Embedded Controlled Languages

Aarne Ranta

CNL 2014, Galway
20-22 August 2014

CLT REMU digital Grammars
Language technology to rely on.

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

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

```
HavePoints p n = mkCl p have_V2 (mkNP n 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

```
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

```
unPassive t = case t of
  PredVP np (PassVPSlash vps) -> liftM2 PredVP (unPassive np) (unPassive vps)
  _ -> 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)

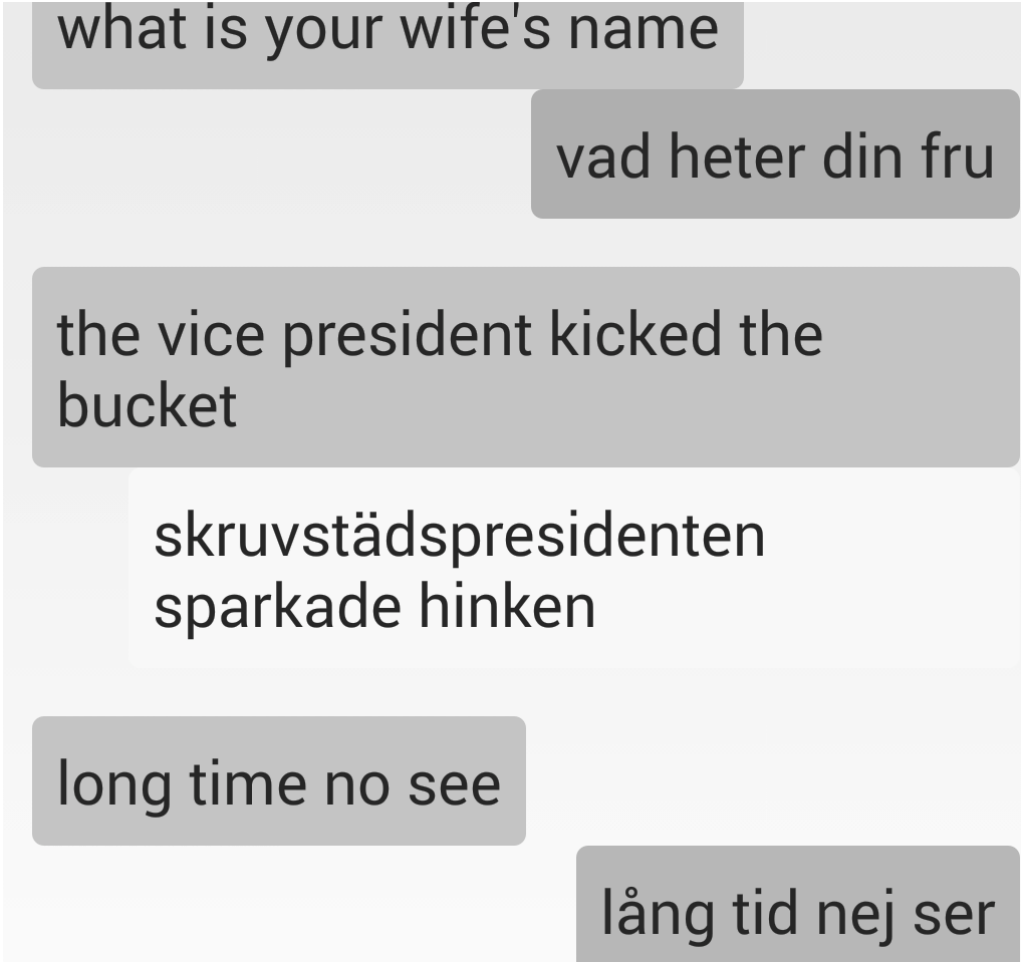
Non-CNL input is not a failure, but can be processed further.

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**



GF translation app in greyscale

what is your wife's name

vad heter din fru

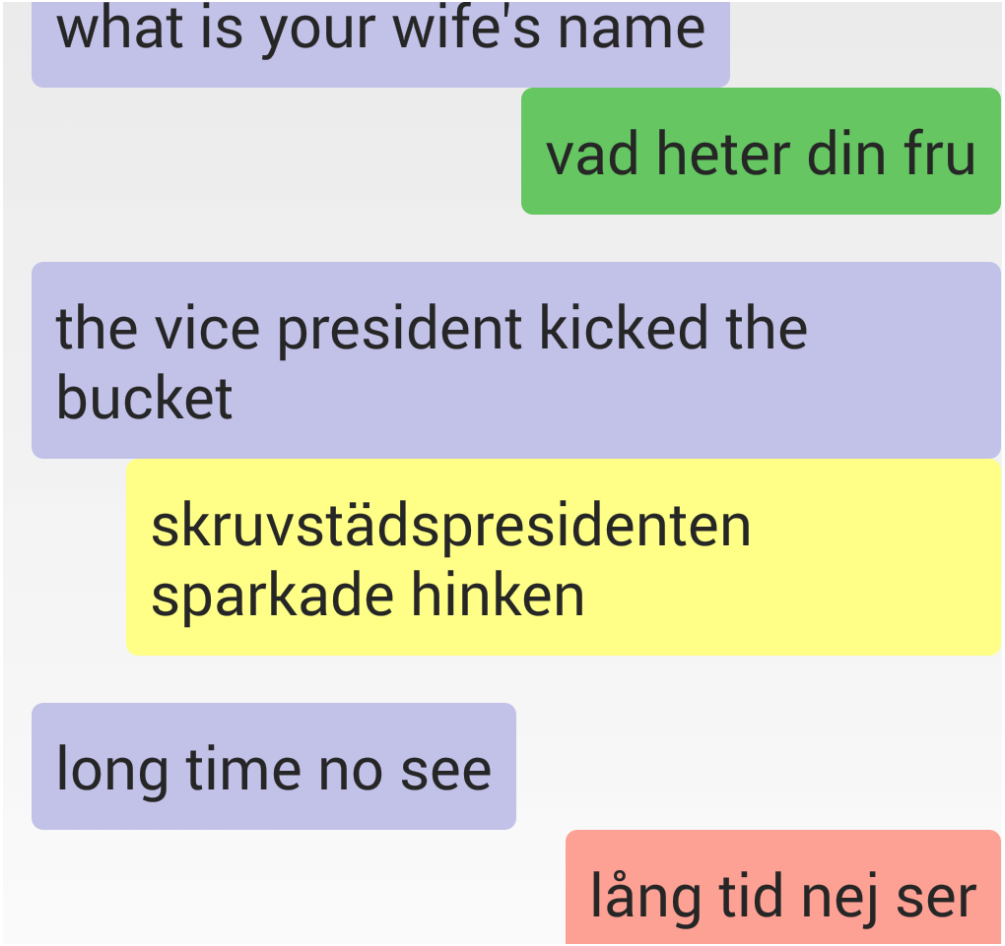
the vice president kicked the bucket

skruvstädspresidenten sparkade hinken

