Machine Translation, Type Theory, Dependent Types

Aarne Ranta

TTR LOCI Workshop, London 16-17 June 2011 (Based on TYPES 2010, FreeRBMT 2011)

Plan

Machine Translation

Grammatical Framework

Dependent Types

Machine Translation

Important research problems

(From Hamming, "You and your research")

What are the important problems in your field?

Are you working on one of them?

If not, why?

`http://www.paulgraham.com/hamming.html`

The important problems in computational linguistics

type-theoretical semantics

The important problems in computational linguistics

type-theoretical semantics

anaphora resolution

The important problems in computational linguistics

type-theoretical semantics

anaphora resolution

multilingual syntax editing

The important problems in computational linguistics

type-theoretical semantics

anaphora resolution

multilingual syntax editing

machine translation

Beginnings of machine translation

Weaver 1947, encouraged by cryptography in WW II

Word lookup $\longrightarrow$ n-gram models (Shannon's "noisy channel")

```
ê = argmax P(f|e)P(e)
      e
```

P(w1 ... wn) approximated by e.g. P(w1w2)P(w2w3)...P(w(n-1)wn) (2-grams)

Modern version: Google translate `translate.google.com`

Word sense disambiguation

Eng. *even* ⟶ Fre *égal*, *équitable*, *pair*, *plat* ; *même*, ...

Eng. *even number* ⟶ Fre *nombre pair*

Eng. *not even* ⟶ Fre *même pas*

Eng. *7 is not even* ⟶ Fre *7 n'est pas pair*

Eng. *7 is not even even* ⟶ Fre *7 n'est même pas pair*

Long-distance dependencies

Ger. *er bringt mich um* $\longrightarrow$ Eng. *he kills me*

Ger. *er bringt seinen besten Freund um* $\longrightarrow$ Eng. *he kills his best friend*

Type theory and machine translation

Bar-Hillel (1953): MT should aim at rendering *meaning*, not words.

Method: Ajdukiewicz syntactic calculus (1935) for syntax and semantics.

Directional types (prefix and postfix functions)

```
             loves : (n\s)n    Mary : n
             --------------------------
  John : n      loves Mary : n\s
  ------------------------------
      John loves Mary : s
```

Categorial grammar, developed further by Lambek (1958), Curry (1961)

Bar-Hillel's criticism

1963: FAHQT (Fully Automatic High-Quality Translation) is impossible - not only in foreseeable future but in principle.

Example: word sense disambiguation for *pen*:

the pen is in the box vs. *the box is in the pen*

Requires unlimited intelligence, universal encyclopedia.

1970's and 1980's

Trade-off: coverage vs. precision

Precision-oriented systems: Curry $\longrightarrow$ Montague $\longrightarrow$ Rosetta

Interactive systems (Kay 1979/1996)

- ask for disambiguation if necessary
- text editor + translation memory

Present day

IBM system (Brown, Jelinek, & al. 1990): back to Shannon's model

Google translate 2007- (Och, Ney, Koehn, ...)

- 57 languages
- models built automatically from text data

Browsing quality rather than *publication quality*

Systran/Babelfish: rule-based, since 1960's

Apertium (2005-): rule-based, closely related languages

MOLTO **Multilingual Online Translation**

Non multa, sed multum not quantity but quality

ABOUT NEWS EVENTS

MOLTO's mission is to develop a set of tools for translating texts between *multiple languages* in *real time* with *high quality*. MOLTO will use multilingual grammars based on semantic interlinguas.

FP7-ICT-247914, Strep, `www.molto-project.eu`

U Gothenburg, U Helsinki, UPC Barcelona, Ontotext (Sofia)

March 2010 - February 2013

What's new?

Tool	**Google, Babelfish**	**MOLTO**
target	consumers	producers
input	unpredictable	predictable
coverage	unlimited	limited
quality	browsing	publishing

Producer's quality

Cannot afford translating French

- *prix 99 euros*

to Swedish

- *pris 99 kronor*

Typical SMT error due to parallel corpus containing localized texts. (N.B. 99 kronor = 11 euros)

