Formalising the Swedish Constructicon in Grammatical Framework

Normunds Grūzītis$^{1,3}$, Dana Dannélls$^2$, Benjamin Lyngfelt$^2$, Aarne Ranta$^1$

$^1$University of Gothenburg, Department of Computer Science and Engineering
$^2$University of Gothenburg, Department of Swedish
$^3$University of Latvia, Institute of Mathematics and Computer Science

ACL/IJCNLP Workshop on Grammar Engineering Across Frameworks
Beijing, China, July 30, 2015
Constructicon

• A collection of conventionalized (learned) pairings of form and meaning (or function), typically based on principles of Construction Grammar, CxG (e.g. Fillmore et al. 1988, Goldberg 1995)
  – Semantics is associated directly with the surface form
  – vs. Lexical units in a dictionary: pairings of word and meaning (frame)
    • Including fixed multi-word units

• Each construction (cx) contains at least one variable element
  – Often at least one fixed element as well
  – Thus, “somewhere” in-between the syntax and the lexicon

• An example from Berkeley Constructicon: “make one’s way”
  – Structure: \{\text{Motion verb} [\text{Verb}] [\text{PossNP}]\}
  – Frame: MOTION
    • [\text{Theme } They] \{\text{hacked their way} [\text{Source out} [\text{Goal into the open}].
    • [\text{Theme } We] \{\text{sang our way} [\text{Path across Europe}].
Constructicons

- Berkeley Constructicon (BCxn) for **English**
  - A pilot project (around 70 cx), linked to Berkeley FrameNet
- **Swedish** Constructicon (SweCcn)
  - An ongoing project (nearly 400 cx so far), partially linked to FrameNet
    - ToDo: links to BCxn
- **Brazilian Portuguese** Constructicon
  - An ongoing project
- ...

- A multilingual (interlingual) constructicon would allow for **non-compositional** translation in a **compositional** way
  - *Constructions with a referential meaning may be linked via FrameNet frames, while those with a more abstract grammatical function may be related in terms of their grammatical properties*
Jag behöver mat till festen.

I need food to the party.
SweCcn

- Partially schematic multi-word units/expressions
- Particularly addresses constructions of relevance for second-language learning, but also covers argument structure constructions
- Descriptions are manually derived from corpus examples

Construction elements (CE):
- Internal CEs are a part of the cx
- External CEs are a part of the valency of the cx
- Described in more detail by attribute-value matrices specifying their syntactic and semantic features

- A central part of cx descriptions is the free text definitions
  - ‘eat himself full’ vs. ‘feel himself tired’
    (äta sig mätt vs. känna sig trött)

<table>
<thead>
<tr>
<th>Name</th>
<th>REFLEXIV_RESULTATIV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category</td>
<td>VP</td>
</tr>
<tr>
<td>Frame</td>
<td>CAUSATION</td>
</tr>
<tr>
<td>Definition</td>
<td>[Someone/something]<em>{NP} performs/undergoes [an action]</em>{Activity} that leads (or is supposed to lead) the [actor/theme]<em>{Pn}, expressed by reflexive, to [a state]</em>{Result}.</td>
</tr>
<tr>
<td>Structure</td>
<td>NP [V Pn_{refl} AP]</td>
</tr>
<tr>
<td>Internal Activity</td>
<td>{cat=V, role=Activity}</td>
</tr>
<tr>
<td>Pn:</td>
<td>{cat=Pn_{refl}, role=Actor</td>
</tr>
<tr>
<td>Result</td>
<td>{cat=AP, role=Result}</td>
</tr>
<tr>
<td>External NP</td>
<td>{cat=NP, role=Actor</td>
</tr>
<tr>
<td>Example</td>
<td>Peter_{NP} [ätter Activity sig Pn_{mätt} Result]</td>
</tr>
</tbody>
</table>
SweCcn → GF