long time no see

lång tid nej ser

GF translation app in full colour

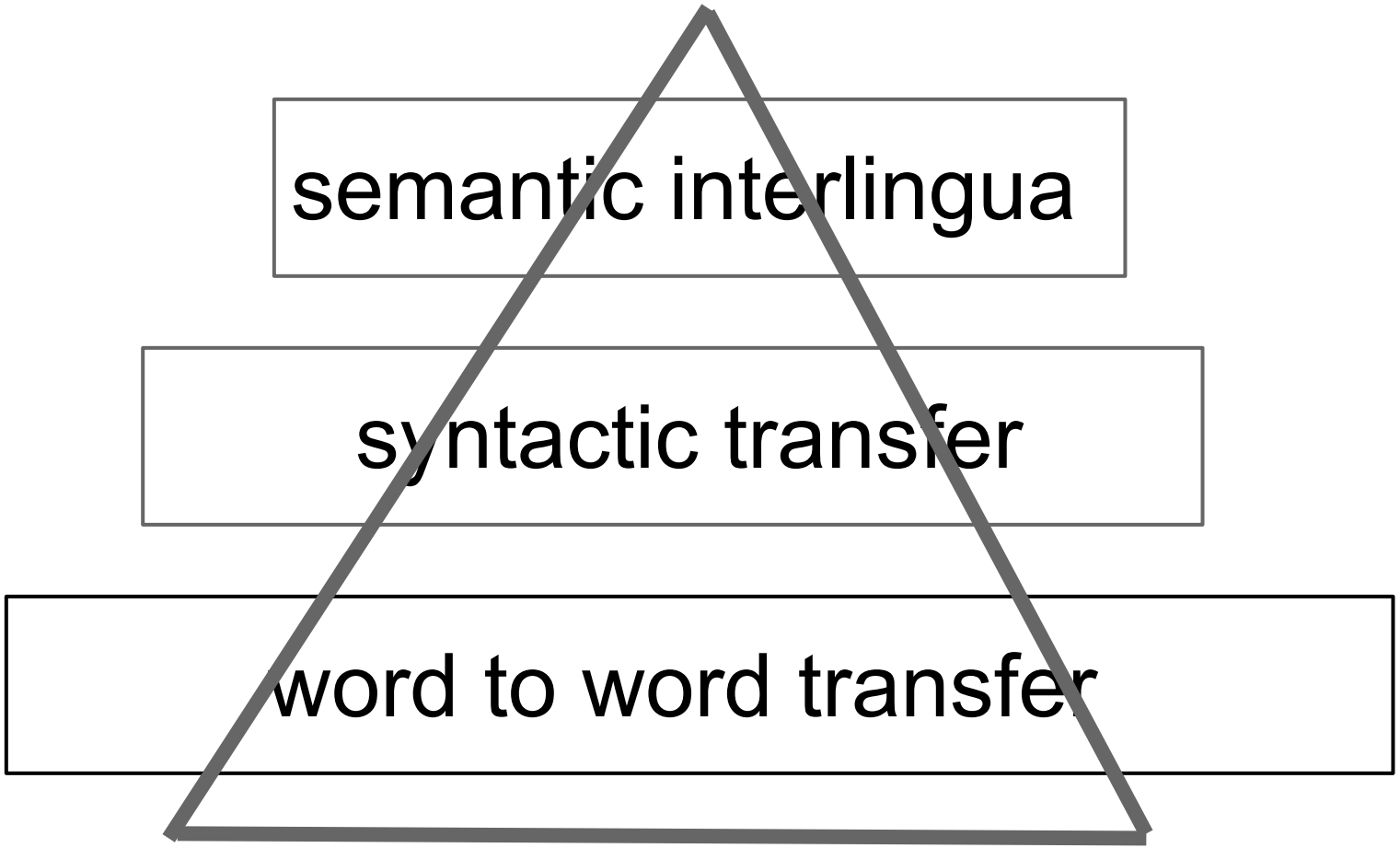


translation by **meaning**
- correct
- idiomatic

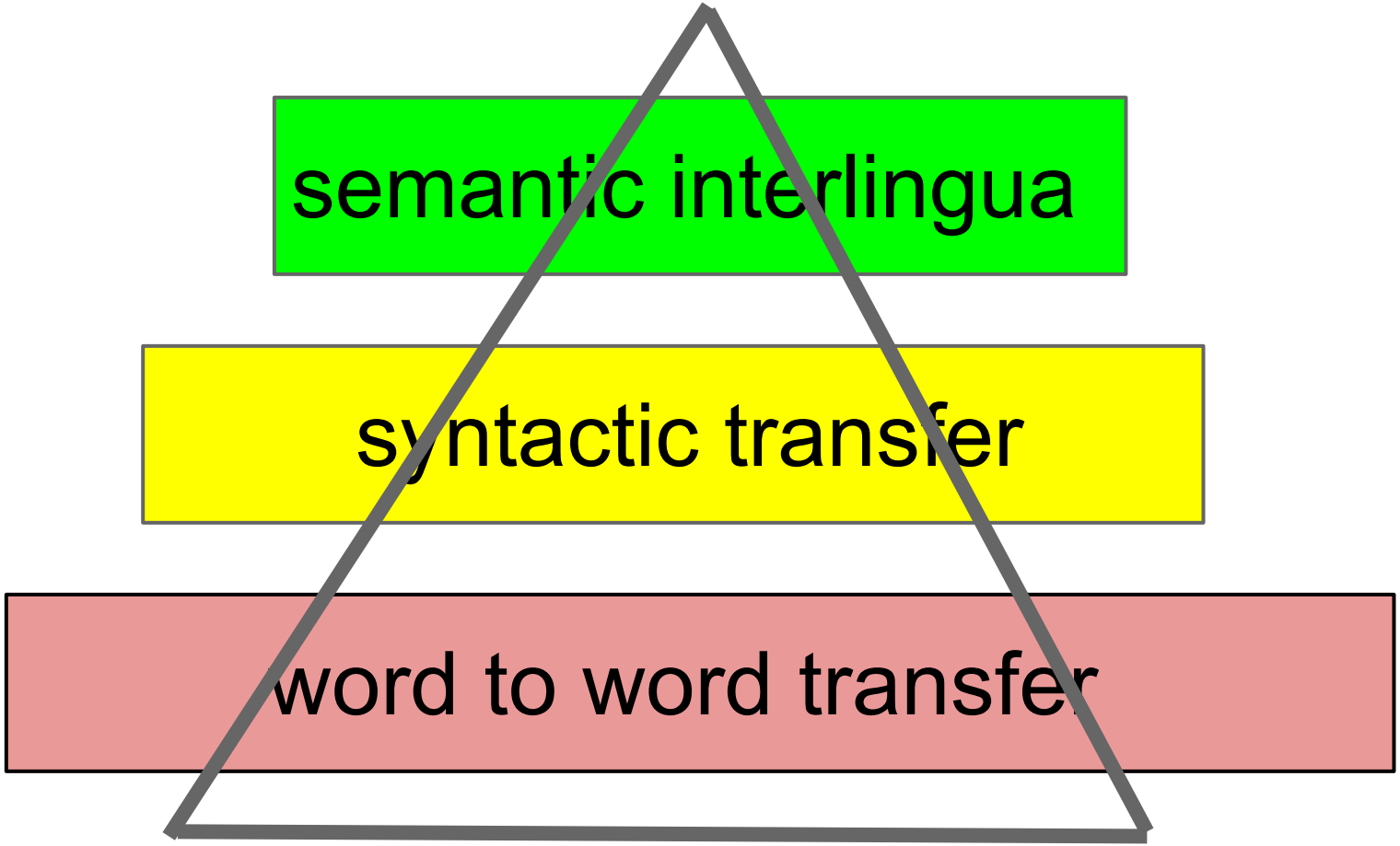
translation by **syntax**
- grammatical
- often strange
- often wrong

translation by **chunks**
- probably ungrammatical
- probably wrong

The Vauquois triangle



The Vauquois triangle



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

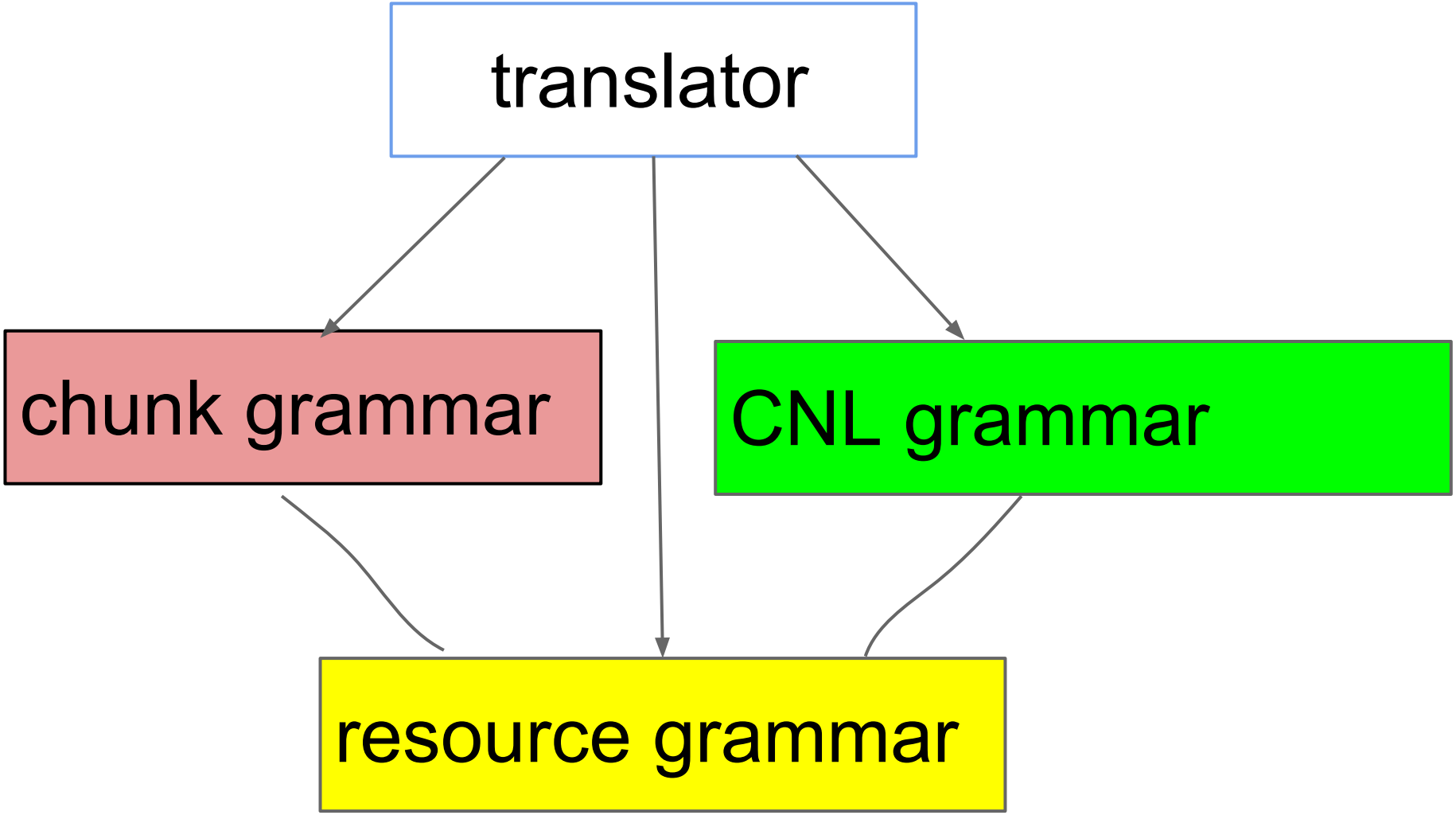
- can reach perfect quality
- has robustness as back-up
- tells the user which is which

We “combine GF, Apertium, and Google”

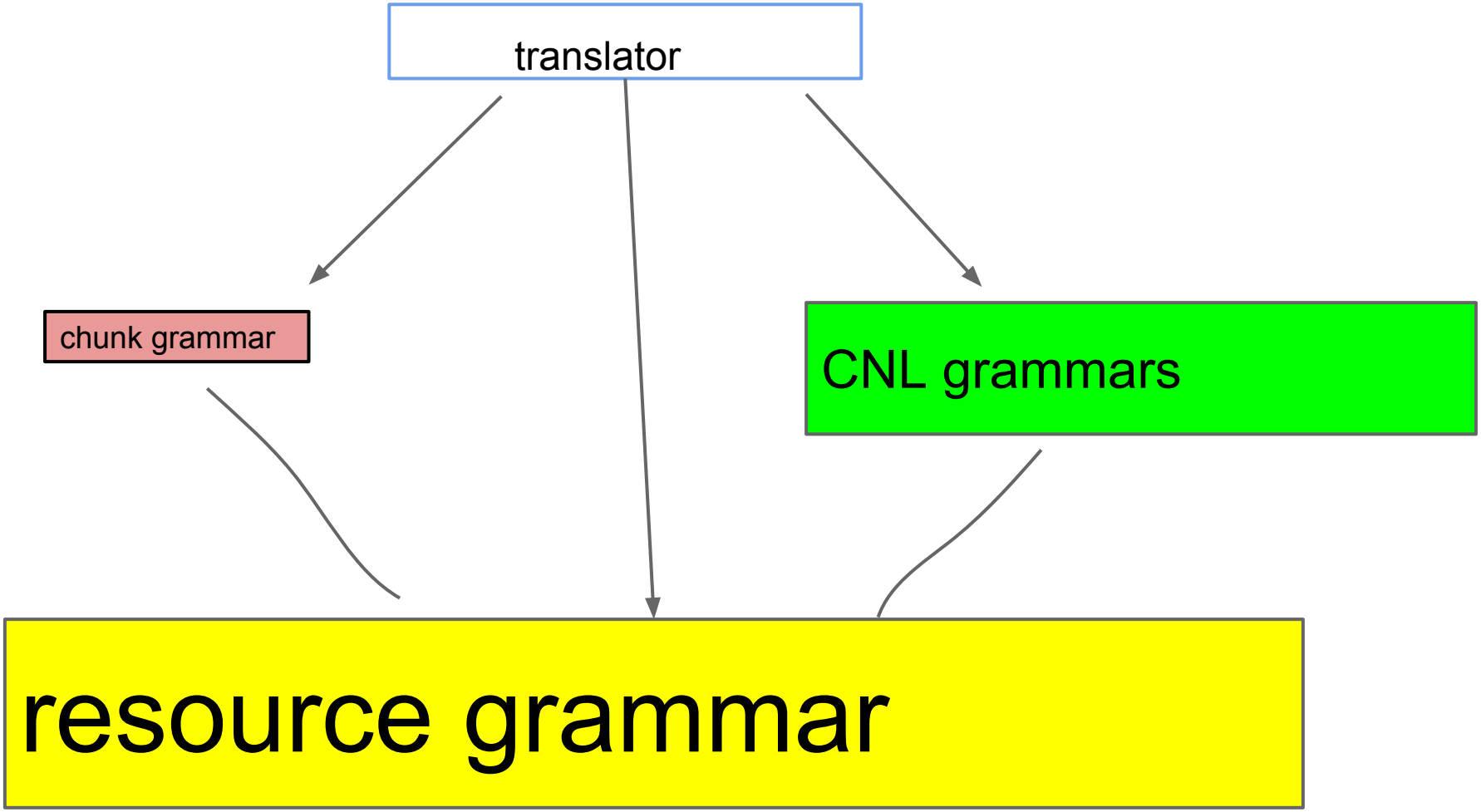
But we do it all in GF!

How to do it?

a brief summary


translator
chunk grammar
CNL grammar
resource grammar

How much work is needed?


translator
chunk grammar
CNL grammars
resource grammar

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