Reliability

German to English

- *er bringt mich um -> he is killing me*

correct, but

- *er bringt meinen besten Freund um -> he brings my best friend for*

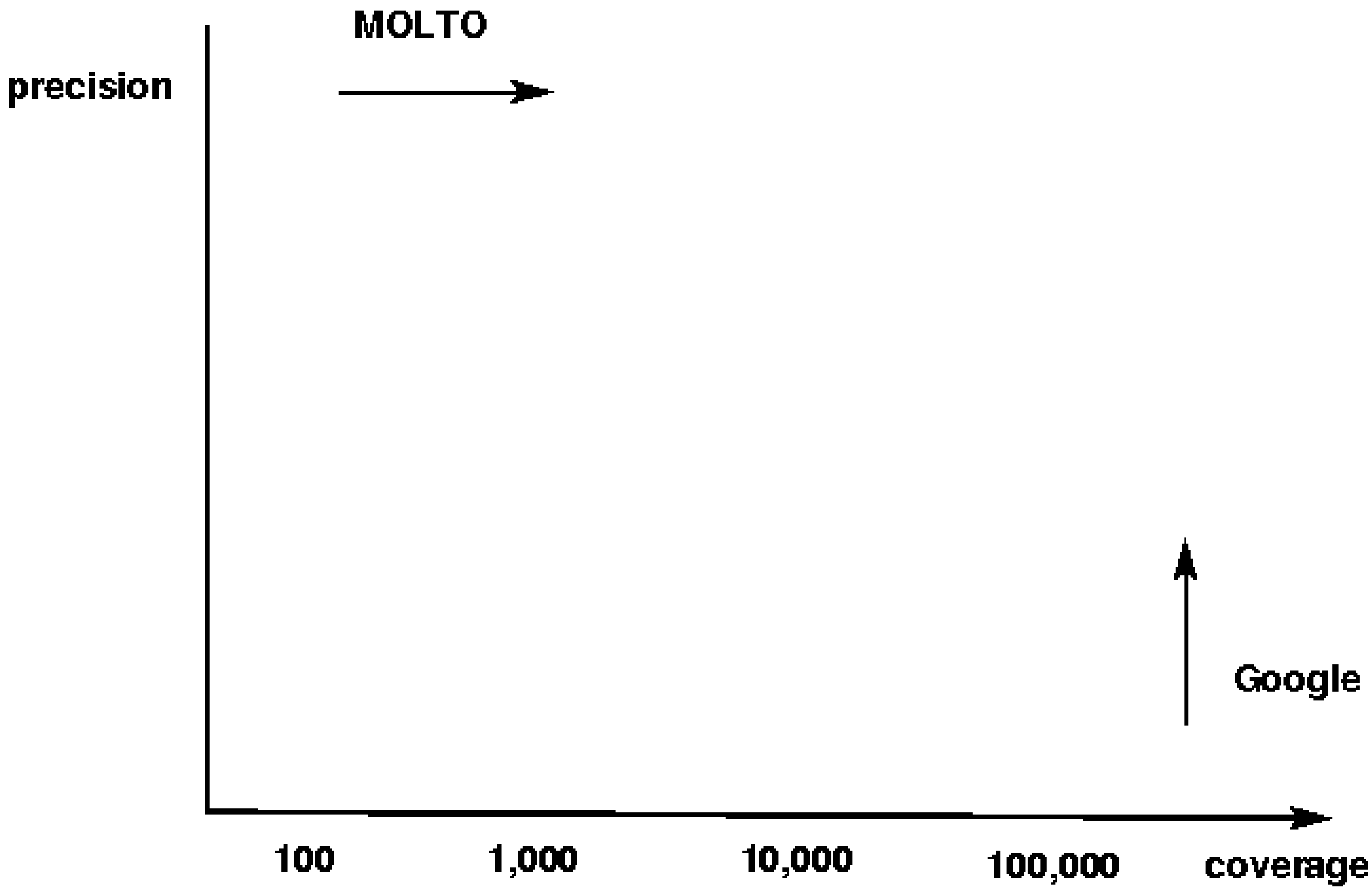
should be *he kills my best friend*. (Typical error due to **long distance dependencies**, causes **unpredictability**)

Aspects of reliability

Separation of levels (syntax, semantics, pragmatics, localization)

Predictability (generalization for similar constructs, and over time)

Programmability / debugging and fixing bugs (vs. holism)


MOLTO
precision
Google
100
1,000
10,000
100,000
coverage

The translation directions

Statistical methods (e.g. Google translate) work decently *to* English

- rigid word order
- simple morphology
- originates in projects funded by U.S. defence

Grammar-based methods work equally well for different languages

- Finnish cases
- German word order

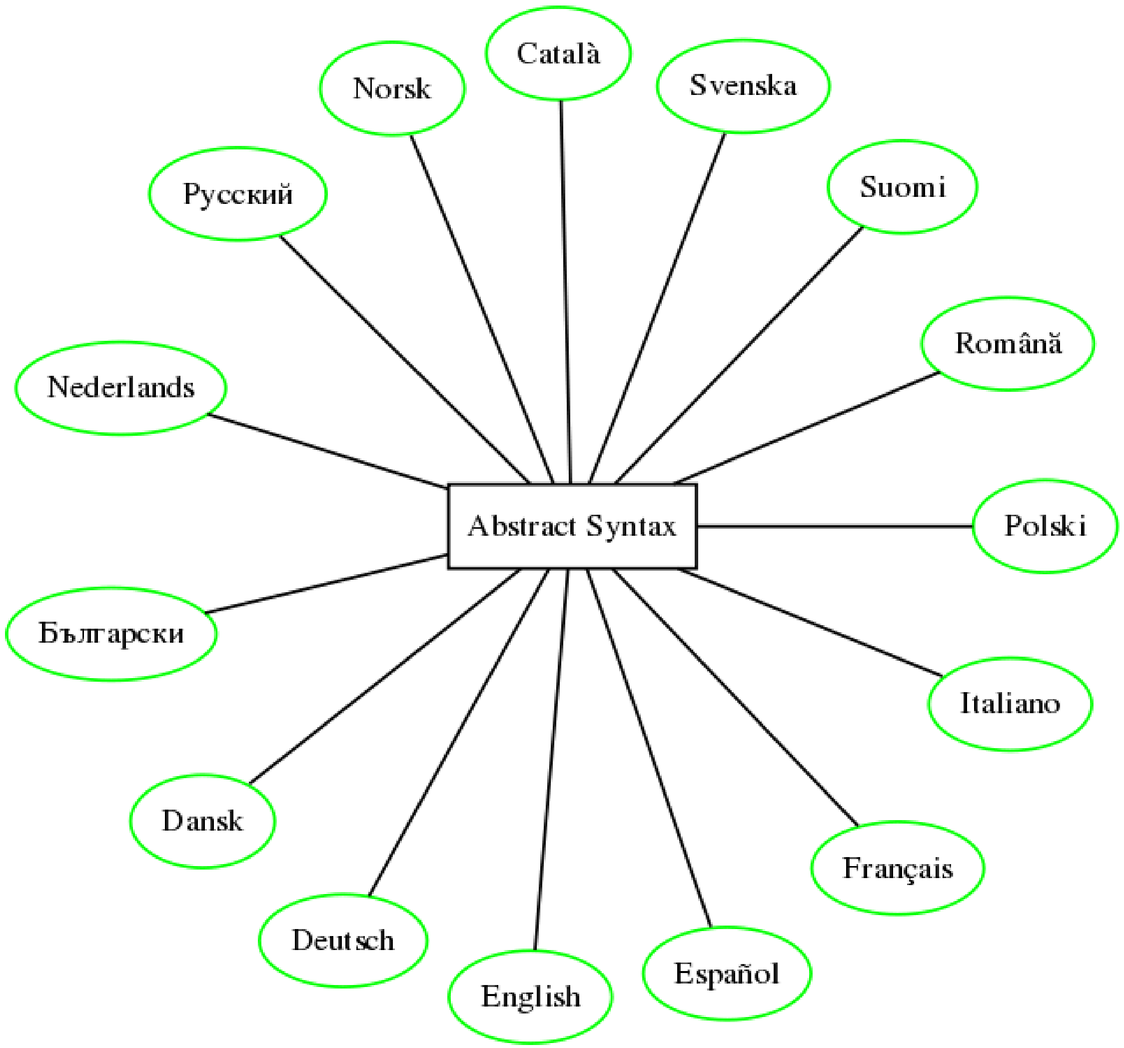
Main technologies

GF, grammaticalframework.org

- Domain-specific interlingua + concrete syntaxes
- GF Resource Grammar Library
- Incremental parsing
- Syntax editing

