Embedded Controlled Languages

Aarne Ranta

CNL 2014, Galway
20-22 August 2014
Joint work with

Krasimir Angelov, Björn Bringert, Grégoire Détrez, Ramona Enache, Erik de Graaf, Normunds Gruzitis, Qiao Haiyan, Thomas Hallgren, Prasanth Kolachina, Inari Listenmaa, Peter Ljunglöf, K.V.S. Prasad, Scharolta Siencnik, Shafqat Virk

50+ GF Resource Grammar Library contributors
Embedded programming languages

DSL = Domain Specific Language

Embedded DSL = fragment (library) of a host language
+ low implementation effort
+ no additional learning if you know the host language
+ you can fall back to host language if DSL is not enough

- reasoning about DSL properties more difficult
Timeline

1998: GF = Grammatical Framework
2001: RGL = Resource Grammar Library
2008: CNL, explicitly
2010: MOLTO: CNL-based translation
2012: wide-coverage translation
2014: embedded CNL translation
Outline

- “CNL is a part of NL”
- CNL embedded in NL
- Example: translation
- Demo: web and mobile app
CNL as a part of NL

It is a part:
● it is understandable without extra learning

It is a proper part:
● it excludes parts that are not so good
● it can be controlled, maybe even defined
How to define and delimit a CNL

How to guarantee that it is a part
- the CNL may be formal, the NL certainly isn’t

How to help keep within the limits
- so that the user stays within the CNL
Bottom-up vs. top-down CNL

**Bottom-up**: define CNL rule by rule
- nothing is in the CNL unless given by rules
- e.g. Attempto Controlled English

**Top-down**: delimit CNL by constraining NL
- everything is in the CNL unless blocked by rules
- e.g. Simplified English
Defining and delimiting CNL

Bottom-up:
- How do we know that the rules are valid NL?

Top-down:
- How do we decide what is in the CNL?
Defining bottom-up

Message ::= “you have” Number “points”

you have five points

you have one points
Delimiting top-down

Passives must be avoided.

How to recognize them in all contexts? Tenses, questions, infinitives, separate from adjectives...
An answer to both problems

Define CNL **formally** as a part of NL

- use a grammar of the whole NL
- bottom-up: rules defined as applications of NL rules
- top-down: constraints written as conditions on NL trees
The whole NL?

An approximation: **GF Resource Grammar Library (RGL)**

- morphology
- syntactic structures
- lexicon
- common syntax API
- 29 languages
Bottom-up CNL

Use RGL as library

- use its API function calls rather than plain strings

\[
\text{HavePoints } p \ n = \text{mkCl } p \ \text{have}_V^2 (\text{mkNP } n \ \text{point}_N)
\]

This generates *you have five points*, *she has one point*, etc.

Also in other languages
Top-down CNL

Use RGL as **run-time grammar**

- use its parser to produce trees
- filter trees by pattern matching

```haskell
hasPassive t = case t of
    PassVPSlash _ -> return True
    _ _ -> composOp hasPassive t
```

(Bringert & Ranta, A Pattern for Almost Compositional Operations, JFP 2008)
Top-down CNL

Use RGL as **run-time grammar**
- change unwanted input

\[
\text{unPassive } t = \text{case } t \text{ of }
\]
\[
\text{PredVP } np \text{ (PassVPSlash } vps) \rightarrow \text{liftM2 PredVP (unPassive } np) \text{ (unPassive } vps) \\
\_ \rightarrow \text{composOp unPassive } t
\]

Non-CNL input is **recognized** but **corrected**.
Embedded bottom-up CNL

1. Define CNL as usual, maybe with RGL as library
2. Build a module that inherits both CNL and RGL

```
abstract Embedded = CNL, RGL ** { 
  cat Start ;
  fun UseCNL : CNL_Start -> Start ;
  fun UseRGL : RGL_Start -> Start ;
}
```
Using embedded CNL

Parsing will try both CNL and RGL.

You can give priority to CNL trees.

The parser is robust (if RGL has enough coverage)

Non-CNL input is not a failure, but can be processed further.
Example: translation

We want to have machine translation that

- delivers **publication quality** in areas where reasonable effort is invested
- degrades gracefully to **browsing quality** in other areas
- shows a clear distinction between these

We do this by using **grammars** and **type-theoretical interlinguas** implemented in **GF, Grammatical Framework**
what is your wife's name
vad heter din fru
the vice president kicked the bucket
skruvstäds-presidenten
sparkade hinken
long time no see
lång tid nej ser
what is your wife's name

vad heter din fru

the vice president kicked the bucket

skruvstådspräsidetin
sparkade hinken

long time no see

lång tid nej ser
what is your wife's name
- translation by **meaning**
  - correct
  - idiomatic

the vice president kicked the bucket
- translation by **syntax**
  - grammatical
  - often strange
  - often wrong

long time no see
- translation by **chunks**
  - probably ungrammatical
  - probably wrong
The Vauquois triangle

- Word to word transfer
- Syntactic transfer
- Semantic interlingua
The Vauquois triangle

- Semantic interlingua
- Syntactic transfer
- Word to word transfer
What is it good for?
publish the content
get the grammar right
get an idea
Who is doing it?
GF in MOLTO

GF the last 15 months

Google, Bing, Apertium
What should we work on?
All!