translator

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

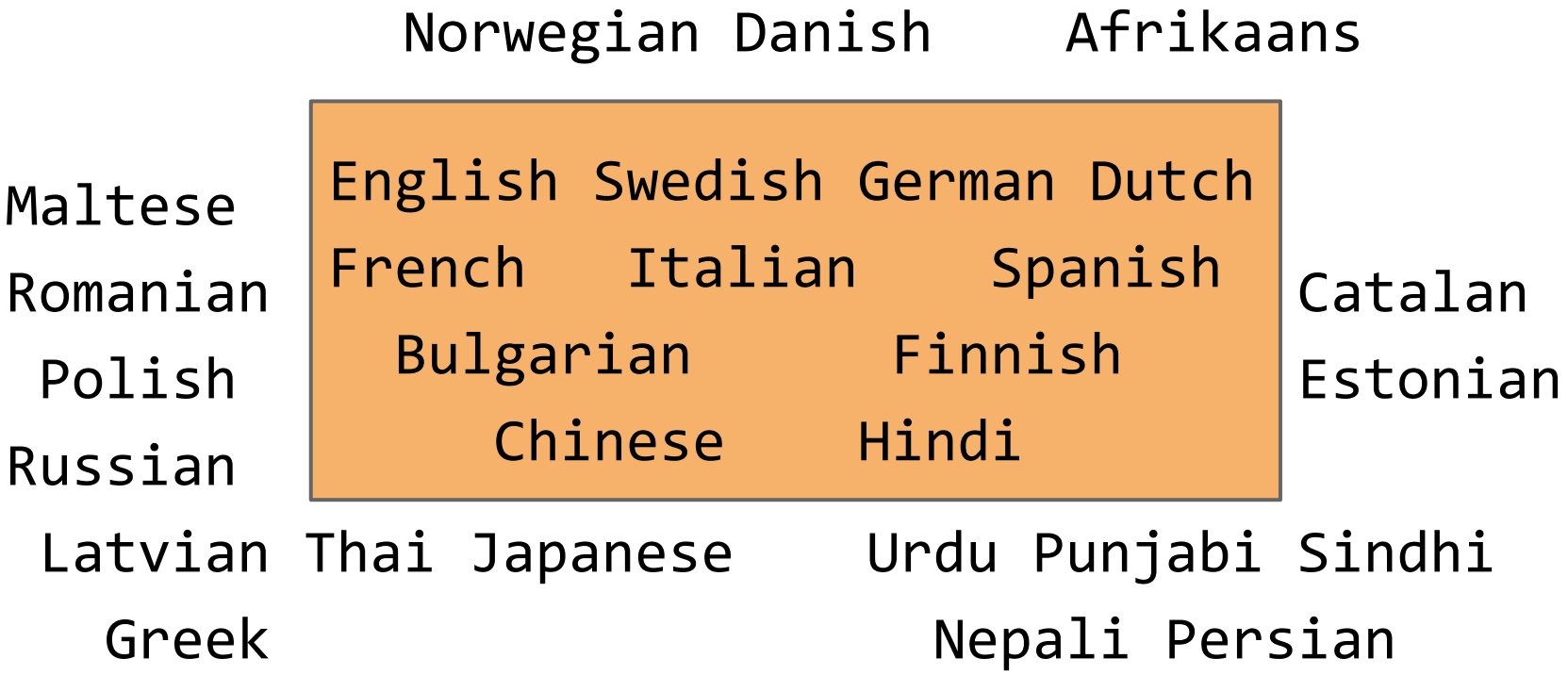
Expressivity: easy to express complex rules

- agreement
- word order
- discontinuity

Abstractions: easy to manage complex code

Interlinguality: easy to add new languages

Resources: basic and bigger



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

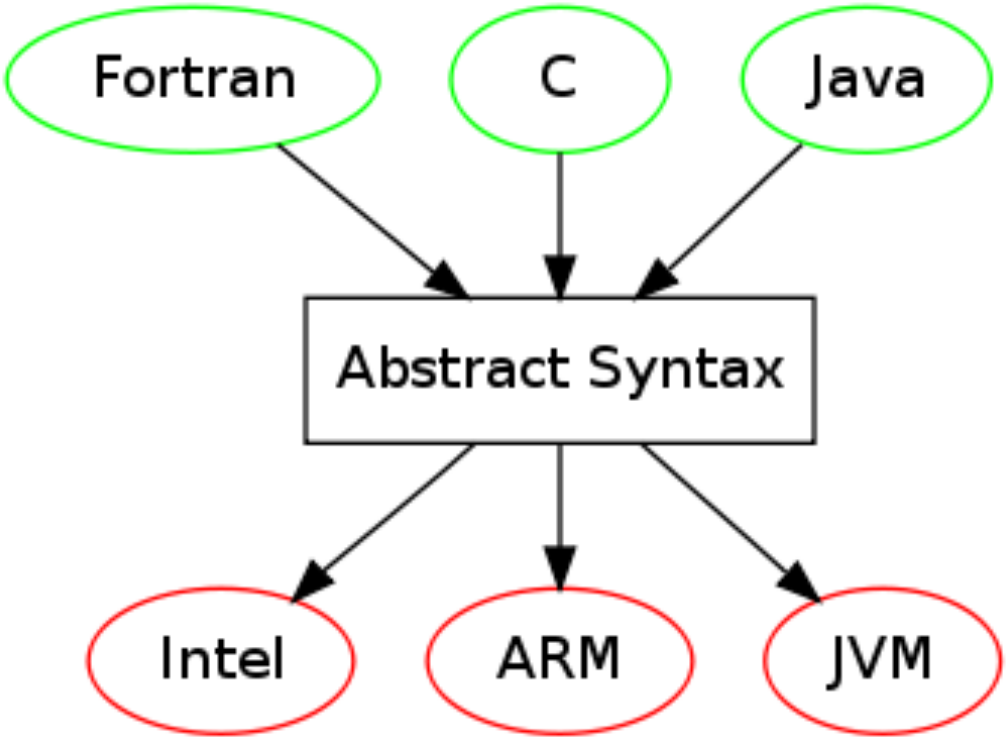
jag är inte en älg

minä en ole hirvi

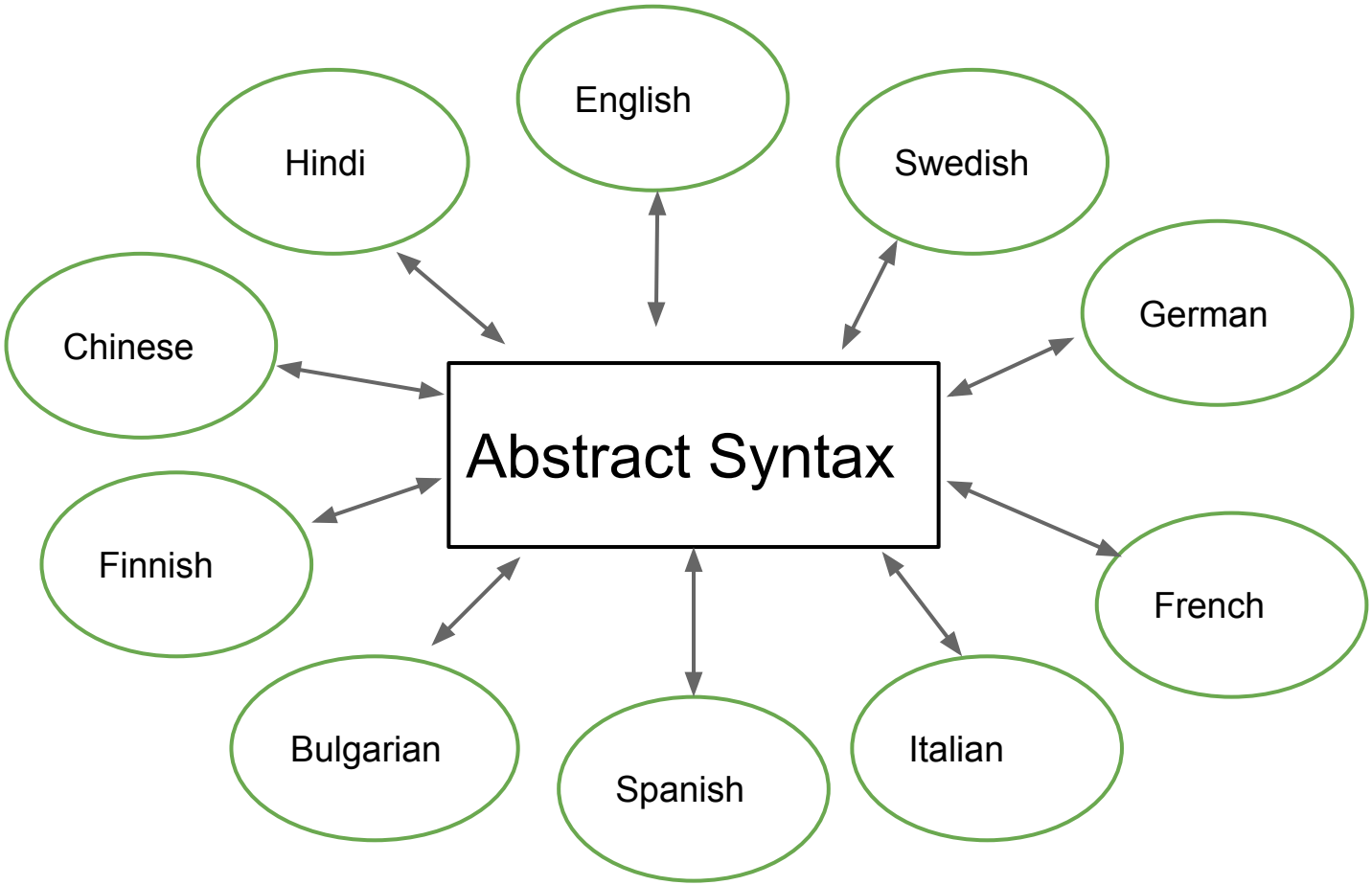
How to do it?

some more details

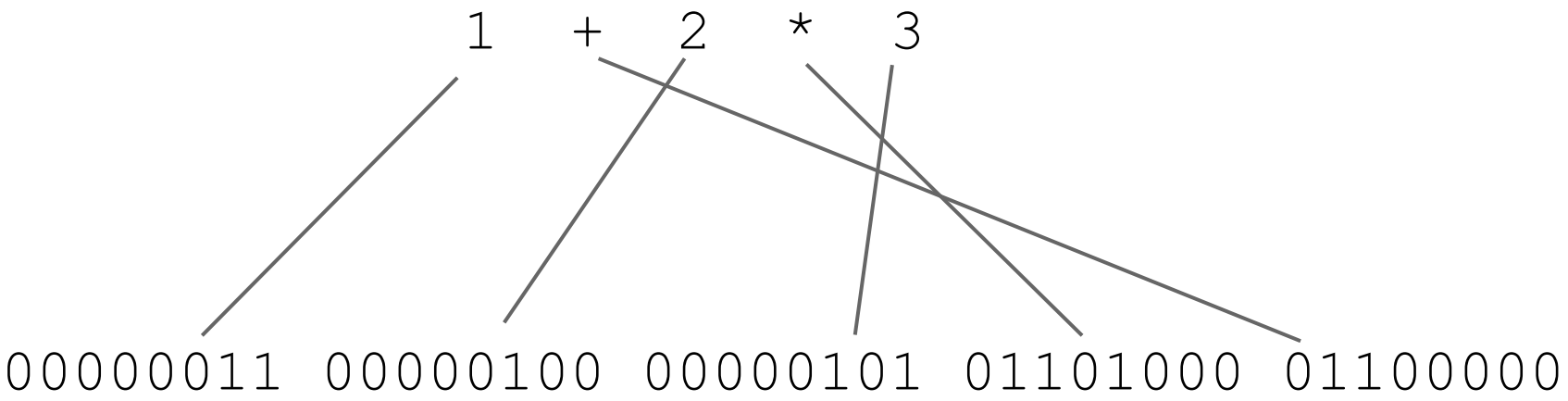
Translation model: multi-source multi-target compiler



Translation model: multi-source multi-target compiler-**decompiler**



Word alignment: compiler



Abstract syntax

Add : Exp -> Exp -> Exp

Mul : Exp -> Exp -> Exp

E1, E2, E3 : Exp

Add E1 (Mul E2 E3)

Concrete syntax

abstrakt	**Java**	**JVM**
Add x y	*x* “+” *y*	*x y* “01100000”
