternatives, which means that any of them will be accepted while parsing but the first one will always be used for the linearisation.

In the result, given the 127 functions in the abstract syntax, we have automatically generated the implementation for 98 functions (77%). At least one function is implemented for 73 out of 93 constructions (78%).

5 Analysis of the Initial Results

We conducted two evaluations, manual and automatic, to determine whether the automatically implemented functions can successfully parse the respective Swedish constructions and whether they

	Functions	Examples	Exemplified functions
Implemented	51	57	24
Pending	13	16	6
Total	64	73	30

Table 3: Statistics of the manually compiled test corpus: the number of examples belonging to the implemented and pending concrete functions in the generated construction grammar, and the number of functions having at least one test example.

	Functions	Examples	Exemplified functions
Implemented	98	224	65
Pending	29	40	11
Total	127	264	76

Table 4: Statistics of the automatically acquired test corpus. Compare to Table 3.

can cope with different linguistic phenomena. The manual evaluation was based on a subset of selected VP constructions and selected examples from the annotated sentences in SweCcn. The automatic evaluation was based on the whole SweCcn dataset of all VP constructions.

For the manual evaluation, we complied a small test corpus containing 73 annotated examples, of which 57 turned out to have a corresponding concrete function in the construction grammar. Table 3 summarizes the total number of examples that belong to any of the implemented functions and the total number of examples that belong to the functions whose implementation is pending, as well as the number of functions that have at least one test example. In the manually compiled corpus, only about half of the functions have at least one test example, and for those that have, there are two examples on average.

Out of the 57 examples that have a corresponding concrete function, 53 examples were successfully parsed yielding a coverage of 93%. It is important to mention that the relatively high coverage is achieved partially because we replaced all the compounds and proper names which were missing in the lexicon (17 words in total). The remaining 7% are examples for which no parse tree was returned. A closer look at those cases unveils that the parser mostly failed because of: (i) annotation errors in the SweCcn database, for instance, a feature matrix constrains the singular form of a NP although the plural form exists among the annotated examples; (ii) ill-formed sentences (with respect to the grammar), often containing coordinating conjunctions, for instance, *jag och min sambo ska till våra vänner* 'me and my partner shall to our friends' – the parser expects a verb such as *gå* 'go' after *ska* 'shall'.

Errors grounded in the manual annotation of the SweCcn entries were reported to SweCcn developers and are already partially corrected. Errors grounded in the automatic grammar generation require a closer analysis of how these constructions can be systematically implemented using lower level means of RGL.

For the automatic evaluation, we implemented a script which pre-processes the annotated SweCcn sentences belonging to the VP constructions and parses each example using the generated GF grammar. Several heuristics on how to insert the subject to make a proper clause before it is parsed are applied. Heuristics mainly concern the tense and type of the verb given a construction with which it should be parsed. Table 4 summarizes the automatically acquired test corpus.

Out of the 224 examples for which the corresponding concrete function is implemented, 157 were successfully parsed, yielding a coverage of 70%. An investigation of the examples that failed to parse unveils that these examples: (i) contain multi-word compounds; (ii) are more than 10 words long, containing irrelevant phrases and punctuations that fall outside the construction; (iii) contain complex syntactic structures that involve coordination and subordination.

Our analysis shows that many of the failures lead to false negative evaluation results. To avoid these and to allow for a more adequate evaluation, there are several complementary options we have to consider. First, the grammatical categories could be included in the annotated examples, but it depends on the SweCcn developers. Second, we could prepare a treebank, at least one abstract tree for each function, to allow for the opposite testing – to check if the functions generate correct linearizations. Third, we could manually derive a larger post-edited test corpus from the SweCcn dataset of annotated examples. For functions having no test example, we might exploit the GF's built-in support for generating random trees. The linearizations could then be presented

to SweCcn developers for examination and consideration of whether an example should be added to the database.