• Task: convert the semi-formal SweCcn into a computational CxG
  – Test Grammatical Framework (GF) as a framework for implementing CxG

• Why GF?
  – There is no formal distinction between lexical and syntactic functions in GF – fits the nature of constructicons
  – The potential support for multilinguality
  – Based on GF Resource Grammar Library (RGL) / an extension to RGL
  – An extension to a FrameNet-based grammar and lexicon in GF

• Goals:
  – From the linguistic point of view
    • Improve insights into the interaction between the lexicon and the grammar
    • Allow for testing the linguistic descriptions of constructions
  – From the language technology point of view:
    • Facilitate the language processing in both mono- and multilingual settings
      – e.g. Information Extraction, Machine Translation
Conversion steps

• **Preprocessing:**
  – **Automatic** normalization and **consistency** checking
  – **Automatic** rewriting of the original structures in case of optional CEs and alternative types of CEs, so that each combination has a separate GF function
    • Does not apply to alternative LUs (either free variants or should be split into alternative constructions, or the CE should be made more general)
  – **Automatic** conversion of SweCcn **categories** to RGL categories
    • May result in more rewriting

• **Automatic** generation of the **abstract** syntax

• **Automatic** generation of the **concrete** syntax
  – By systematically applying the high-level RGL constructors
    • And limited low-level means

• **Manual** verification and completion (ToDo)
  – Requires a good knowledge and linguistic intuition of the language
Preprocessing examples

• *behöva NP₁ till NP₂*|VP →
  \( behöva_v \) NP₁ *till*{\text{Prep}} NP₂ | \( behöva_v \) NP *till*{\text{Prep}} VP

• *snacka|prata|tala* NP_{\text{indef}} →
  \( snacka_v|prata_v|tala_v \) aSg\_Det CN |
  \( snacka_v|prata_v|tala_v \) aPl\_Det CN |
  \( snacka_v|prata_v|tala_v \) CN

• *V av Pn\_refl (NP)* →
  \( V av_{\text{Prep}} refl_{\text{Pron}} \) NP | \( V av_{\text{Prep}} refl_{\text{Pron}} \)

• *N|Adj+städa* →
  \( N + städa_v \) | \( A + städa_v \)
Abstract syntax

• Each **construction** is represented by **one or more functions** depending on how many **alternative structures** are produced in the preprocessing steps.

• Each **function** takes **one or more arguments** that correspond to the **variable CEs** of the respective alternative construction.

• `behöva_något_till_något_VP_1` : NP -> NP -> VP
  `behöva_något_till_något_VP_2` : NP -> VP -> VP

• `snacka_NP_1` : CN -> VP
  `snacka_NP_2` : CN -> VP
  `snacka_NP_3` : CN -> VP

• `verba_av_sig_transitiv_1` : V -> NP -> VP
  `verba_av_sig_transitiv_2` : V -> VP

• `x_städa_1` : N -> VP
  `x_städa_2` : A -> VP
Concrete syntax

• Many constructions can be implemented by systematically applying the high-level RGL constructors
  – A parsing problem: which constructors in which order?

<table>
<thead>
<tr>
<th>Construction</th>
<th>Elements</th>
<th>Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>behöva_något_till_något_VP_1</td>
<td>behöva_V NP_1 till_Prep NP_2</td>
<td>{V} NP {Prep} NP</td>
</tr>
<tr>
<td>behöva_något_till_något_VP_2</td>
<td>behöva_V NP_1 till_Prep VP</td>
<td>{V} NP {Prep} VP</td>
</tr>
</tbody>
</table>