Mul x y	*x* “*” *y*	*x y* “01101000”
E1	“1”	“00000011”
E2	“2”	“00000100”
E3	“3”	“00000101”

Compiling natural language

Abstract syntax

Pred : NP -> V2 -> NP -> S

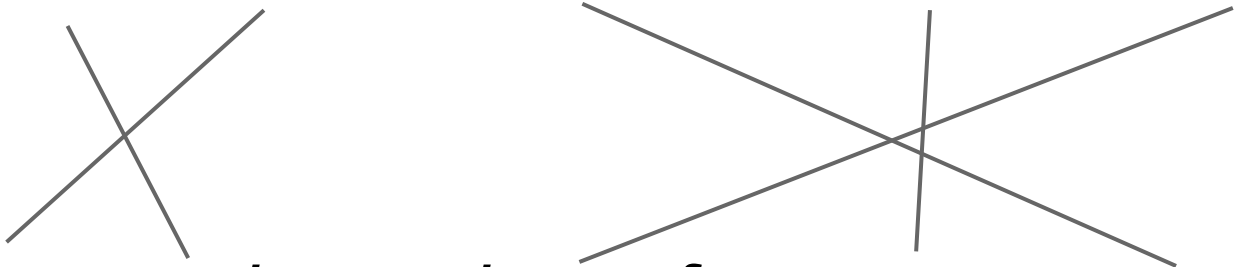
Mod : AP -> CN -> CN

Love : V2

Concrete syntax:	**English**	**Latin**
Pred s v o	*s v o*	*s o v*
Mod a n	*a n*	*n a*
Love	*“love”*	*“amare”*

Word alignment

the clever woman loves the handsome man



femina sapiens virum formosum amat

Pred (Def (Mod Clever Woman)) Love
(Def (Mod Handsome Man))

Linearization types

	English	Latin
CN	*{s : Number => Str}*	*{s : Number => Case => Str ; g : Gender}*
AP	*{s : Str}*	*{s : Gender => Number => Case => Str}*

Mod ap cn

{s = \\n => ap.s ++ cn.s ! n}	*{s = \\n,c => cn.s ! n ! c ++ ap.s ! cn.g ! n ! c ;* *g = cn.g* *}*

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

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

2. Introduce chunks to cover phrases