OWL Ontologies: resources for domain semantics

Statistical Machine Translation: robustness, grammar learning



MOLTO languages

Domain-specific interlinguas

The abstract syntax must be formally specified, well-understood

- semantic model for translation
- fixed word senses
- proper idioms

For instance: a mathematical theory, an ontology

Example: social network

Abstract syntax:

```
fun Like : Person -> Item -> Fact
```

Concrete syntax (first approximation):

```
lin Like x y = x ++ "likes" ++ y      -- Eng
lin Like x y = x ++ "tycker om" ++ y -- Swe
lin Like x y = y ++ "piace a" ++ x   -- Ita
```

Complexity of concrete syntax

Italian: agreement, rection, clitics (*il vino piace a Maria* vs. *il vino mi piace* ; *tu mi piaci*)

```
lin Like x y = y.s ! nominative ++ case x.isPron of {
  True  => x.s ! dative ++ piacere_V ! y.agr ;
  False => piacere_V ! y.agr ++ "a" ++ x.s ! accusative
  }
oper piacere_V = verbForms "piaccio" "piaci" "piace" ...
```

Moreover: contractions (*tu piaci ai bambini*), tenses, mood, ...

Two things we do better than before

No universal interlingua:

- *The Rosetta stone is not a monolith, but a boulder field.*

Yes universal concrete syntax:

- no hand-crafted *ad hoc* grammars
- but a general-purpose **Resource Grammar Library**

The GF Resource Grammar Library

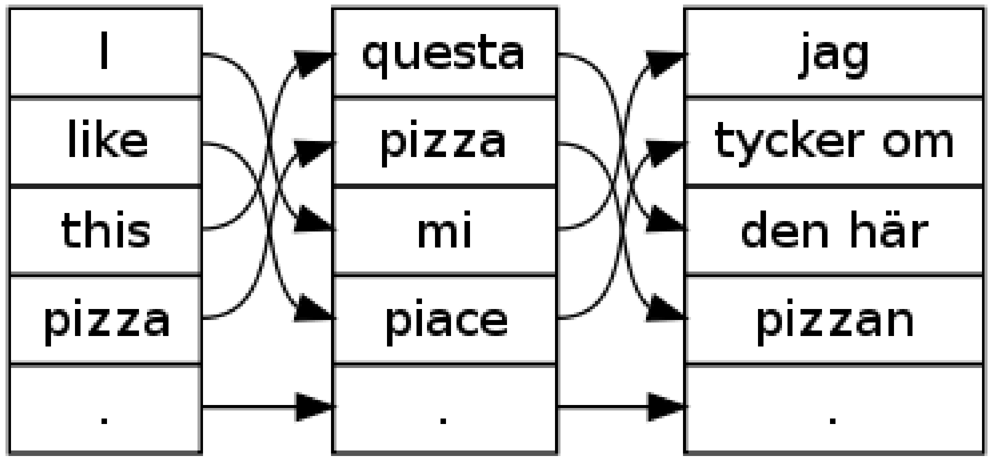
Currently for 16 languages; 3-6 months for a new language.

Complete morphology, comprehensive syntax, lexicon of irregular words.

Common syntax API:

```
lin Like x y = mkCl x (mkV2 (mkV "like")) y          -- Eng
lin Like x y = mkCl x (mkV2 (mkV "tycker") "om") y -- Swe
lin Like x y = mkCl y (mkV2 piacere_V dative) x    -- Ita
```

Word/phrase alignments via abstract syntax



Domains for case studies

Mathematical exercises (<- WebALT)

Patents in biomedical and pharmaceutical domain

Museum object descriptions

Demo: a tourist phrasebook (web and Android phones)

Other potential uses

Wikipedia articles

E-commerce sites

Medical treatment recommendations

Social media

SMS

Contracts

Challenge: grammar tools

Scale up production of domain interpreters

- from 100's to 1000's of words
- from GF experts to domain experts and translators
- from months to days
- writing a grammar $\approx$ translating a set of examples

Example-based grammar writing

Abstract syntax	`Like She He`	first grammarian
English example	*she likes him*	first grammarian
German translation	*er gefällt ihr*	human translator
resource tree	`mkCl he_Pron gefallen_V2 she_Pron`	GF parser
concrete syntax rule	`Like x y = mkCl y gefallen_V2 x`	variables renamed

Learning GF grammars by statistics

Abstract syntax	`Like She He`	first grammarian
English example	*she likes him*	first grammarian
German translation	*er gefällt ihr*	**SMT system**
resource tree	`mkCl he_Pron gefallen_V2 she_Pron`	GF parser
concrete syntax rule	`Like x y = mkCl y gefallen_V2 x`	variables renamed

Rationale: SMT is *good* for sentences that are *short* and *frequent*

Improving SMT by grammars

Rationale: SMT is *bad* for sentences that are *long* and involve *word order variations*

if you like me, I like you

`If (Like You I) (Like I You)`

wenn ich dir gefalle, gefällst du mir

Grammatical Framework

History

Background: type theory, logical frameworks (LF), compilers

GF = LF + concrete syntax

Started at Xerox (XRCE Grenoble) in 1998 for **multilingual document authoring**

Functional language with dependent types, parametrized modules, optimizing compiler

Run-time: Parallel Multiple Context-Free Grammar, polynomial

Factoring out functionalities

GF grammars are declarative programs that define

- parsing
- generation
- translation
- editing

Some of this can also be found in BNF/Yacc, HPSG/LKB, LFG/XLE
...

