Proof Objects for the Languages of the World

Aarne Ranta
aarne@chalmers.se

Truth Makers and Proof Objects, ENS, Paris, 23–25 November 2011

Plan

Type theory and semantics

Semantics and syntax

Multilinguality

What is semantics

Type theory and semantics

Donkey sentences

Geach 1962,

if a man owns a donkey he beats it

Stoics (Diogenes Laertios, acc. to Egli).

if a man is in Rhodes he is not in Athens

Problem with compositionality

a	$\exists$
if	$\supset$
a man is in Rhodes	$(\exists x)\text{Rhodes}(x)$

We get : $(\exists x)\text{Rhodes}(x) \supset \text{notAthens}(??)$
We want : $(\forall x)(\text{Rhodes}(x) \supset \text{notAthens}(x))$

Sigma and Pi

Martin-Löf, *Intuitionistic Type Theory*, 1984.

$$\frac{A : \mathsf{prop} \quad \begin{array}{c}(x : A)\\ B : \mathsf{prop}\end{array}}{(\Sigma x : A)B(x) : \mathsf{prop}} \qquad \frac{a : A \;\; b : B(a)}{(a, b) : (\Sigma x : A)B(x)} \qquad \frac{c : (\Sigma x : A)B(x)}{p(c) : A} \qquad \frac{c : (\Sigma x : A)B(x)}{q(c) : B(p(c))}$$

$$\frac{A : \mathsf{prop} \quad \begin{array}{c}(x : A)\\ B : \mathsf{prop}\end{array}}{(\Pi x : A)B(x) : \mathsf{prop}} \qquad \frac{\begin{array}{c}(x : A)\\ b : B(x)\end{array}}{(\lambda x)b(x) : (\Pi x : A)B(x)} \qquad \frac{c : (\Pi x : A)B(x) \;\; a : A}{Ap(c, a) : B(a)}$$

Progressive connectives

Martin-Löf 1984

$A \& B = (\Sigma x : A)B$ where B doesn't depend on $x : A$

$A \supset B = (\Pi x : A)B$ where B doesn't depend on $x : A$

Progressive conjunction and implication

$(x : A) \& B(x) = (\Sigma x : A)B(x)$ where $B(x)$ may depend on $x : A$

$(x : A) \supset B(x) = (\Pi x : A)B(x)$ where $B(x)$ may depend on $x : A$

Donkey sentences in type theory

a	Σ
if	Π

a man is in Rhodes

$$(\Sigma x : \mathsf{man})\mathsf{Rhodes}(x)$$

if a man is in Rhodes he is not in Athens

$$(\Pi z : (\Sigma x : \mathsf{man})\mathsf{Rhodes}(x))\mathsf{notAthens}(p(z))$$

The donkey sentence

a man owns a donkey

$$(\Sigma x : \mathsf{man})(\Sigma y : \mathsf{donkey})\mathsf{own}(x, y)$$

if a man owns a donkey he beats it

$$(\Pi z : (\Sigma x : \mathsf{man})(\Sigma y : \mathsf{donkey})\mathsf{own}(x, y))\mathsf{beat}(p(z), p(q(z)))$$

First observed: Mönnich 1985, Sundholm 1986.

Later developments: Ranta 1987, 1991, 1994; Fernando, Piwek, Cooper, Lecomte, Boldini.

Other uses of Sigma and Pi

Σ

- existential phrases: *a*, *some*
- conjunction: *and*
- relative: *such that*, *who*

Π

- universal phrases: *every*, *all*
- implication: *if*
- negation: *not* (*"if P, I will eat up my hat"*)

Prop = Set

A : Noun	<->	*there is a A* : Sentence
A : Sentence	<->	*proof that A* : Noun

Compositional paraphrases of the donkey sentence

$(\Pi z : (\Sigma x : \mathsf{man})(\Sigma y : \mathsf{donkey})\mathsf{own}(x, y))\mathsf{beat}(p(z), p(q(z)))$

if there is a man and there is a donkey and he owns it he beats it

if there is a man and he owns a donkey he beats it

if a man owns a donkey he beats it

every man who owns a donkey beats it

The spectrum of anaphoric expressions

given object	**expression**	**example**
$a : A$	Pron(A, a)	*he, it*
$a : A$	Def(A, a)	*the donkey, the man*
$a : A, b : B(b)$	DefMod(A, B, a, b)	*the man who owns the donkey*

Definition: $E(a) = a : A$ for each form E.

