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

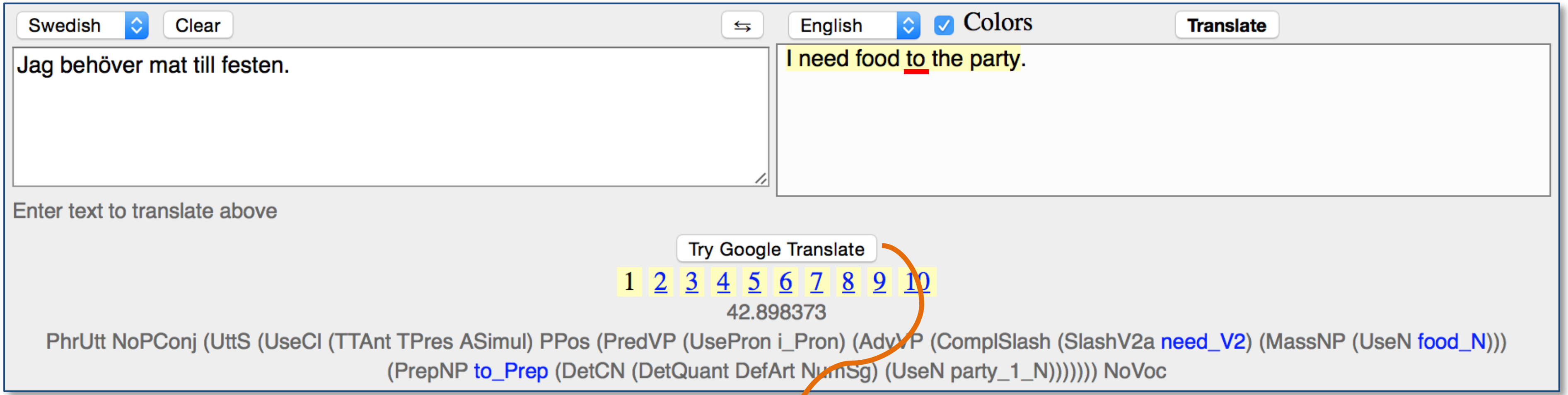
- An example from Berkeley Constructicon: “*make one’s way*”
  - Structure: {$^{\underline{\text{Motion verb}}}$ [$^{\text{Verb}}$] [$^{\text{PossNP}}$]}
  - Frame: Motion
    - [$_{\text{Theme}}$*They*] {*hacked their way*} [$_{\text{Source}}$*out*] [$_{\text{Goal}}$*into the open*].
    - [$_{\text{Theme}}$*We*] {*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*

    [Bäckström L., Lyngfelt B., Sköldberg E. (2014) Towards interlingual constructicography]



Jag behöver mat till festen.

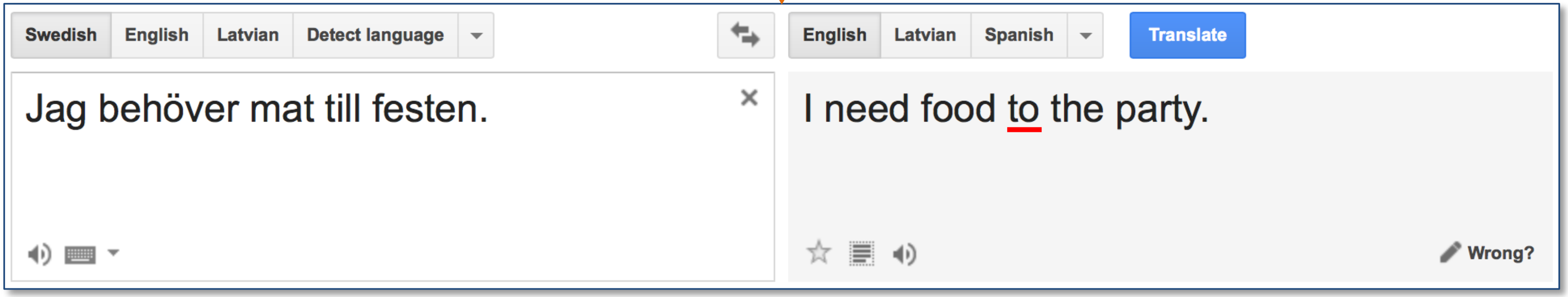
I need food to the party.