A model for reliable automatic translation: compilers

Translate source code to target code, *preserving meaning*

Method: parsing, semantic analysis, optimization, code generation

Multilingual grammars in compilers

Source and target language related by abstract syntax

```
                                                  iconst_2
                                                  iload_0
2 * x + 1  <----->  plus (times 2 x) 1  <------>  imul
                                                  iconst_1
                                                  iadd
```

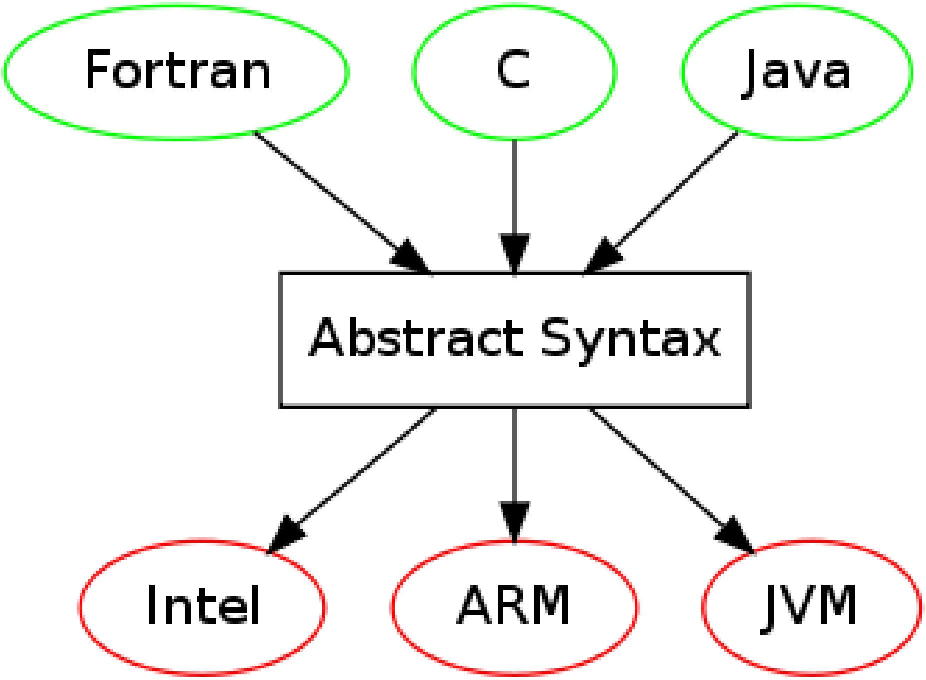
A GF grammar for arithmetic expressions

```
abstract Expr = {
  cat Exp ;
  fun plus : Exp -> Exp -> Exp ;
  fun times : Exp -> Exp -> Exp ;
  fun one, two : Exp ;
  }
```

```
concrete ExprJava of Expr = {
  lincat Exp = Str ;
  lin plus x y = x ++ "+" ++ y ;
  lin times x y = x ++ "*" ++ y ;
  lin one = "1" ;
  lin two = "2" ;
  }
```

```
concrete ExprJVM of Expr= {
  lincat Expr = Str ;
  lin plus x y = x ++ y ++ "iadd" ;
  lin times x y = x ++ y ++ "imul" ;
  lin one = "iconst_1" ;
  lin two = "iconst_2" ;
  }
```

Multi-source multi-target compilers



Multilingual grammars in natural language

```
Mary loves John                                     Maria Ioannem amat
                \                               /
                  Pred Mary (Compl Love John)
                /                               \
Marie aime Jean                                     מרי אוהבת את ג׳ון
```

Natural language structures

Predication: *John* + *loves Mary*

Complementation: *love* + *Mary*

Noun phrases: *John*

Verb phrases: *love Mary*

2-place verbs: *love*

Abstract syntax of sentence formation

```
abstract Zero = {
  cat
    S ; NP ; VP ; V2 ;
  fun
    Pred  : NP -> VP -> S ;
    Compl : V2 -> NP -> VP ;
    John, Mary : NP ;
    Love : V2 ;
}
```

Concrete syntax, English

```
concrete ZeroEng of Zero = {
  lincat
    S, NP, VP, V2 = Str ;
  lin
    Pred np vp = np ++ vp ;
    Compl v2 np = v2 ++ np ;
    John = "John" ;
    Mary = "Mary" ;
    Love = "loves" ;
}
```

Multilingual grammar

The same system of trees can be given

- different words
- different word orders
- different linearization types

Concrete syntax, French

```
concrete ZeroFre of Zero = {
  lincat
    S, NP, VP, V2 = Str ;
  lin
    Pred np vp = np ++ vp ;
    Compl v2 np = v2 ++ np ;
    John = "Jean" ;
    Mary = "Marie" ;
    Love = "aime" ;
}
```

Just use different words

Translation and multilingual generation in GF

Import many grammars with the same abstract syntax

```
> i ZeroEng.gf ZeroFre.gf
Languages: ZeroEng ZeroFre
```

Translation: pipe parsing to linearization

```
> p -lang=ZeroEng "John loves Mary" | l -lang=ZeroFre
Jean aime Marie
```

Multilingual random generation: linearize into all languages

```
> gr | l
Pred Mary (Compl Love Mary)
Mary loves Mary
Marie aime Marie
```

Parameters in linearization

Latin has *cases*: nominative for subject, accusative for object.

- *Ioannes Mariam amat* "John-Nom loves Mary-Acc"
- *Maria Ioannem amat* "Mary-Nom loves John-Acc"

Parameter type for case (just 2 of Latin's 6 cases):

```
param Case = Nom | Acc
```

Concrete syntax, Latin

```
concrete ZeroLat of Zero = {
  lincat
    S, VP, V2 = Str ;
    NP = Case => Str ;
  lin
    Pred  np vp = np ! Nom ++ vp ;
    Compl v2 np = np ! Acc ++ v2 ;
    John = table {Nom => "Ioannes" ; Acc => "Ioannem"} ;
    Mary = table {Nom => "Maria" ; Acc => "Mariam"} ;
    Love = "amat" ;
  param
    Case = Nom | Acc ;
}
```

Different word order (SOV), different linearization type, parameters.