Examples

if a man owns a donkey... $z : (\Sigma x : \text{man})(\Sigma y : \text{donkey})\text{own}(x, y)$

- *he*, Pron(Man,$p(z)$)
- *the man*, Def(Man,$p(z)$)
- *the man who owns a donkey*,
  DefMod(Man,$(x)(\Sigma y : \text{donkey})\text{own}(x, y), p(z), q(z)$)
- *the man who owns the donkey*,
  DefMod(Man,$(x)\text{own}(x, p(q(z))), p(z), q(q(z))$)
- *the man who owns the donkey that the man owns*
- ...

Choice principles

From **game-theoretical semantics** (Hintikka and Kulas, *Anaphora and Definite Descriptions*, Reidel, 1985)

Existence: the object (the **referent**) must be given in the context.

Uniqueness: no more than one possible referent may be given.

Optimally: use the least specific expression that guarantees uniqueness.

Optimal choice of referent

if a man owns a horse and a donkey, he beats

- **it*
- *the donkey*
- ?*the donkey that he owns*

if a man owns a young and an old donkey, he beats

- **it*
- **the donkey*
- *the young donkey*

Anaphora resolution is an instance of **proof search**.

Presuppositions

John stops smoking presupposes that John smokes.

$$\text{stop}(A, B, x, y) : \text{prop}\ (A : \text{set}, B : A \to \text{prop}, x : A, y : B(x))$$

Generally: *A* is a proposition if *B* is true.

More presuppositions

John's wife presupposes that John is married.

$$\text{wife}(x, y) : \text{prop}\ (x : \text{man}, y : \text{married}(x))$$

Cf. well-formed division:

$$x/_b y : N\ (x, y : N, b :\sim I(N, y, 0))$$

In both cases: a "hidden" proof object.

Notice: *John's wife* vs. *the woman John is married to*: the same proof objects.

Presupposition under Sigma and Pi

if John smokes, he will stop smoking - no presupposition!

Karttunen, "Presuppositions of Compound Sentences", *Linguistic Inquiry*, 1973:

If Harry is married, then his wife is no longer living with him.

Fred has managed to kiss Cecilia and Fred will kiss Cecilia again.

Karttunen's analysis

(13) Let S stand for any sentence of the form "If A then B".
(a) If A presupposes C (A $\gg$ C), then S presupposes C (S $\gg$ C).
(b) If B presupposes C (B $\gg$ C), then S presupposes C (S $\gg$ C) unless A semantically entails C (A $\Vdash$ C).[10]

(17) Let S stand for any sentence of the form "A and B".
a. If A $\gg$ C, then S $\gg$ C.
b. If B $\gg$ C, then S $\gg$ C unless A $\Vdash$ C.

Karttunen's analysis

(13) Let S stand for any sentence of the form "If A then B".
(a) If A presupposes C (A $\gg$ C), then S presupposes C (S $\gg$ C).
(b) If B presupposes C (B $\gg$ C), then S presupposes C (S $\gg$ C) unless A semantically entails C (A $\Vdash$ C).[10]

(17) Let S stand for any sentence of the form "A and B".
a. If A $\gg$ C, then S $\gg$ C.
b. If B $\gg$ C, then S $\gg$ C unless A $\Vdash$ C.

These are special cases of Π and Σ,

$$\frac{A : \text{prop} \quad B : \text{prop}\ (A\ \text{true})}{A \supset B : \text{prop}} \qquad \frac{A : \text{prop} \quad B : \text{prop}\ (A\ \text{true})}{A \& B : \text{prop}}$$

Events and adverbs

Davidson, *Essays on Actions and Events*, 1980.

> *Jones buttered the toast in the bathroom with a knife at midnight*
>
> $(\Sigma x : \textit{buttered})(\text{bathroom}(x)\&\text{knife}(x)\&\text{midnight}(x))$

Paraphrase by anaphora:

> *Jones buttered the toast. It was in the bathroom. It was with a knife. It was at midnight.*

Rationale: an **event** is a proof of an event proposition.