Enter text to translate above

1 2 3 4 5 6 7 8 9 10

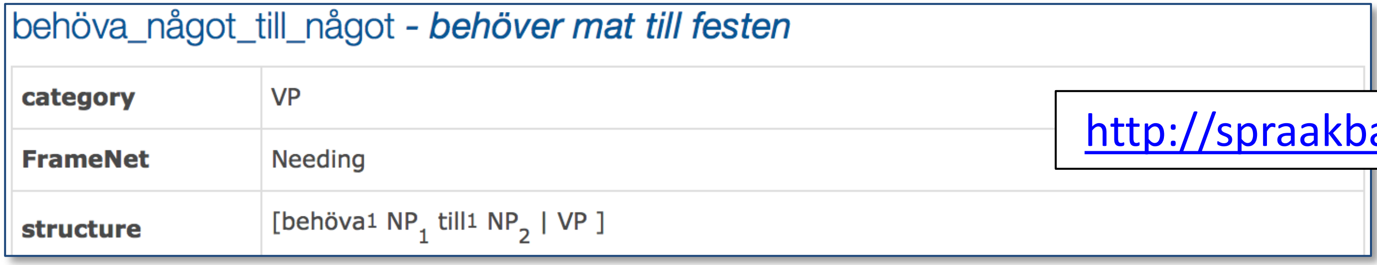
42.898373

PhrUtt NoPConj (UttS (UseCl (TTAnt TPres ASimul) PPos (PredVP (UsePron i_Pron) (AdvVP (ComplSlash (SlashV2a need_V2) (MassNP (UseN food_N))) (PrepNP to_Prep (DetCN (DetQuant DefArt NumSg) (UseN party_1_N))))))) NoVoc



Jag behöver mat till festen.

I need food to the party.



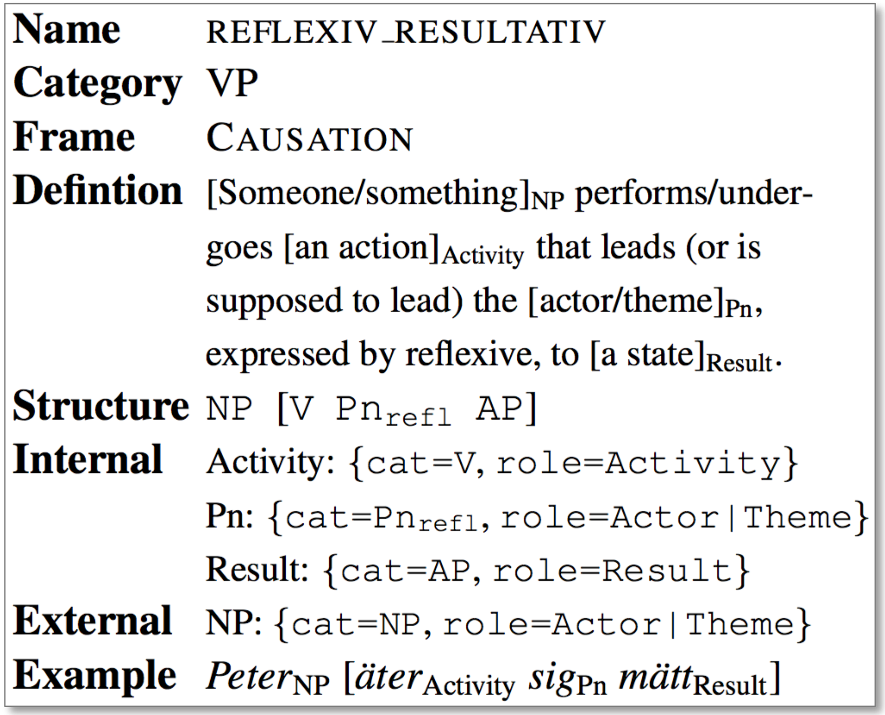
behöva_något_till_något - *behöver mat till festen*

category	VP
FrameNet	Needing
structure	[behöva1 NP$_1$ till1 NP$_2$ | VP]

http://spraakbanken.gu.se/eng/sweccn

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*)