Table types and tables

The linearization type of `NP` is a **table type**: from `Case` to `Str`,

```
lincat NP = Case => Str
```

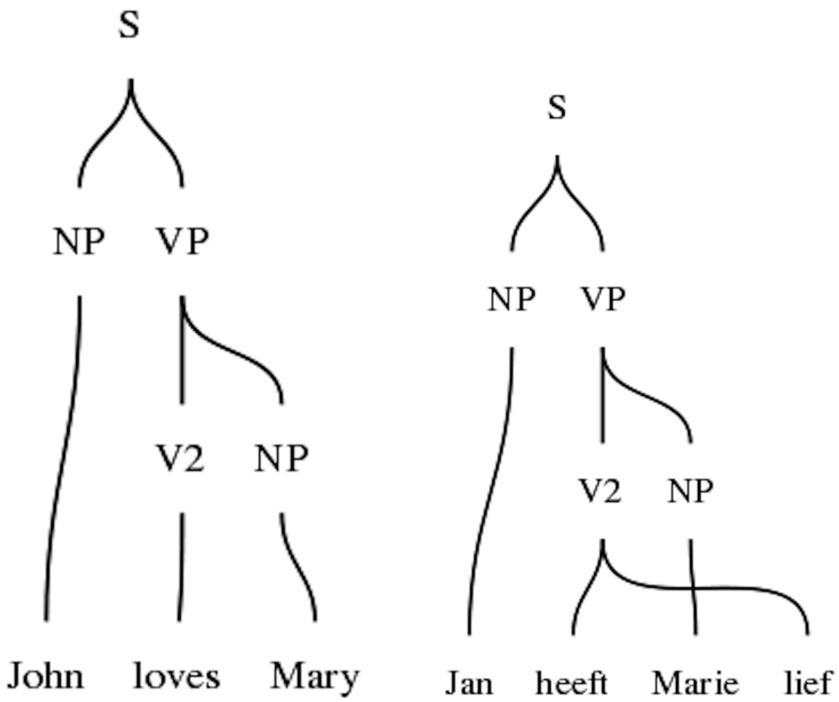
The linearization of `John` is an **inflection table**,

```
lin John = table {Nom => "Ioannes" ; Acc => "Ioannem"}
```

When using an NP, **select** (`!`) the appropriate case from the table,

```
Pred  np vp = np ! Nom ++ vp
Compl v2 np = np ! Acc ++ v2
```

Love in Dutch



Concrete syntax, Dutch

```
concrete ZeroDut of Zero = {
  lincat
    S, NP, VP = Str ;
    V2 = {v : Str ; p : Str} ;
  lin
    Pred np vp = np ++ vp ;
    Compl v2 np = v2.v ++ np ++ v2.p ;
    John = "Jan" ;
    Mary = "Marie" ;
    Love = {v = "heeft" ; p = "lief"} ;
}
```

The verb *heeft lief* is a **discontinuous constituent**.

Record types and records

The linearization type of `V2` is a **record type**

```
lincat V2 = {v : Str ; p : Str}
```

The linearization of `Love` is a **record**

```
lin Love = {v = "heeft" ; p = "lief"}
```

The values of fields are picked by **projection** (.)

```
lin Compl v2 np = v2.v ++ np ++ v2.p
```

Concrete syntax, Hebrew

```
concrete ZeroHeb of Zero = {
   flags coding=utf8 ;
 lincat
   S = Str ;
   NP = {s : Str ; g : Gender} ;
   VP, V2 = Gender => Str ;
 lin
   Pred np vp = np.s ++ vp ! np.g ;
   Compl v2 np = table {g => v2 ! g ++ "את" ++ np.s} ;
   John = {s = "ג'ון" ; g = Masc} ;
   Mary = {s = "מרי" ; g = Fem} ;
   Love = table {Masc => "אוהב" ; Fem => "אוהבת"} ;
 param
   Gender = Masc | Fem ;
}
```

The verb **agrees** to the gender of the subject.