- semantics for full quality and speed
- syntax for grammaticality
- chunks for robustness and speed
We want a system that
● can reach perfect quality
● has robustness as back-up
● tells the user which is which

We “combine GF, Apertium, and Google”

But we do it all in GF!
How to do it?

a brief summary
translator

chunk grammar

CNL grammar

resource grammar
How much work is needed?
translator

chunk grammar

resource grammar

CNL grammars
resource grammar

- morphology
- syntax
- generic lexicon

precise linguistic knowledge
manual work can’t be escaped
CNL grammars

domain semantics, domain idioms
● need domain expertise
use resource grammar as library
● minimize hand-hacking

the work never ends
● we can only cover some domains
chunk grammar

words
suitable word sequences
● local agreement
● local reordering
easily derived from resource grammar
easily varied
minimize hand-hacking
PGF run-time system
● parsing
● linearization
● disambiguation
generic for all grammars
portable to different user interfaces
● web
● mobile
Disambiguation?

**Grammatical:** give priority to green over yellow, yellow over red

**Statistical:** use a distribution model for grammatical constructs (incl. word senses)

**Interactive:** for the last mile in the green zone
Advantages of GF

Expressivity: easy to express complex rules
- agreement
- word order
- discontinuity

Abstractions: easy to manage complex code

Interlinguality: easy to add new languages
Resources: basic and bigger

Norwegian Danish Afrikaans

English Swedish German Dutch
French Italian Spanish
Bulgarian Chinese Finnish

Greek Latvian Thai Japanese

Urdu Punjabi Sindhi Nepali Persian

Greek

English
Swedish
German
Dutch
French
Italian
Spanish
Bulgarian
Chinese
Finnish

Greek

Urdu
Punjabi
Sindhi
Nepali
Persian

Greek

Greek

Greek

Greek
my new house is very big

मेरा अजनबी शाला बहुत महत्वपूर्ण है

你爱我吗

est-ce que tu m'aimes

ich wohne in einem gelben Haus

io risiedo in una casa gialla

jag är inte en ålg

minä en ole hirvi
How to do it?

some more details
Translation model: multi-source multi-target compiler
Translation model: multi-source multi-target compiler-decompiler

Abstract Syntax

- Hindi
- Chinese
- Finnish
- Bulgarian
- Spanish
- Italian
- German
- French
- Swedish
- English
Word alignment: compiler

1  +  2  *  3

00000011 00000100 00000101 01101000 01100000
Abstract syntax

Add : Exp -> Exp -> Exp
Mul : Exp -> Exp -> Exp
E1, E2, E3 : Exp

Add E1 (Mul E2 E3)
## Concrete syntax

<table>
<thead>
<tr>
<th>abstrakt</th>
<th>Java</th>
<th>JVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add $x \ y$</td>
<td>$x \ &quot;+&quot; \ y$</td>
<td>$x \ y \ &quot;01100000&quot;$</td>
</tr>
<tr>
<td>Mul $x \ y$</td>
<td>$x \ &quot;*&quot; \ y$</td>
<td>$x \ y \ &quot;01101000&quot;$</td>
</tr>
<tr>
<td>$E_1$</td>
<td>&quot;1&quot;</td>
<td>&quot;00000011&quot;</td>
</tr>
<tr>
<td>$E_2$</td>
<td>&quot;2&quot;</td>
<td>&quot;00000100&quot;</td>
</tr>
<tr>
<td>$E_3$</td>
<td>&quot;3&quot;</td>
<td>&quot;00000101&quot;</td>
</tr>
</tbody>
</table>
Compiling natural language

Abstract syntax

\[ \text{Pred} : \ NP \rightarrow \ V2 \rightarrow \ NP \rightarrow \ S \]
\[ \text{Mod} : \ AP \rightarrow \ CN \rightarrow \ CN \]
\[ \text{Love} : \ V2 \]

Concrete syntax:

\[ \text{Pred} \ s \ v \ o \]
\[ \text{Mod} \ a \ n \]
\[ \text{Love} \]

<table>
<thead>
<tr>
<th>English</th>
<th>Latin</th>
</tr>
</thead>
<tbody>
<tr>
<td>s v o</td>
<td>s o v</td>
</tr>
<tr>
<td>a n</td>
<td>n a</td>
</tr>
<tr>
<td>“love”</td>
<td>“amare”</td>
</tr>
</tbody>
</table>
the clever woman loves the handsome man

femina sapiens virum formosum amat

Pred (Def (Mod Clever Woman)) Love
(Def (Mod Handsome Man))
# Linearization types