Name	REFLEXIV_RESULTATIV
Category	VP
Frame	CAUSATION
Defintion	[Someone/something]$_{NP}$ performs/undergoes [an action]$_{Activity}$ that leads (or is supposed to lead) the [actor/theme]$_{Pn}$, expressed by reflexive, to [a state]$_{Result}$.
Structure	NP [V Pn$_{refl}$ AP]

Internal	Activity: {cat=V, role=Activity}
	Pn: {cat=Pn$_{refl}$, role=Actor|Theme}
	Result: {cat=AP, role=Result}

External	NP: {cat=NP, role=Actor|Theme}
Example	*Peter*$_{NP}$ [*äter*$_{Activity}$ *sig*$_{Pn}$ *mätt*$_{Result}$]

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* NP$_1$ *till* NP$_2$|VP →
  behöva$_V$ NP$_1$ *till*$_{Prep}$ NP$_2$ | *behöva*$_V$ NP *till*$_{Prep}$ VP

- *snacka*|*prata*|*tala* NP$_{indef}$ → (~synonyms of “to talk”)
  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*$_{Prep}$ refl$_{Pron}$ NP | V *av*$_{Prep}$ refl$_{Pron}$

- N|Adj+*städa* → (compounds)
  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?

Construction	Elements	Patterns
behöva_något_till_något_VP_1	behöva_V NP_1 till_Prep NP_2	{V} NP {Prep} NP
behöva_något_till_något_VP_2	behöva_V NP_1 till_Prep VP	{V} NP {Prep} VP