Abstract trees vs. parse trees

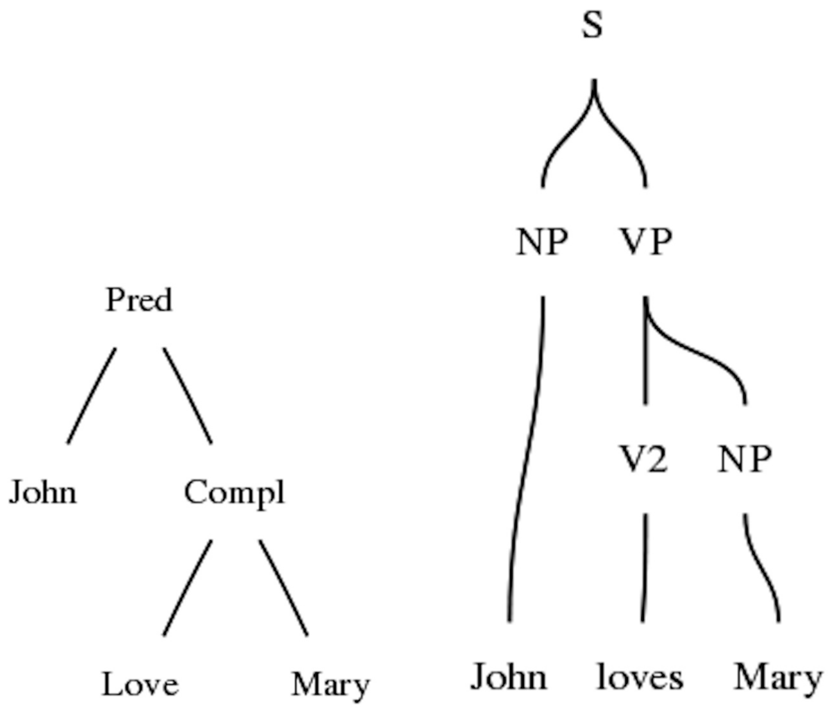
Abstract trees

- nodes: constructor functions
- leaves: constructor functions

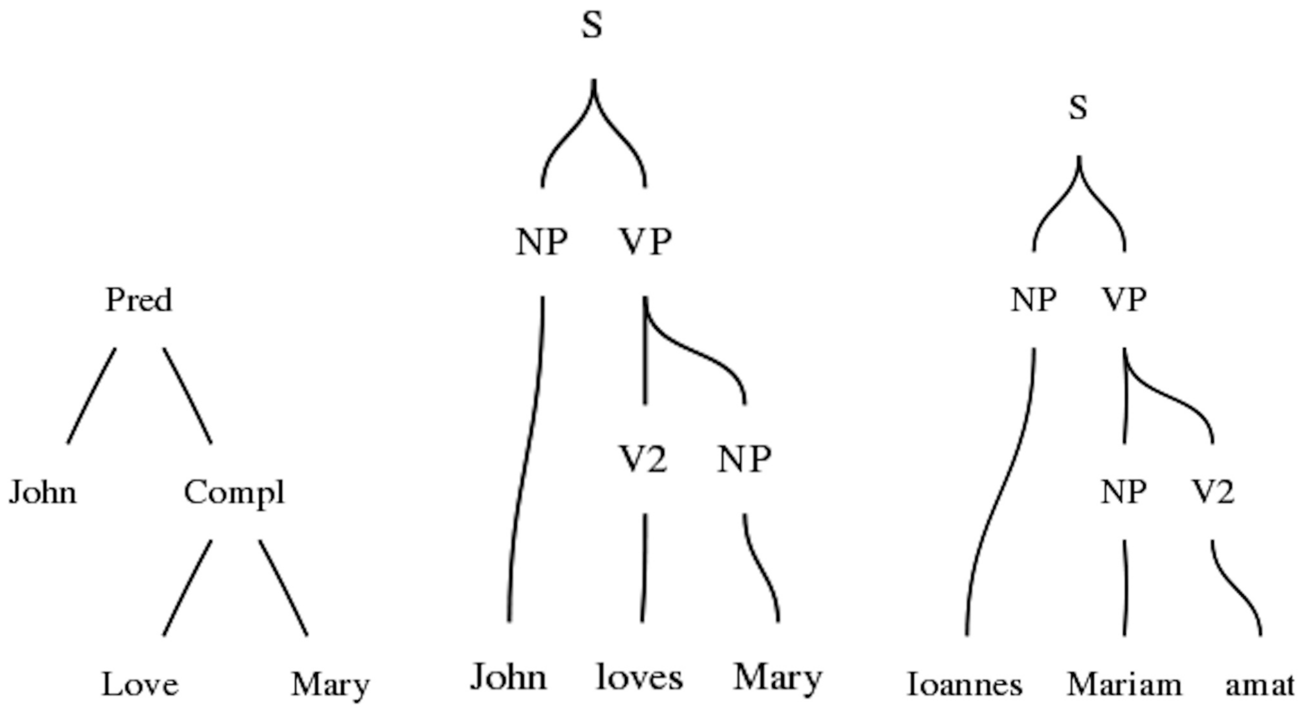
Parse trees

- nodes: categories
- leaves: words

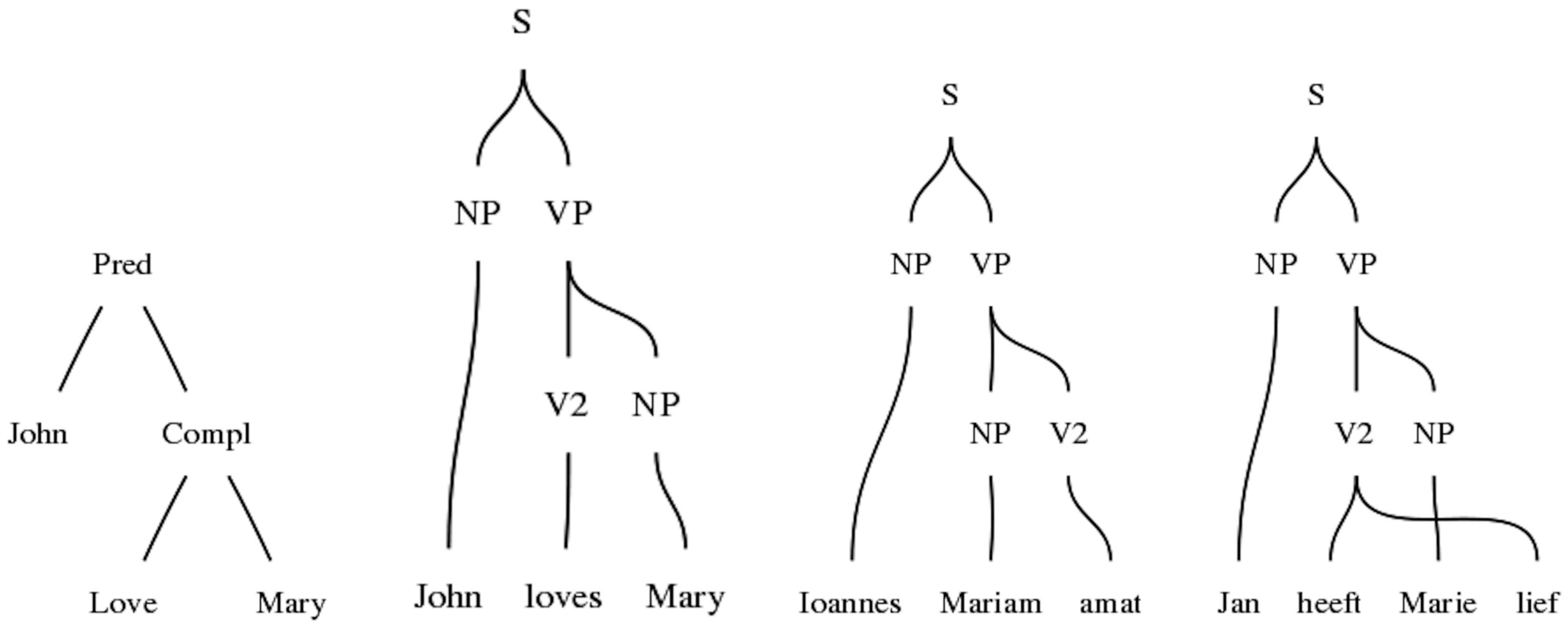
Abstract is more abstract



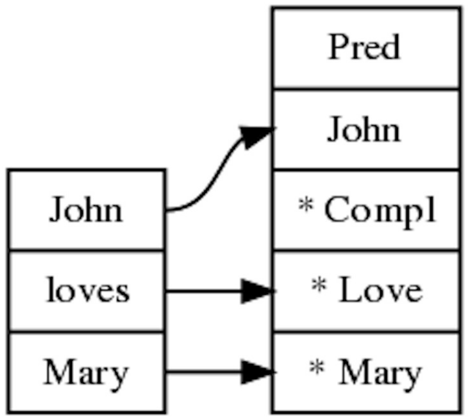
Abstract is more abstract



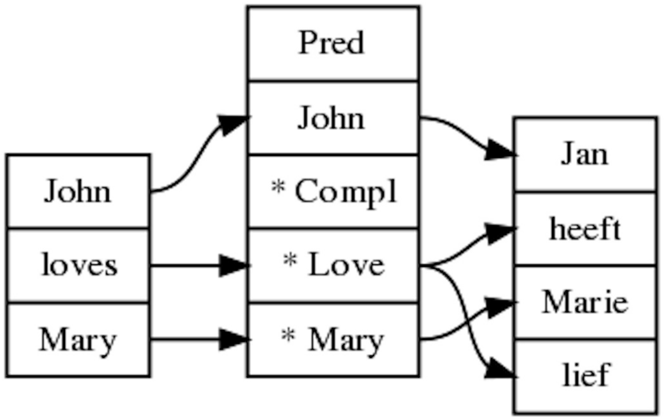
Abstract is more abstract



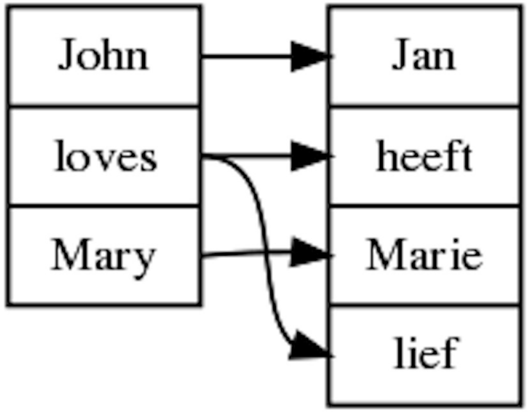
From trees to words



From words to trees to words



From words to words



Generating word alignment: summary

In L1 and L2: link every word with its smallest spanning subtree

Delete the intervening tree, combining links directly from L1 to L2

Notice: in general, this gives **phrase alignment**

Notice: links can be crossing, phrases can be discontinuous

Complexity of grammar writing

To implement a translation system, we need

- domain expertise: technical and idiomatic expression
- linguistic expertise: how to inflect words and build phrases

The GF Resource Grammar Library

Morphology and basic syntax

Common API for different languages

Currently (June 2011) 19 languages: Afrikaans, Bulgarian, Catalan, Danish, Dutch, English, Finnish, French, German, Italian, Norwegian, Persian, Polish, Punjabi, Romanian, Russian, Spanish, Swedish, Urdu.

Under construction for more languages: Amharic, Arabic, Hindi, Irish, Latin, Latvian, Nepali, Swahili, Thai, Turkish.

Contributions welcome!

The scope of resource grammars

Morphology: all inflectional forms and paradigms

Syntax: basic syntax, "complete in expressive power" (cf. CLE)

Lexicon:

- multilingual test lexicon of 500 words (structural and irregular; Swadesh)
- comprehensive monolingual for Bulgarian, English, Finnish, Swedish, Turkish

Inflectional morphology

Goal: a complete system of inflection paradigms