<table>
<thead>
<tr>
<th>English</th>
<th>Latin</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CN</strong></td>
<td>{s : Number =&gt; Str}</td>
</tr>
<tr>
<td><strong>AP</strong></td>
<td>{s : Str}</td>
</tr>
</tbody>
</table>

**Mod ap cn**

\[
\begin{align*}
{s = \backslash n => ap.s ++ cn.s ! n} & \quad {s = \backslash n, c => cn.s ! n ! c ++ ap.s ! cn.g ! n ! c ;} \\
g & = cn.g
\end{align*}
\]
Abstract syntax trees

my name is John

HasName I (Name “John”)
Abstract syntax trees

my name is John

HasName I (Name “John”)

Pred (Det (Poss i_NP) name_N)) (NameNP “John”)

...
Abstract syntax trees

my name is John

HasName I (Name “John”)

Pred (Det (Poss i_NP) name_N)) (NameNP “John”)

[DetChunk (Poss i_NP), NChunk name_N, copulaChunk, NPChunk (NameNP “John”)]
Building the yellow part
Building a basic resource grammar

Programming skills
Theoretical knowledge of language
3-6 months work
3000-5000 lines of GF code
- not easy to automate
+ only done once per language
Building a large lexicon

Monolingual (morphology + valencies)
- extraction from open sources (SALDO etc)
- extraction from text (extract)
- smart paradigms

Multilingual (mapping from abstract syntax)
- extraction from open sources (Wordnet, Wiktionary)
- extraction from parallel corpora (Giza++)

Manual quality control at some point needed
Improving the resources

Multiwords: non-compositional translation
- *kick the bucket - ta ner skylten*

Constructions: multiwords with arguments
- *i sötaste laget - excessively sweet*

Extraction from free resources (Konstruktikon)
Extraction from phrase tables
- example-based grammar writing
Building the green part
Define **semantically based abstract syntax**

```
fun HasName : Person -> Name -> Fact
```

Define **concrete syntax by mapping to resource grammar structures**

```
lin HasName p n = mkCl (possNP p name_N) y
  my name is John
lin HasName p n = mkCl p heta_V2 y
  jag heter John
lin HasName p n = mkCl p (reflV chiamare_V) y
  (io) mi chiamo John
```
Resource grammars give crucial help
● CNL grammarians need not know linguistics
● a substantial grammar can be built in a few days
● adding new languages is a matter of a few hours

MOLTO’s goal was to make this possible.
Automatic extraction of CNLs?

- abstract syntax from ontologies
- concrete syntax from examples
  - including phrase tables

As always, full green quality needs expert verification

- formal methods help (REMU project)
These grammars are a source of
- “non-compositional” translations
- compile-time transfer
- idiomatic language
- translating meaning, not syntax

Constructions are the generalized form of this idea, originally domain-specific.
Building the red part
1. Write a grammar that builds sentences from sequences of chunks

```plaintext
cat Chunk
fun SChunks : [Chunk] -> S
```

2. Introduce chunks to cover phrases

```plaintext
fun NP_nom_Chunk : NP -> Chunk
fun NP_acc_Chunk : NP -> Chunk
fun AP_sg_masc_Chunk : AP -> Chunk
fun AP_pl_fem_Chunk : AP -> Chunk
```
Do this for all categories and feature combinations you want to cover.

Include both long and short phrases
- long phrases have better quality
- short phrases add to robustness

Give long phrases priority by probability settings.
Long chunks are better:

[this yellow house] - [det här gula huset]

[this] [yellow house] - [den här] [gult hus]

[this] [yellow] [house] - [den här] [gul] [hus]

Limiting case: whole sentences as chunks.
Accurate feature distinctions are good, especially between closely related language pairs.

Apertium does this for every language pair.
Resource grammar chunks of course come with reordering and internal agreement.

**French:***
- *Prep*  : *dans*
- *Det+Fem+Sg* : *la*
- *N+Fem+Sg* : *maison*
- *A+Fem+Sg* : *bleue*

**German:***
- *Prep*  : *im*
- *Det+Neutr+Sg+Dat* : *blauen*
- *A+Weak+Dat* : *Haus*
- *N+Neutr+Sg* : *Haus*
Recall: chunks are just a by-product of the real grammar.

Their size span is

single words $\leftrightarrow$ entire sentences

A wide-coverage chunking grammar can be built in a couple of hours by using the RGL.
Building the translation system
GF source

probability model
GF source → GF compiler → PGF binary
user interface

PGF runtime system

another PGF binary

CNL
**White**: free, open-source.  **Green**: a business idea (Digital Grammars)
User interfaces
command-line
shell
web server
web applications
mobile applications
Demos
To test it yourself

Android app

http://www.grammaticalframework.org/demos/app.html

Web app

http://www.grammaticalframework.org/demos/translation.html
Take home
Implementing CNL in GF using RGL
  ● less work and linguistic expertise
  ● multilinguality (29 languages)

Embedding CNL in RGL
  ● robustness
  ● confidence control

On-going effort: translation
  ● CNL as semantic model
  ● contributions wanted to lexicon etc!

Other CNL applications: to do!