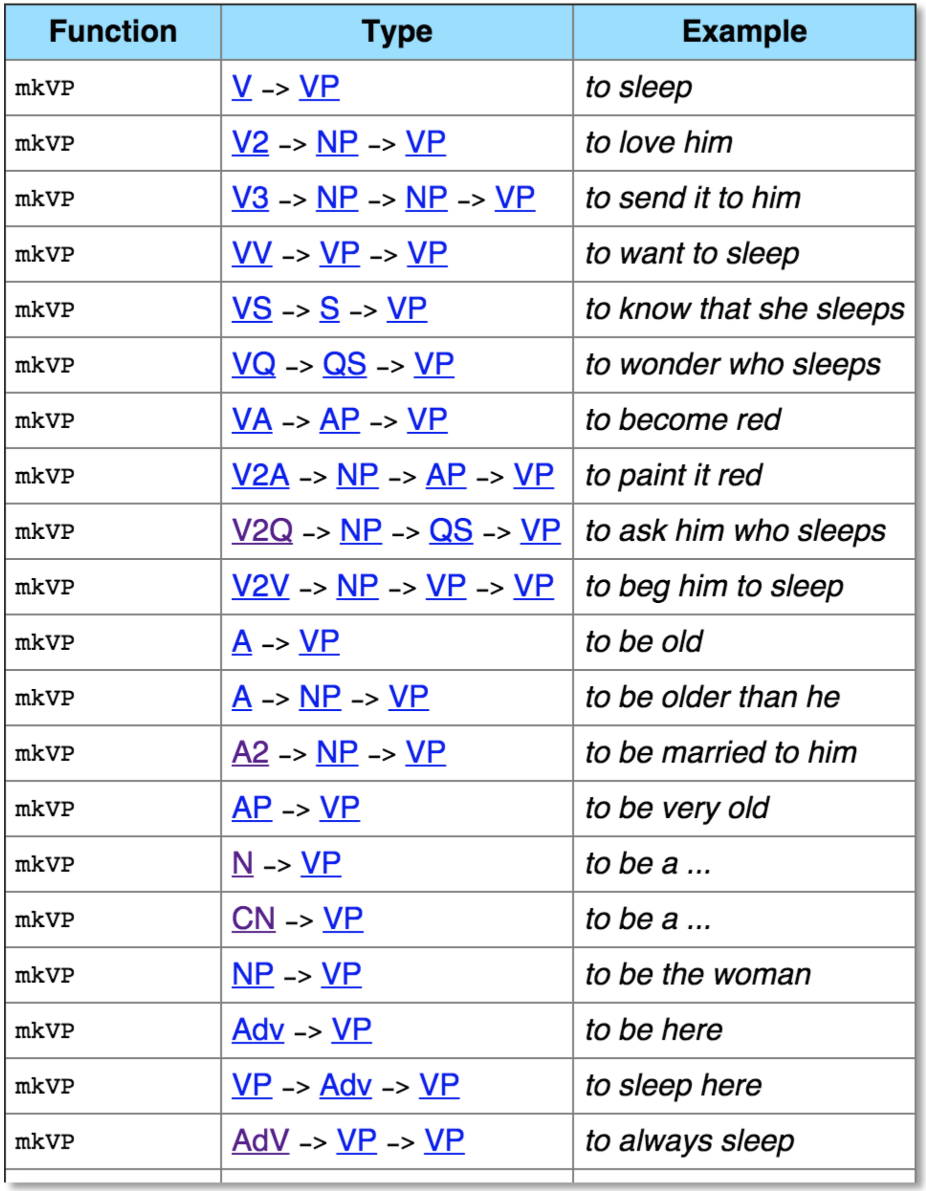
A simple GF grammar

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

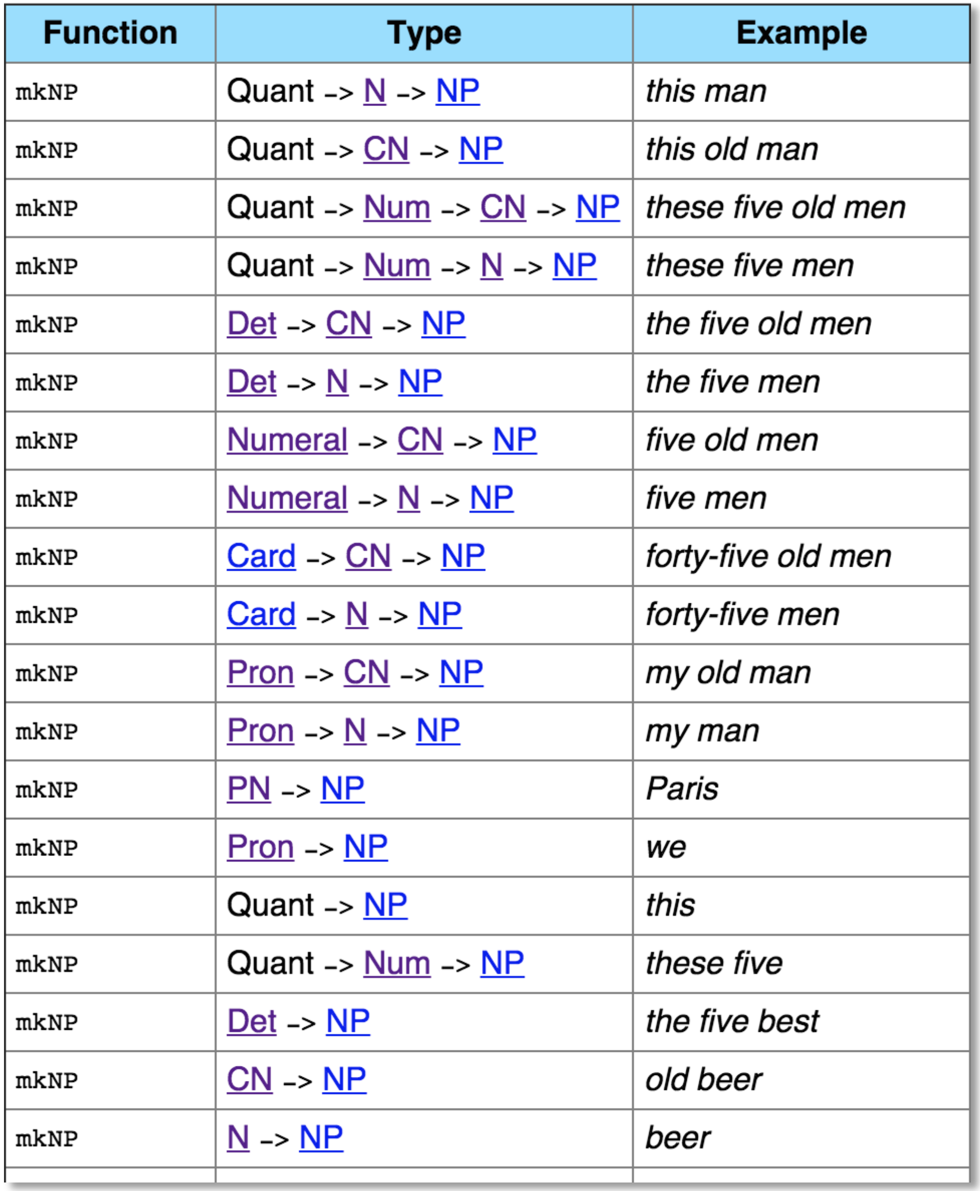
Final code (by automatic post-processing)

```
lin behöva_något_till_något_VP_1 np_1 np_2 = mkVP
  (mkVP (mkV2 (mkV "behöver")) np_1)
  (SyntaxSwe.mkAdv (mkPrep "till") np_2) ;
```