Paradigm: a function from "basic form" to full inflection table

GF morphology is inspired by

- Zen (Huet 2005): typeful functional programming
- XFST (Beesley and Karttunen 2003): regular expressions

Smart paradigm, implementor's view

Help the lexicographers work by **pattern matching on strings**

```
regV : Str -> V = \v -> case v of {
  fi + ("s"|"z"|"x"|"ch"|"sh") => mkV v (v + "es") (v + "ed") (v  + "ing") ;
  d  + "ie"                    => mkV v (v + "s")  (v + "d")  (d  + "ying") ;
  fr + "ee"                    => mkV v (v + "s")  (v + "d")  (v  + "ing") ;
  us + "e"                     => mkV v (v + "s")  (v + "d")  (us + "ing") ;
  pl + ("a"|"e"|"o"|"u") + "y" => mkV v (v + "s")  (v + "ed") (v  + "ing") ;
  cr + "y"                     => mkV v (cr + "ies") (cr + "ied") (v + "ing") ;
  dr + o@(#vowel) + p@(#cons)  => mkV v (v + "s")  (v + p + "ed") (v + p + "ing") ;
  _                            => mkV v (v + "s")  (v + "ed") (v  + "ing") ;
    } ;
```

Morphology API

Overloaded function, heuristic variables for arguments

```
mkV : (fix : Str) -> V
mkV : (sing, sang, sung : Str) -> V

mkN : (bunch : Str) -> N
mkN : (man, men : Str) -> N
```

This is how the lexicon looks

Principle: just the minimum of information given (POS, characteristic forms)

```
mkN "boy"
mkV "cut" "cut" "cut"
mkV "drop"
mkA "happy"
mkN "mouse" "mice"
mkV "munch"
mkV "sing" "sang" "sung"
mkV "try"
```

This scales up

In Finnish, nouns have 30 forms.

- 85% need only one form
- 1.42 is the average

Finnish verbs with hundreds of forms need an average of 1.2 forms.

Syntax API

Combination rules

```
mkCl : NP -> V2 -> NP -> Cl    -- John loves Mary
mkNP : Numeral  -> CN -> NP    -- five houses
```

Structural words

```
the_Det   : Det
youSg_NP : NP
```

Meaning-preserving translation

Translation must preserve meaning.

It need not preserve syntactic structure.