When it comes to the lexicon, the coverage of lexical units is very high. Most of the words the parser fails with are proper names and compounds. These could be extracted from the SweCcn corpus and added to the lexicon if access to the grammatical categories is available.

6 Conclusions and Future Work

We have taken a functional view to acquire a computational construction grammar in Grammatical Framework from the semi-formal representation of the Swedish Constructicon. We have presented an approach to detect and correct inconsistencies and errors in the original resource of constructions. We were able to improve the quality of the resource and thereby increase its value for the use in language technology applications.

Following the proposed approach, the implementation of a construction grammar can be automatically generated for nearly 80% of the constructions (functions) achieving a 70–90% accuracy, and there is clear space for improvement. However, it is still an open question how far we should advance the automation in order to keep it cost effective; the rest can be implemented or post-edited manually. So far we have avoided any manual intervention in the generated grammar because SweCcn is being actively improved and extended in parallel to our work, and this would complicate the synchronisation of changes.

Regarding future work, a rather short-term goal is to extend the grammar generator to cover the other major types of constructions as well. This would primarily require the extension of the auxiliary code generating grammar. Among the long-term goals is to take this approach from the monolingual construction grammar to a multilingual one. This would require not only taking the links to FrameNet into account but also adapting the processing and generation pipeline to the constructicons of other languages. This also relates to our previous research on implementing a multilingual FrameNet-based grammar in GF (Dannélls and Gruzitis, 2014). The GF construction grammar and FrameNet grammar approaches are complementary to each other, at least with regard to constructions with a referential meaning, and an integration of them would be mutually beneficial.

Acknowledgements

This work was supported by Swedish Research Council under Grant No. 2012-5746 (Reliable Multilingual Digital Communication) and by the Centre for Language Technology in Gothenburg.

References

Linnéa Bäckström, Benjamin Lyngfelt, and Emma Sköldberg. 2014. Towards interlingual constructicography. On correspondence between constructicon resources for English and Swedish. *Frames, constructions and computation. Special issue of Constructions and Frames*, 6(1).

Benjamin K. Bergen and Nancy Chang. 2013. Embodied Construction Grammar. In *The Oxford Handbook of Construction Grammar*.

Hans C. Boas and Ivan A. Sag, editors. 2012. *Sign-based Construction Grammar*. CSLI Publications.

Lars Borin, Dana Dannélls, Markus Forsberg, Maria Toporowska Gronostaj, and Dimitrios Kokkinakis. 2010. The past meets the present in Swedish FrameNet++. In *Proceedings of EURALEX*.

Lars Borin, Markus Forsberg, and Lennart Lönngren. 2013. SALDO: a touch of yin to WordNet's yang. *Language Resources and Evaluation*, 47(4).

Dana Dannélls and Normunds Gruzitis. 2014. Extracting a bilingual semantic grammar from FrameNet-annotated corpora. In *Proceedings of LREC*.

Adele E. Goldberg. 2013. Constructionist approaches. In *The Oxford Handbook of Construction Grammar*.

Benjamin Lyngfelt, Lars Borin, Markus Forsberg, Julia Prentice, Rudolf Rydstedt, Emma Sköldberg, and Sofia Tingsell. 2012. Adding a constructicon to the Swedish resource network of Språkbanken. In *Proceedings of KONVENS*.

Carl Pollard and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. University of Chicago Press.

Aarne Ranta. 2004. Grammatical Framework, a type-theoretical grammar formalism. *Journal of Functional Programming*, 14(2).

Aarne Ranta. 2009. The GF Resource Grammar Library. *LiLT*, 2(2).

Luc Steels. 2013. Fluid Construction Grammar. In *The Oxford Handbook of Construction Grammar*.

Tiago Timponi Torrent, Ludmila Meireles Lage, Thais Fernandes Sampaio, Tatiane da Silva Tavares, and Ely Edison da Silva Matos. 2014. Revisiting border conflicts between FrameNet and Construction Grammar. *Constructions and Frames*, 6(1).