GF RGL API



Function	Type	Example
mkVP	V -> VP	*to sleep*
mkVP	V2 -> NP -> VP	*to love him*
mkVP	V3 -> NP -> NP -> VP	*to send it to him*
mkVP	VV -> VP -> VP	*to want to sleep*
mkVP	VS -> S -> VP	*to know that she sleeps*
mkVP	VQ -> QS -> VP	*to wonder who sleeps*
mkVP	VA -> AP -> VP	*to become red*
mkVP	V2A -> NP -> AP -> VP	*to paint it red*
mkVP	V2Q -> NP -> QS -> VP	*to ask him who sleeps*
mkVP	V2V -> NP -> VP -> VP	*to beg him to sleep*
mkVP	A -> VP	*to be old*
mkVP	A -> NP -> VP	*to be older than he*
mkVP	A2 -> NP -> VP	*to be married to him*
mkVP	AP -> VP	*to be very old*
mkVP	N -> VP	*to be a ...*
mkVP	CN -> VP	*to be a ...*
mkVP	NP -> VP	*to be the woman*
mkVP	Adv -> VP	*to be here*
mkVP	VP -> Adv -> VP	*to sleep here*
mkVP	AdV -> VP -> VP	*to always sleep*



Function	Type	Example
mkNP	Quant -> N -> NP	*this man*
mkNP	Quant -> CN -> NP	*this old man*
mkNP	Quant -> Num -> CN -> NP	*these five old men*
mkNP	Quant -> Num -> N -> NP	*these five men*
mkNP	Det -> CN -> NP	*the five old men*
mkNP	Det -> N -> NP	*the five men*
mkNP	Numeral -> CN -> NP	*five old men*
mkNP	Numeral -> N -> NP	*five men*
mkNP	Card -> CN -> NP	*forty-five old men*
mkNP	Card -> N -> NP	*forty-five men*
mkNP	Pron -> CN -> NP	*my old man*
mkNP	Pron -> N -> NP	*my man*
mkNP	PN -> NP	*Paris*
mkNP	Pron -> NP	*we*
mkNP	Quant -> NP	*this*
mkNP	Quant -> Num -> NP	*these five*
mkNP	Det -> NP	*the five best*
mkNP	CN -> NP	*old beer*
mkNP	N -> NP	*beer*

Code-generating grammar

```
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 **abstract syntax**

```
parse -cat=VP "{V} {Prep} NP"

mkVP__V2_NP
  (mkV2__V (partV _mkV___V
  (toStr__Prep _mkPrep_))) _NP_

mkVP__V2_NP (mkV2__V_Prep
  _mkV___V _mkPrep_) _NP_

mkVP__VP_Adv (mkVP__V _mkV___V)
  (mkAdv _mkPrep_ _NP_)
```

```
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"
```

A simplified fragment of the **concrete syntax**

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 (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