```
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:

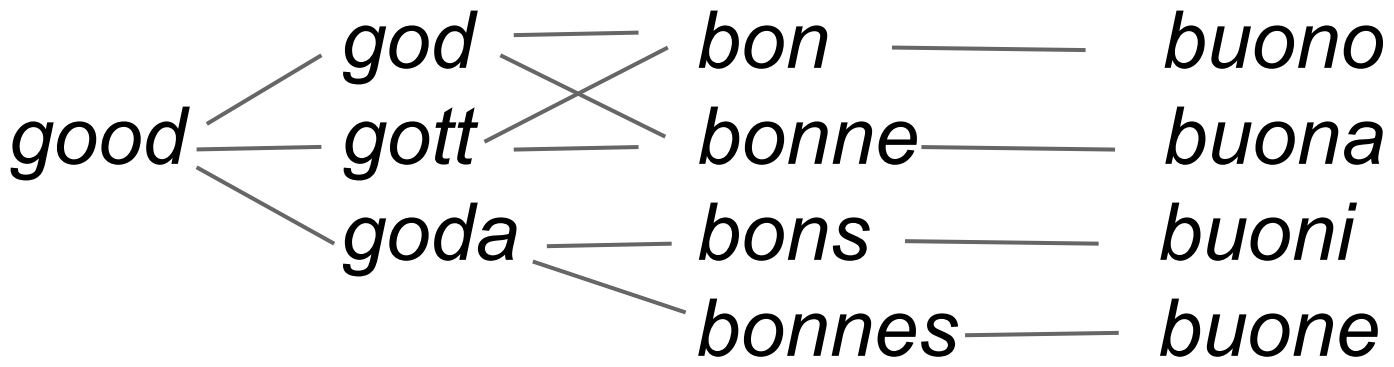
[this yellow house] - *[det här gula huset]*