Code template

1. mkVP (mkVP (mkV2 mkV) NP) (mkAdv mkPrep NP)
2. The parser failed at token VP

Final code (by automatic post-processing)

```
lin behöva_något_till_något_VP_1 np_1 np_2 = mkVP
  (mkVP (mkV2 (mkV "behöver") np_1)
   (SyntaxSwe.mkAdv (mkPrep "till") np_2));
```
<table>
<thead>
<tr>
<th>Function</th>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>mkVP</td>
<td>V -&gt; VP</td>
<td>to sleep</td>
</tr>
<tr>
<td>mkVP</td>
<td>V2 -&gt; NP -&gt; VP</td>
<td>to love him</td>
</tr>
<tr>
<td>mkVP</td>
<td>V3 -&gt; NP -&gt; NP -&gt; VP</td>
<td>to send it to him</td>
</tr>
<tr>
<td>mkVP</td>
<td>VV -&gt; VP -&gt; VP</td>
<td>to want to sleep</td>
</tr>
<tr>
<td>mkVP</td>
<td>VS -&gt; S -&gt; VP</td>
<td>to know that she sleeps</td>
</tr>
<tr>
<td>mkVP</td>
<td>VQ -&gt; QS -&gt; VP</td>
<td>to wonder who sleeps</td>
</tr>
<tr>
<td>mkVP</td>
<td>VA -&gt; AP -&gt; VP</td>
<td>to become red</td>
</tr>
<tr>
<td>mkVP</td>
<td>V2A -&gt; NP -&gt; AP -&gt; VP</td>
<td>to paint it red</td>
</tr>
<tr>
<td>mkVP</td>
<td>V2Q -&gt; NP -&gt; QS -&gt; VP</td>
<td>to ask him who sleeps</td>
</tr>
<tr>
<td>mkVP</td>
<td>V2V -&gt; NP -&gt; VP -&gt; VP</td>
<td>to beg him to sleep</td>
</tr>
<tr>
<td>mkVP</td>
<td>A -&gt; VP</td>
<td>to be old</td>
</tr>
<tr>
<td>mkVP</td>
<td>A -&gt; NP -&gt; VP</td>
<td>to be older than he</td>
</tr>
<tr>
<td>mkVP</td>
<td>A2 -&gt; NP -&gt; VP</td>
<td>to be married to him</td>
</tr>
<tr>
<td>mkVP</td>
<td>AP -&gt; VP</td>
<td>to be very old</td>
</tr>
<tr>
<td>mkVP</td>
<td>N -&gt; VP</td>
<td>to be a ...</td>
</tr>
<tr>
<td>mkVP</td>
<td>CN -&gt; VP</td>
<td>to be a ...</td>
</tr>
<tr>
<td>mkVP</td>
<td>NP -&gt; VP</td>
<td>to be the woman</td>
</tr>
<tr>
<td>mkVP</td>
<td>Adv -&gt; VP</td>
<td>to be here</td>
</tr>
<tr>
<td>mkVP</td>
<td>VP -&gt; Adv -&gt; VP</td>
<td>to sleep here</td>
</tr>
<tr>
<td>mkVP</td>
<td>Adv -&gt; VP -&gt; VP</td>
<td>to always sleep</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>mkNP</td>
<td>Quant -&gt; N -&gt; NP</td>
<td>this man</td>
</tr>
<tr>
<td>mkNP</td>
<td>Quant -&gt; CN -&gt; NP</td>
<td>this old man</td>
</tr>
<tr>
<td>mkNP</td>
<td>Quant -&gt; Num -&gt; CN -&gt; NP</td>
<td>these five old men</td>
</tr>
<tr>
<td>mkNP</td>
<td>Quant -&gt; Num -&gt; N -&gt; NP</td>
<td>these five men</td>
</tr>
<tr>
<td>mkNP</td>
<td>Det -&gt; CN -&gt; NP</td>
<td>the five old men</td>
</tr>
<tr>
<td>mkNP</td>
<td>Det -&gt; N -&gt; NP</td>
<td>the five men</td>
</tr>
<tr>
<td>mkNP</td>
<td>Numeral -&gt; CN -&gt; NP</td>
<td>five old men</td>
</tr>
<tr>
<td>mkNP</td>
<td>Numeral -&gt; N -&gt; NP</td>
<td>five men</td>
</tr>
<tr>
<td>mkNP</td>
<td>Card -&gt; CN -&gt; NP</td>
<td>forty-five old men</td>
</tr>
<tr>
<td>mkNP</td>
<td>Card -&gt; N -&gt; NP</td>
<td>forty-five men</td>
</tr>
<tr>
<td>mkNP</td>
<td>Pron -&gt; CN -&gt; NP</td>
<td>my old man</td>
</tr>
<tr>
<td>mkNP</td>
<td>Pron -&gt; N -&gt; NP</td>
<td>my man</td>
</tr>
<tr>
<td>mkNP</td>
<td>PN -&gt; NP</td>
<td>Paris</td>
</tr>
<tr>
<td>mkNP</td>
<td>Pron -&gt; NP</td>
<td>we</td>
</tr>
<tr>
<td>mkNP</td>
<td>Quant -&gt; NP</td>
<td>this</td>
</tr>
<tr>
<td>mkNP</td>
<td>Quant -&gt; Num -&gt; NP</td>
<td>these five</td>
</tr>
<tr>
<td>mkNP</td>
<td>Det -&gt; NP</td>
<td>the five best</td>
</tr>
<tr>
<td>mkNP</td>
<td>CN -&gt; NP</td>
<td>old beer</td>
</tr>
<tr>
<td>mkNP</td>
<td>N -&gt; NP</td>
<td>beer</td>
</tr>
</tbody>
</table>
Code-generating grammar