Existential vs. universal

**if every man owns a donkey he beats it*

$(\Pi z : (\Pi x : \text{man})(\Sigma y : \text{donkey})\text{own}(x, y))\text{beat}(?, ?)$

Suggested principle: **universal phrases don't introduce discourse referents**

- Discourse Representation Theory (Kamp 1981)
- Dynamic Predicate Logic (Groenendijk & Stokhof 1989)

But surely there is a proof object z...

Proofs of universals

Karttunen (cited by Hintikka in 1979)

If you give every child a present for Christmas, some child will open it the same day.

$(\Pi f : (\Pi x : \mathsf{child})(\Sigma y : \mathsf{present})\mathsf{give}(x, y))(\Sigma u : \mathsf{child})\mathsf{open}(u, p(\mathsf{Ap}(f, u)))$

Proof objects and anaphora: conclusions

Π and Σ correctly predict most discourse referents and permit **compositional semantics**.

More generally, **text as context**

$$x_1 : A_1, x_2 : A_2(x_1), \ldots, x_n : A_n(x_1, \ldots, x_{n-1})$$

explains many questions of backward reference.

Related ideas:

- Hintikka's Game-Theoretical Semantics
- Karttunen's presuppositions of compound sentences
- Davidson's ontology of events

Open problems

When can we refer to a constant?

- *he beats it* can also refer to John, Bill,...

Odd examples

- *Every player chooses a pawn. He puts it on square one.*
- *if you have a dime put it in the parking meter*

Semantics and syntax

Compositionality

Definition: an operation * is **compositional** if for every constructor t, there is an operator t^* such that

$$(t(x_1, \ldots, x_n))^* = t^*(x_1^*, \ldots, x_n^*)$$

Example: if * is semantic interpretation of terms, each subterm has a semantic interpretation of its own.

The Montague architecture

Analysis trees: abstract syntax trees built by typed constructors

$$F_4 : \mathsf{NP} \to \mathsf{VP} \to \mathsf{S}$$

Interpretation: translation of abstract syntax trees to logic

$$F_4(Q, F)^* = Q^*(F^*)$$

Linearization: conversion of abstract syntax trees to strings

$$F_4(Q, F)^\circ = Q^\circ +\!+ F^\circ$$

($+\!+$ is string concatenation)

Logical Frameworks (LF)

"A framework for defining logics" (Harper, Honsell, and Plotkin)

"Higher-level type theory" (Martin-Löf)

Implementing inference rules as functions

- premisses as arguments, the conclusion as value
- variable binding and discharge as abstraction

Example: conjunction introduction

$$\&I : (A, B : \mathsf{Prop}) \to A \to B \to A\&B$$

Program code version:

```
fun ConjI : (A,B : Prop) -> El A -> El B -> El (Conj A B)
```

Montague Grammar in LF: English

Abatract syntax of English:

```
cat
  S ; NP ; VP ; CN
fun
  Pred  : NP -> VP -> S
  Every : CN -> NP
  Man   : CN
  Walk  : VP
```

Example: *every man walks*

```
Pred (Every Man) Walk : S
```

Montague Grammar in LF: Logic

Abstract syntax of Logic:

```
cat
  Prop ; Ind
fun
  Univ : (Ind -> Prop) -> Prop
  Impl : Prop -> Prop -> Prop
  man  : Ind -> Prop
  walk : Ind -> Prop
```

Example: $(\forall x)(\mathsf{man}(x) \supset \mathsf{walk}(x))$

```
Univ (\x -> Impl (man x) (walk x)) : Prop
```

Montague Grammar in LF: Interpretation

Interpretation functions, defining **domains of possible denotation**:

```
fun
  iS  : S  -> Prop
  iNP : NP -> (Ind -> Prop) -> Prop
  iVP : VP -> Ind -> Prop
  iCN : CN -> Ind -> Prop
```

Interpretation rules, one for each English constructor

```
def
  iS  (Pred np vp) = iNP np (iVP vp)
  iNP (Every cn)   = \F -> Univ (\x -> Impl (iCN cn x) (F x))
  iCN Man          = man
  iVP Walk         = walk
```

Grammatical Framework

GF = LF + **concrete syntax**

Concrete syntax = rules for **linearization**

This completes the declarative implementation of Montague grammars.

http://www.grammaticalframework.org

AR, *Grammatical Framework: Programming with Multilingual Grammars*, CSLI, Stanford, 2011.