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

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

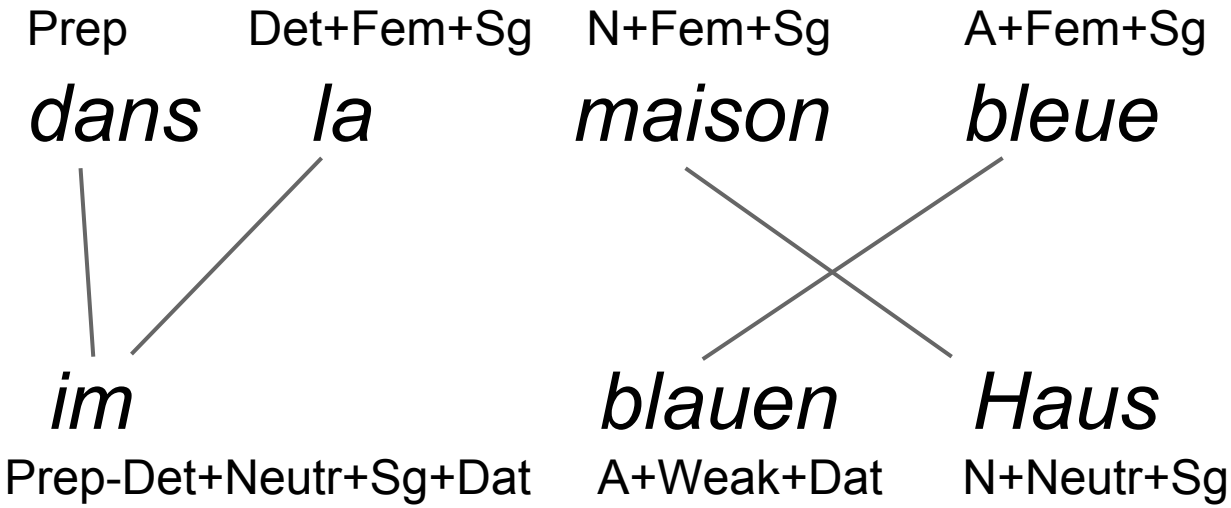
Limiting case: whole sentences as chunks.