Sometimes this is even impossible:

- *John likes Mary* in Italian is *Maria piace a Giovanni*

The abstract syntax in the semantic grammar is a logical predicate:

```
fun Like : Person -> Item -> Fact
lin Like x y = x ++ "likes" ++ y           -- English
lin Like x y = y ++ "piace" ++ "a" ++ x  -- Italian
```

Translation and resource grammar

To get all grammatical details right, we use resource grammar and not strings

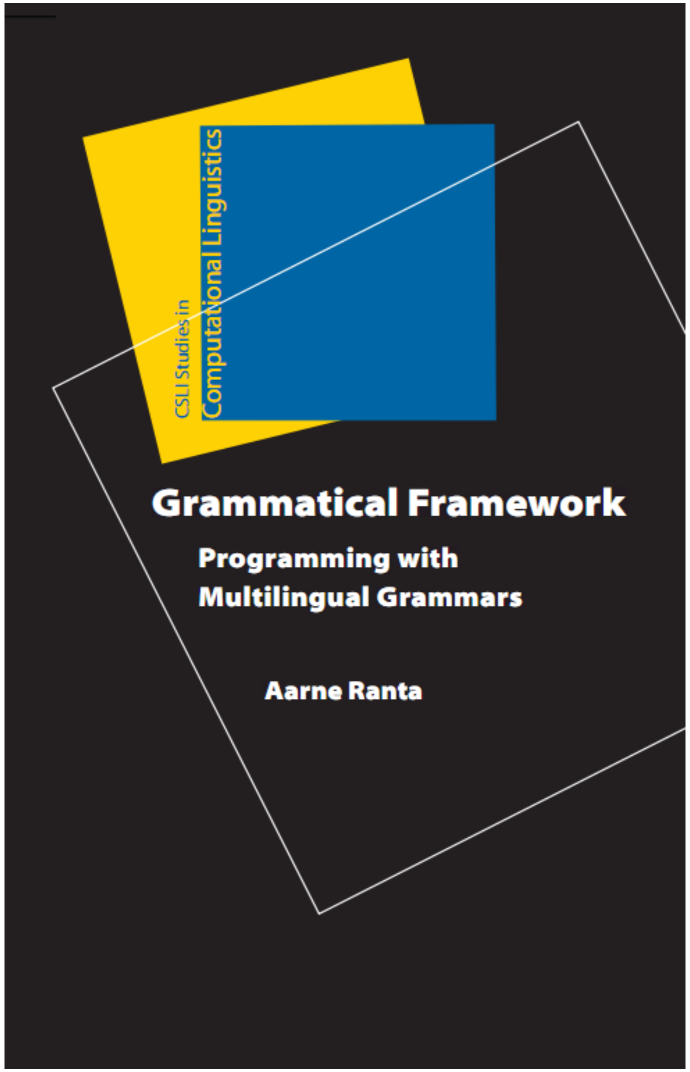
```
lincat Person, Item = NP ; Fact = Cl ;

lin Like x y = mkCl x like_V2 y      -- English
lin Like x y = mkCl y piacere_V2 x  -- Italian
```

From syntactic point of view, we perform **transfer**, i.e. structure change.

GF has **compile-time transfer**, and uses interlingua (semantic abstrac syntax) at run time.

More on GF

GF homepage, http://grammaticalframework.org



A. Ranta, *Grammatical Framework: Programming with Multilingual Grammars*, CSLI Publications, Stanford, 2011, ISBN 1-57586-626-9.

Registration open till 30 June: http://school.grammaticalframework.org

Dependent Types

Semantics: well-typedness

Domain-dependent categories

```
cat Dom ; NP Dom ; VP Dom ; S

fun Pred : (d : Dom) -> NP d -> VP d -> S
```

Uses

- word sense disambiguation
- better generation of synthetic corpora

Generalization of well-typedness: type classes

Proof objects establish class membership

```
cat Dom ; Animate Dom

fun
  Sleep : (d : Dom) -> Animate d -> VP d
  Man, Donkey : Dom
  ManIsAnimate : Animate Man
  DonkeyIsAnimate : Animate Donkey
```

Notice: this may well be language dependent, e.g. German *essen* - *fressen* "eat"

Another generalization of well-typedness: coercive subtyping

Proof objects establish subtype relation

```
cat Dom ; Subtype Dom Dom

fun
  Pred : (d,e : Dom) -> Subtype d e -> NP d -> VP e -> S
  Human, Teacher : Dom
  TeacherIsHuman : Subtype Teacher Human
```

Semantics: anaphora

the monkey ate the banana because **it** *was hungry* - **er** *war hungrig*

the monkey ate the banana because **it** *was ripe* - **sie** *war reif*

the monkey ate the banana because **it** *was tea-time* - **es** *war Teezeit*

The grammar of pronouns

Simplified German:

```
  fun Pron : (t : Typ) -> Ref t -> Exp t

  lin Pron t _ = case (gender t) of {
    Masc  => "er" ;
    Fem   => "sie" ;
    Neutr => "es"
    }
```

Parsing English *it* creates the tree `Pron ?1 ?2`.

Algorithm

1. Analyse the **context** to form the **referent space** {r1 : R1,...,rn : Rn}.
2. Collect all types {T1,...,Tm} that an object may have in the position of the pronoun.
3. Consider the set of those elements ri : Ri whose type Ri matches some of the types Tj.
   (a) If the set is singleton {ri : Ri}, then ri is the referent and its type is Ri.
   (b) If the set is empty, then report an anaphora resolution error (or widen the referent space).
   (c) If the set has many elements, then ask the user to disambiguate (or look for more constraints).

Syntax: agreement

Agreement *could* be modelled by

```
fun Pred : (a : Agr) -> NP a -> VP a -> S
```

However, we find it better to model agreement in *concrete syntax*

Syntax: subcategorization

Instead of

```
ComplV1 : V1 -> VP          -- sleep
ComplV2 : V2 -> NP -> VP    -- love
ComplVS : VS -> S  -> VP    -- believe
```

one could have

```
Compl : (s : Subcat) -> V s -> Comps s -> VP
```

However, the saving is marginal, since one has to define `Subcat` and `Comps` with as many rules.

Syntax: coordination

Rule: V2 coordination requires common complement case/preposition

```
ConjV2 : Conj -> (c : Case) -> V2 c -> V2 c -> V2 c
```

This is the *only* rule known to us that requires the use of language-specific features in concrete syntax.

Conclusion

You shouldn't expect

- general-purpose translation ("Google competitor")

You can expect

- high quality multilingual translation
- portability to limited domains (up to 1000's of words)
- productivity (days, weeks, months)
- ease of use (no training for authoring, a few days for grammarians)

Dependent types: used minimally so far, mostly for disambiguation.