CSLI Studies in Computational Linguistics

GRAMMATICAL FRAMEWORK is a programming language designed for writing grammars, which has the capability of addressing several languages in parallel. This thorough introduction demonstrates how to write grammars in Grammatical Framework and use them in applications such as tourist phrasebooks, spoken dialogue systems, and natural language interfaces. The examples and exercises presented here address several languages, and the readers are shown how to look at their own languages from the computational perspective.

Since the book requires no previous knowledge of linguistics, it can be an effective and useful resource for computer scientists and programmers, while introducing linguists to a novel approach to multilingual grammars inspired by the theory of programming languages.

Aarne Ranta is professor of computer science at the University of Gothenburg, Sweden. He is the acting coordinator of the European Union research project MOLTO (Multilingual On-Line Translation), which develops techniques for high-quality translation among fifteen languages.

CSLI PUBLICATIONS
Center for the Study of Language and Information
Stanford, California

ISBN-13 978-1-57586-626-0
ISBN-10 1-57586-626-9

Aarne Ranta

Grammatical Framework
Programming with Multilingual Grammars

CSLI

CSLI Studies in Computational Linguistics

Grammatical Framework

Programming with Multilingual Grammars

Aarne Ranta

Montague Grammar in LF: English linearization

Linearization is compositional:

```
lin
  Pred  np vp = np ++ vp
  Every cn    = "every" ++ cn
  Man         = "man"
  Walk        = "walks"
```

Linearization types

"Domains of denotation" for linearization.

Baseline: just strings:

```
lincat S, NP, VP, CN = Str
```

But this doesn't scale up well. For instance, **agreement** is not handled in

every man walks

all men walk

Parameters, records and tables

Richer linearization types

```
param Number = Sg | Pl
lincat
  S       = Str
  NP      = {s : Str ; n : Number}
  VP, CN = Number => Str
```

NP has **inherent** number, VP and CN have **variable** number.

Linearization with agreement

Still compositional, with the same abstract syntax:

```
lin
  Pred  np vp = np.s ++ vp ! np.n
  Every cn    = {s = "every" ++ cn ! Sg ; n = Sg}
  Man         = table {Sg => "man"   ; Pl => "men"}
  Walk        = table {Sg => "walks" ; Pl => "walk"}
```

$t!v$ **select** from table

$r.k$ **project** from record

The framework idea

LF: a framework for defining logics

- classical, modal, intuitionistic
- arithmetic, set theory, ontology

GF: a framework for defining grammars

- Montague grammar, type-theoretical grammar
- English, French; logical symbolisms, programming languages

Grammar = abstract syntax + concrete syntax

Type theory in GF

```
cat
  Set ; El (A : Set)
fun
  Sigma, Pi : (A : Set) -> (El A -> Set) -> Set
  Pair : (A,B : Set) -> (a : El A) -> (b : El B) -> El (Sigma A B)
  p    : (A,B : Set) -> (c : El (Sigma A B)) -> El A
  q    : (A,B : Set) -> (c : El (Sigma A B)) -> El (B (p A B c))
```

Type-theoretical semantics: one domain

```
fun
  D   : Set
  iS  : S  -> Set
  iNP : NP -> (El D -> Set) -> Set
  iVP : VP -> El D -> Set
  iCN : CN -> El D -> Set
def
  iNP (Every cn) = \F -> Pi D (\x -> Pi (iCN cn x) (\_ -> F x))
```

Abstract syntax with multiple domains

```
cat
  S
  CN
  NP (A : Set)
  VP (A : Set)
fun
  Pred  : (A : Set) -> NP A -> VP A -> S
  Every : (A : CN)  -> NP (iCN A)

  Man   : CN
  Walk  : VP (iCN Man)
```