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



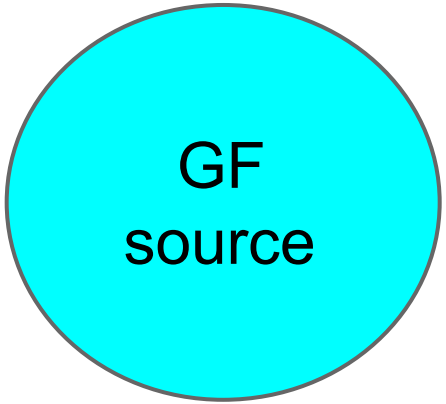
Recall: chunks are just a by-product of the real grammar.

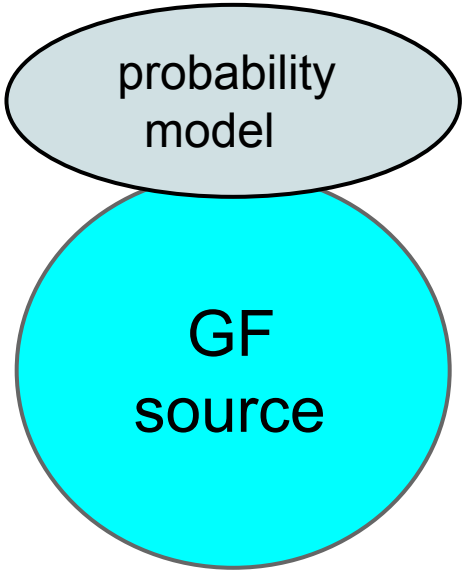
Their size span is

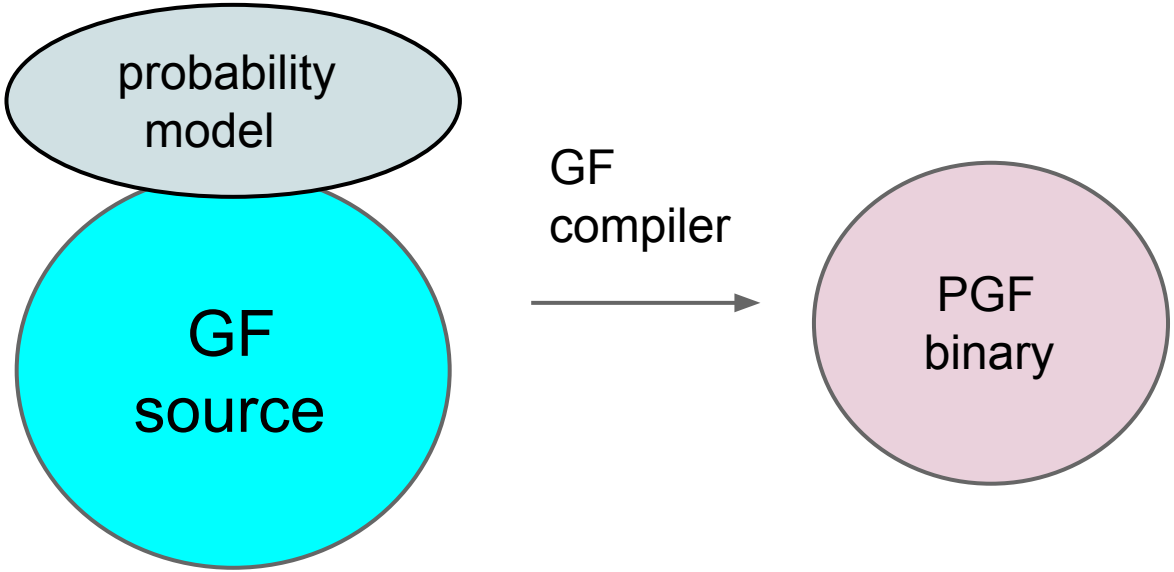
single words <---> entire sentences

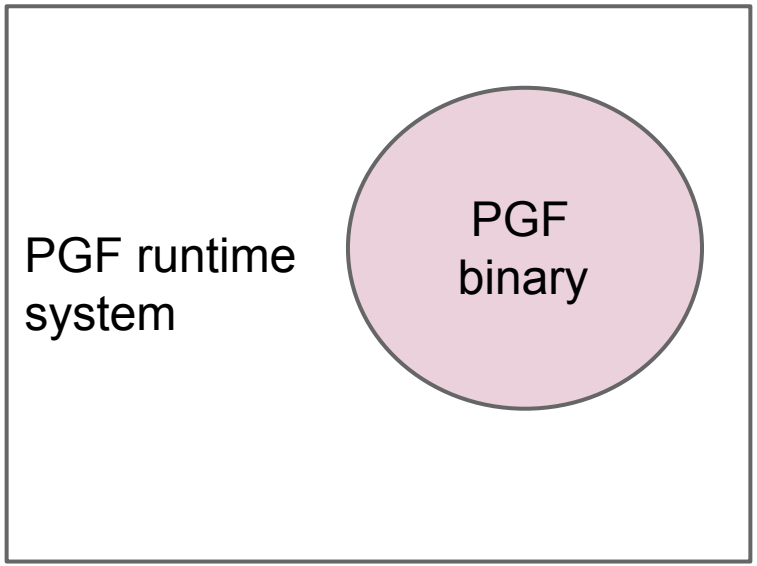
A wide-coverage chunking grammar can be built in a couple of hours **by using the RGL**.