A simplified fragment of the abstract syntax

```plaintext
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
```

A simplified fragment of the concrete syntax

```plaintext
param Voice = Act | Pass
lincat
V, V2 = Voice => Str
VP, NP, Adv, Prep = Str
lin
mkV2 v = \v\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"
```
Running examples

- **parse "jag behöver något till något"**
  - PredVP (UsePron i_Pron)
    (behöva_något_till_något_1 (DetNP someSg_Det) (DetNP someSg_Det))
  - PredVP (UsePron i_Pron)
    (behöva_något_till_något_1 (DetNP someSg_Det) something_NP)
  - PredVP (UsePron i_Pron)
    (behöva_något_till_något_1 something_NP (DetNP someSg_Det))
  - PredVP (UsePron i_Pron)
    (behöva_något_till_något_1 something_NP something_NP)

- **parse "han äter sig mätt"**
  - PredVP (UsePron he_Pron)
    (reflexiv_resultativ aeta_vb_1_1_V (PositA maett_av_1_1_A))
  - PredVP (UsePron he_Pron)
    (AdvVP (SI_refl aeta_vb_1_1_V) (PositAdvAdj maett_av_1_1_A))
  - PredVP (UsePron he_Pron)
    (AdvVP (reciprok_refl aeta_vb_1_1_V) (PositAdvAdj maett_av_1_1_A))
  - PredVP (UsePron he_Pron)
    (AdvVP (trans_refl aeta_vb_1_1_V) (PositAdvAdj maett_av_1_1_A))
  - PredVP (UsePron he_Pron)
    (V_refl_rörelse aeta_vb_1_1_V (PositAdvAdj maett_av_1_1_A))
Results

• In the current experiment, we have considered only the 96 VP constructions which resulted in 127 functions
  – Dominating in SweCcn; have the most complex internal structure

• Given the 127 functions, we have automatically generated the implementation for 98 functions (77%) achieving a 70–90% accuracy
  – There is clear space for improvement

• Manual completion postponed because of the active development of SweCcn (changes → synchronization)

• [https://github.com/GrammaticalFramework/gf-contrib](https://github.com/GrammaticalFramework/gf-contrib) (SweCcn)

• A methodology on how to systematically formalise the semi-formal representation of SweCcn in GF, showing that a GF construction grammar can be, to a large extent, acquired automatically

• Consequence: feedback to SweCcn developers on how to improve the annotation consistency and adequacy of the original construction resource