Type-theoretical semantics: multiple domains

```
fun
  iS  : S  -> Set
  iCN : CN -> Set
  iNP : (A : Set) -> NP A -> (El A -> Set) -> Set
  iVP : (A : Set) -> VP A -> El A -> Set
def
  iS  (Pred A np vp) = iNP A np (iVP A vp)
  iNP _ (Every cn)   = Pi (iCN cn)
```

Donkey sentences: abstract syntax

```
cat
  V2 (A, B : Set)
fun
  If       : (A : S) -> (El (iS A) -> S) -> S
  Compl    : (A,B : Set) -> V2 A B -> NP B -> VP A
  Indef    : (A : CN)  -> NP (iCN A)
  Pron     : (A : CN) -> El (iCN A) -> NP (iCN A)
  Donkey   : CN
  Own, Beat : V2 (iCN Man) (iCN Donkey)
```

Donkey sentences: semantics

```
fun
  iV2 : (A,B : Set) -> V2 A B -> El A -> El B -> Set
def
  iS    (If A B)          = Pi (iS A) (\x -> iS (B x))
  iVP _ (Compl A B F R)   = \x -> iNP B R (\y -> iV2 A B F x y)
  iNP _ (Indef cn)        = Sigma (iCN cn x)
  iNP _ (Pron _ x)        = \F -> F x
```

Donkey sentences: concrete syntax 1

```
param
  Number = Sg | Pl
  Case   = Nom | Acc
lincat
  S      = Str
  NP     = {s : Case => Str ; n : Number}
  VP, V2 = Number => Str
lin
  If         A  B  = "if" ++ A ++ B
  Pred  _    np vp = np.s ! Nom ++ vp ! np.n
  Compl _ _ v2 np = table {n => v2 ! n ++ np.s ! Acc}
  Every      cn    = {s = table {_ => "every" ++ cn.s ! Sg} ; n = Sg}
  Indef      cn    = {s = table {_ => "a"     ++ cn.s ! Sg} ; n = Sg}
```

Donkey sentences: concrete syntax 2

```
param  Gender = He | She | It
lincat CN     = {s : Number => Str ; g : Gender}
lin
  Man    = {s = table {Sg => "man"    ; Pl => "men"}      ; g = He}
  Donkey = {s = table {Sg => "donkey" ; Pl => "donkeys"} ; g = It}

  Pron A _ = {
    s = case A.g of {
      He  => table {Nom => "he"  ; Acc => "him"} ;
      She => table {Nom => "she" ; Acc => "her"} ;
      It  => table {_   => "it"}
      } ;
    n = Sg
    }
```

The donkey sentence

Abstract syntax

```
If (Pred _ (Indef Man) (Compl Own (Indef Donkey)))
   (\z -> Beat (Pron Man (p _ _ z)) (Pron Donkey (p _ _ (q _ _ z))))
```

Linearization

```
if a man owns a donkey he beats it
```

Interpretation

```
Pi (Sigma man (\x -> Sigma donkey (\y -> own x y)))
   (\z -> Beat (p _ _ z) (p _ _ (q _ _ z)))
```