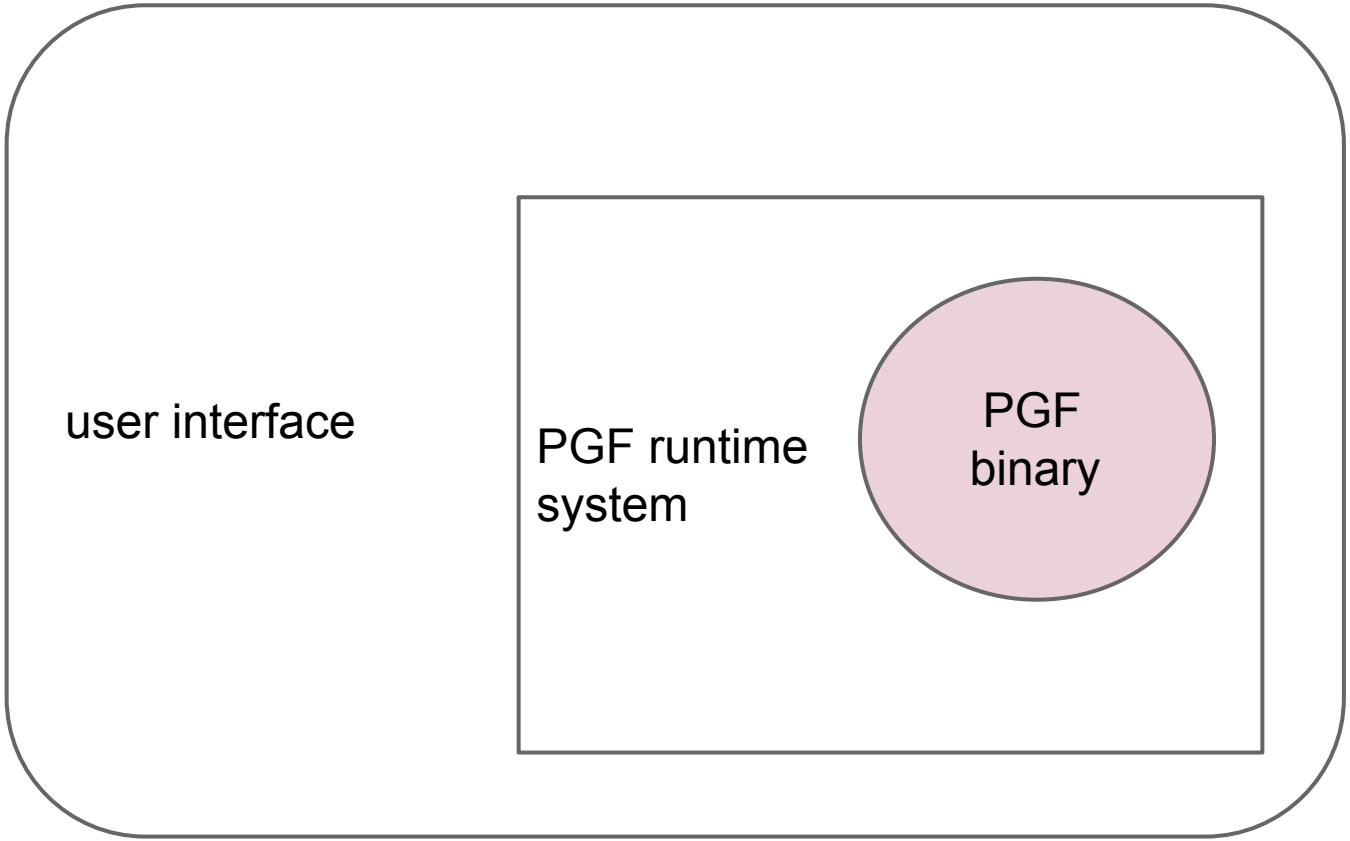
Building the translation system

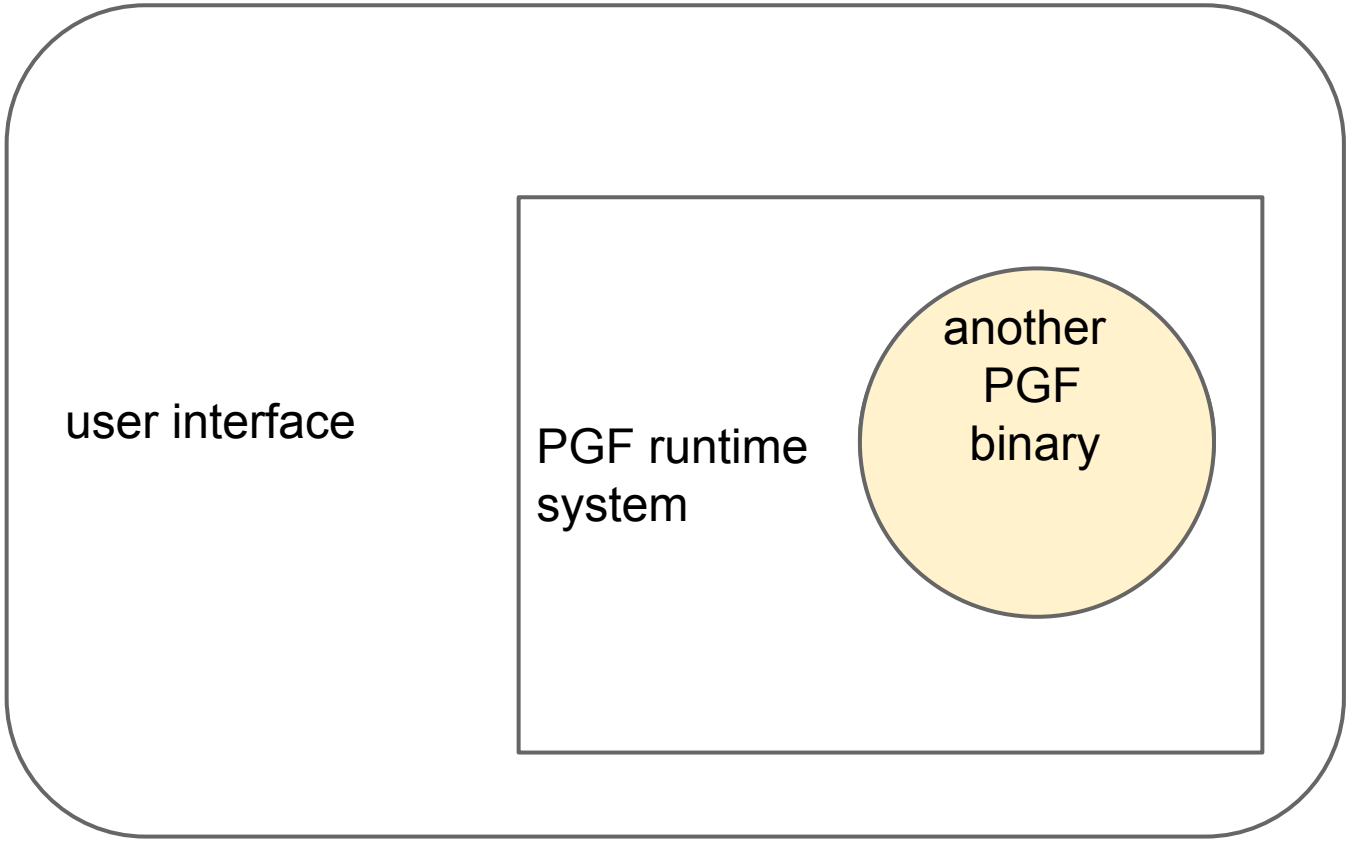

GF
source

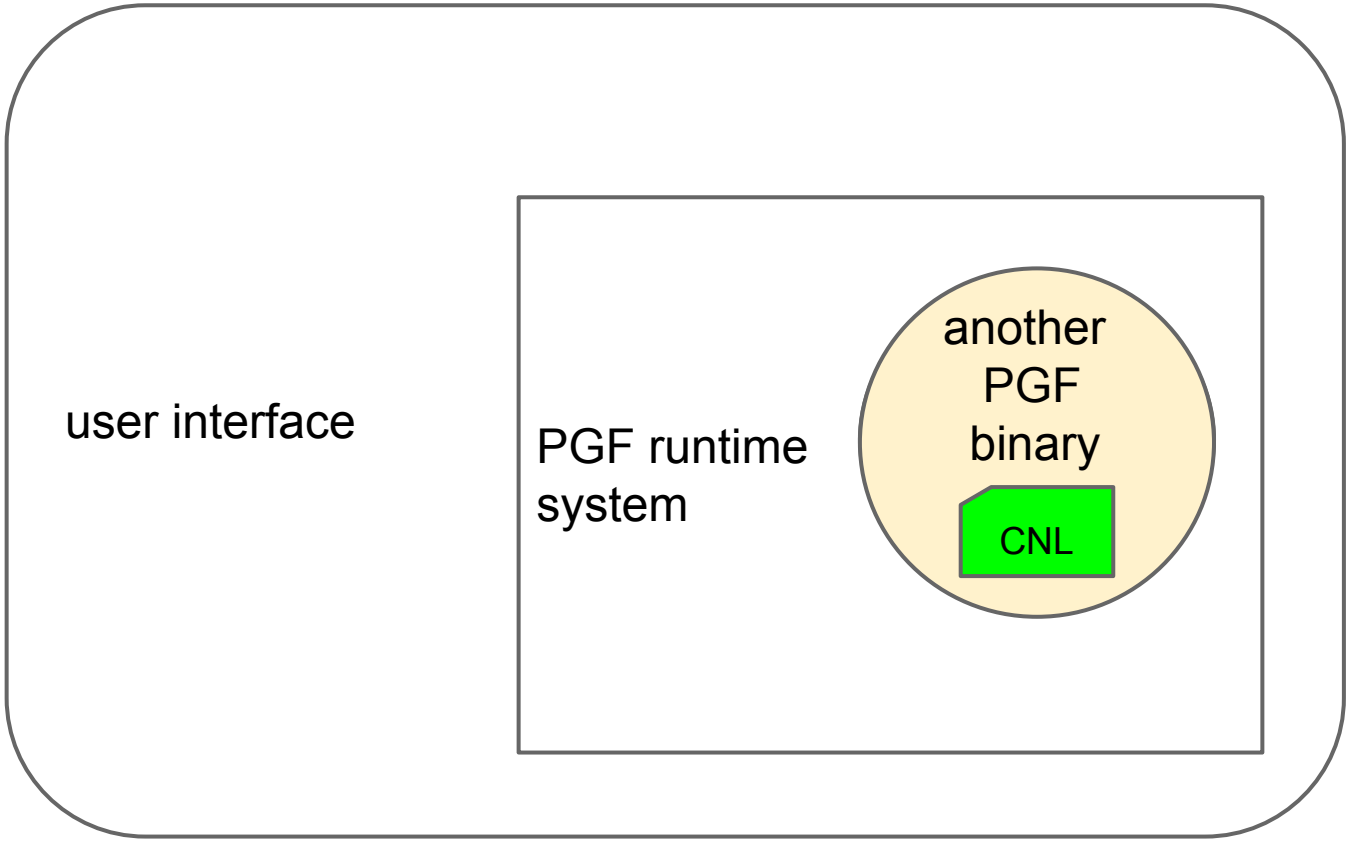

probability
model
GF
source

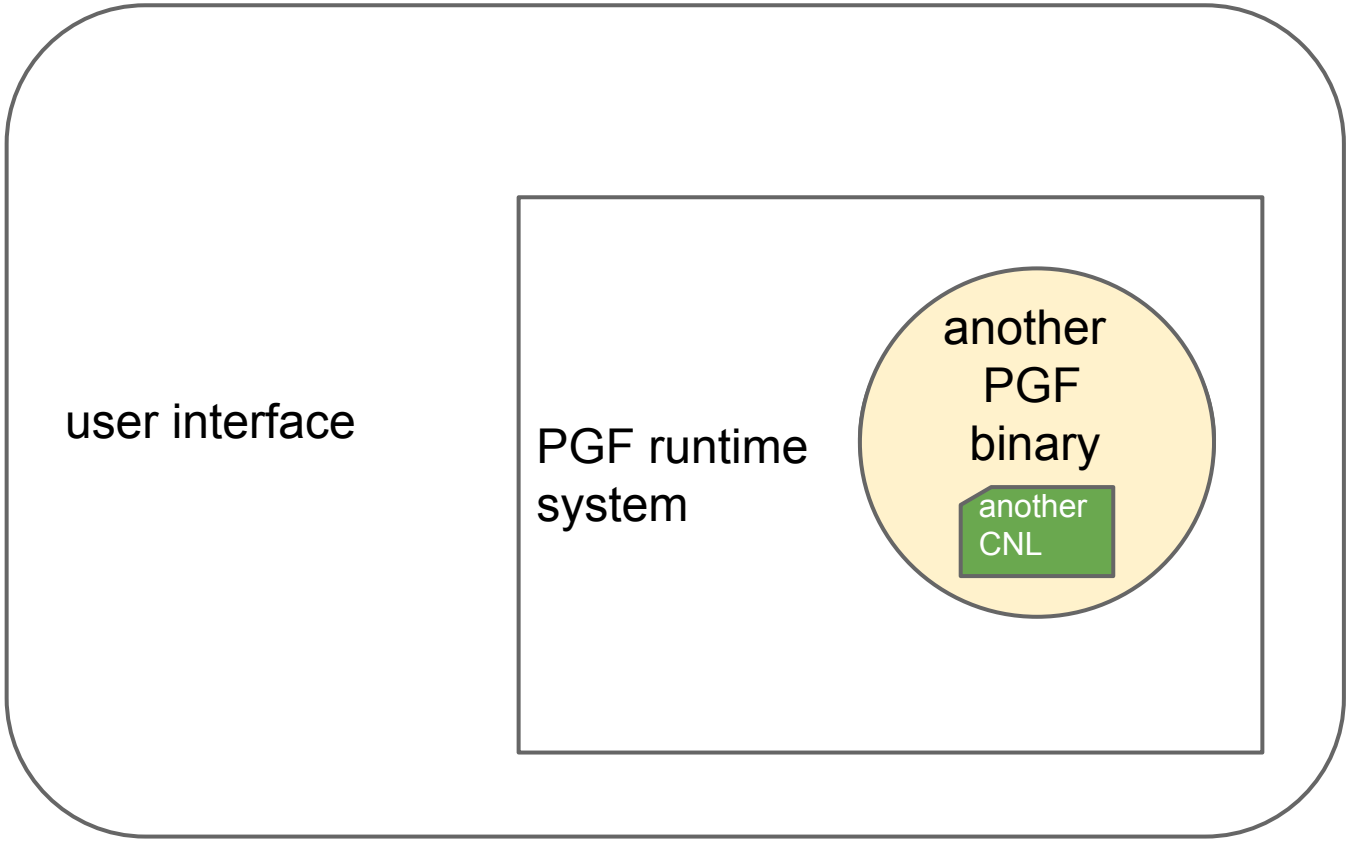

probability
model
GF
source
GF
compiler
PGF
binary


PGF runtime system
PGF binary

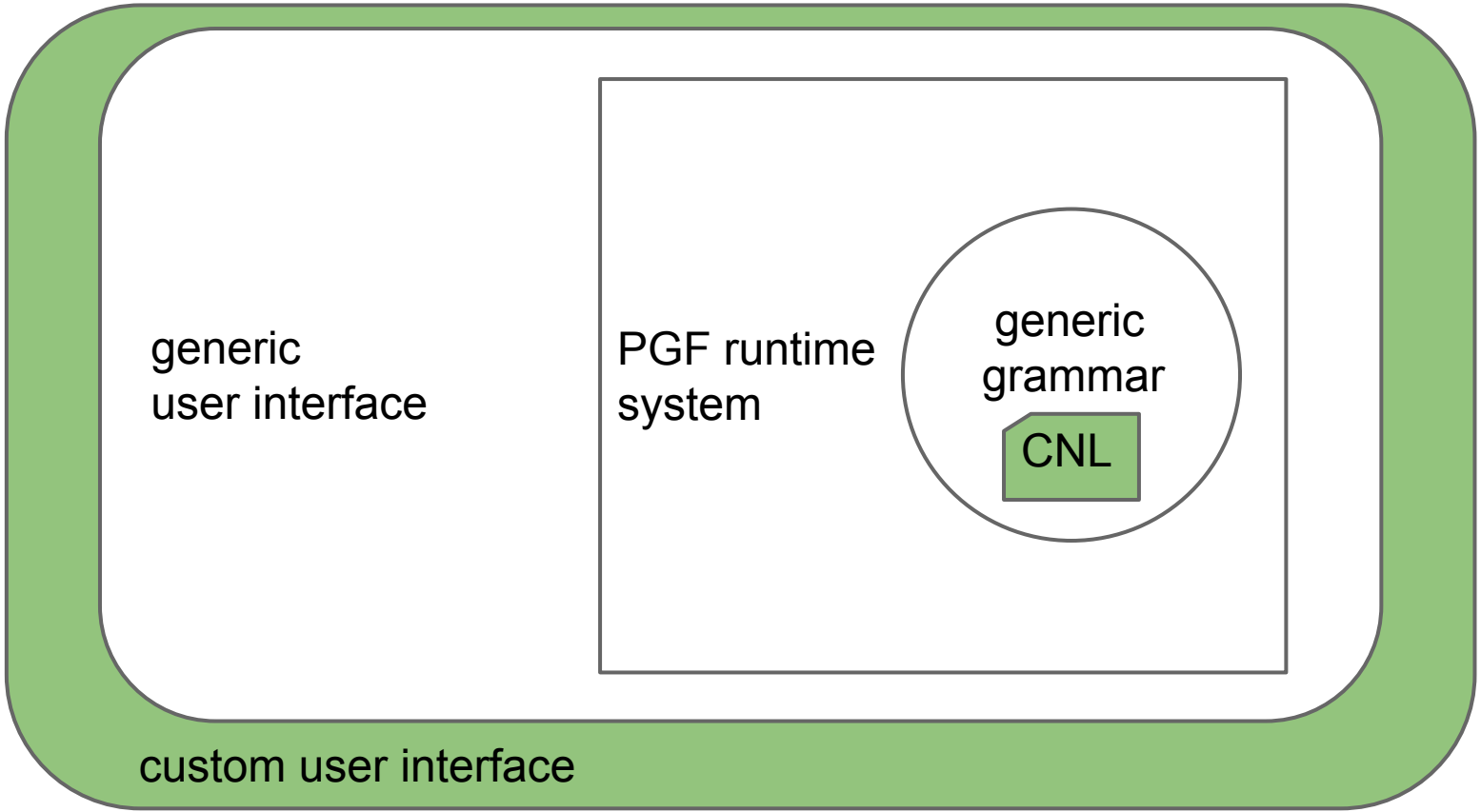

user interface
PGF runtime system
PGF binary


user interface
PGF runtime system
another PGF binary


user interface
PGF runtime system
another PGF binary
CNL


user interface
PGF runtime system
another PGF binary
another 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!