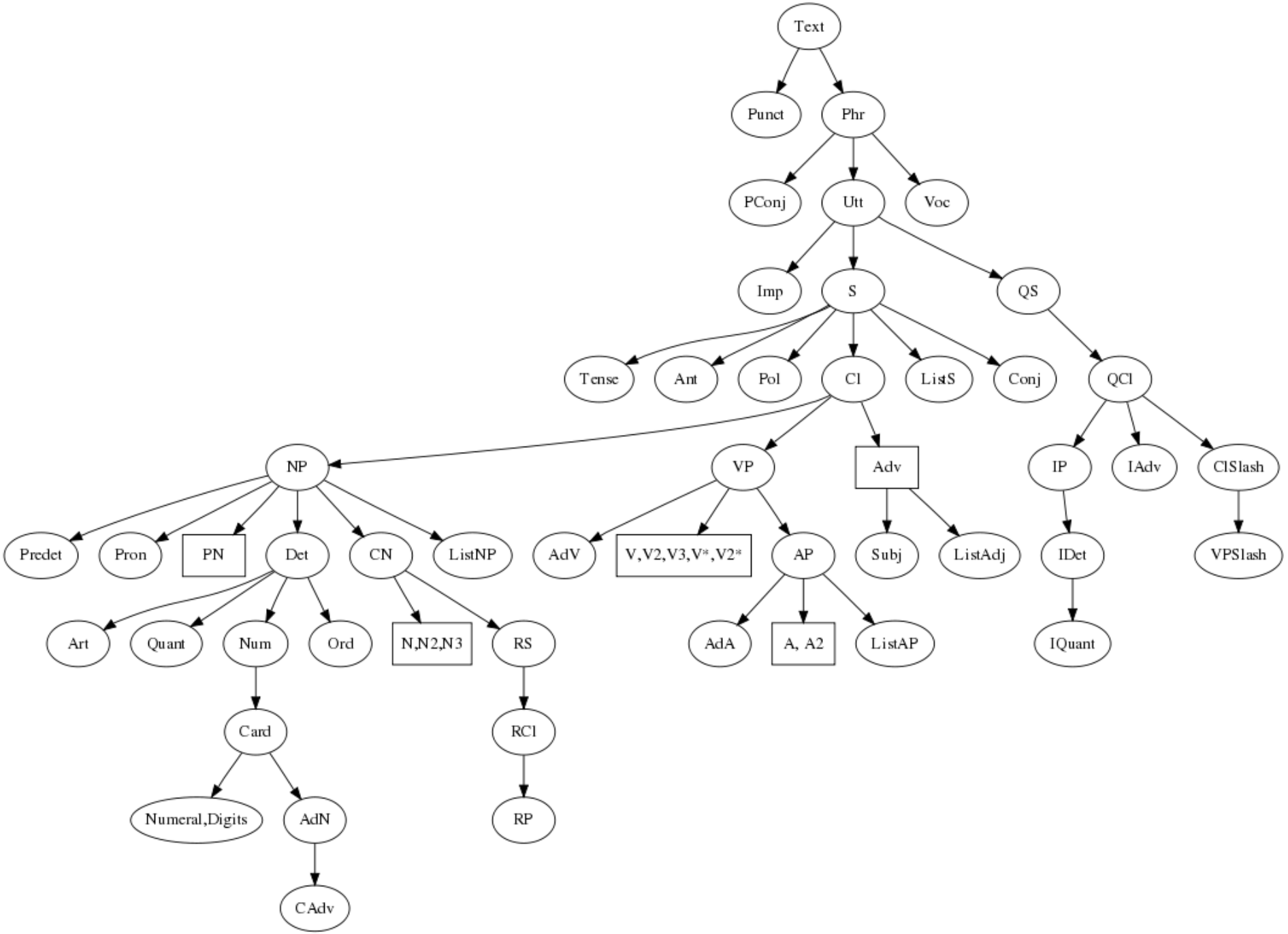
Scaling up

PTQ $\subset$ CLE $\subset$ Penn Treebank structures *

Covered by **GF Resource Grammar Library** (2001–)

Montague semantics by Björn Bringert (2008) (classical logic with event variables, automated theorem proving)

*PTQ = Proper Treatment of Quantification (Montague 1970), CLE = Core Language Engine (Alshawi & al. 1992, Penn Treebank = 1M word corpus from Wall Street Journal.


Text
Punct
Phr
PConj
Utt
Voc
Imp
S
QS
Tense
Ant
Pol
Cl
ListS
Conj
QCl
NP
VP
Adv
IP
IAdv
ClSlash
Predet
Pron
PN
Det
CN
ListNP
AdV
V,V2,V3,V*,V2*
AP
Subj
ListAdj
IDet
VPSlash
Art
Quant
Num
Ord
N,N2,N3
RS
AdA
A, A2
ListAP
IQuant
Card
RCl
Numeral,Digits
AdN
RP
CAdv

Summary of work so far

Compositional semantics of donkey sentences in type theory.

Compositional linearization of donkey sentences in GF.

Scaling up to the comprehensive GF Resource Grammar Library.

Multilinguality

Multilingual grammars

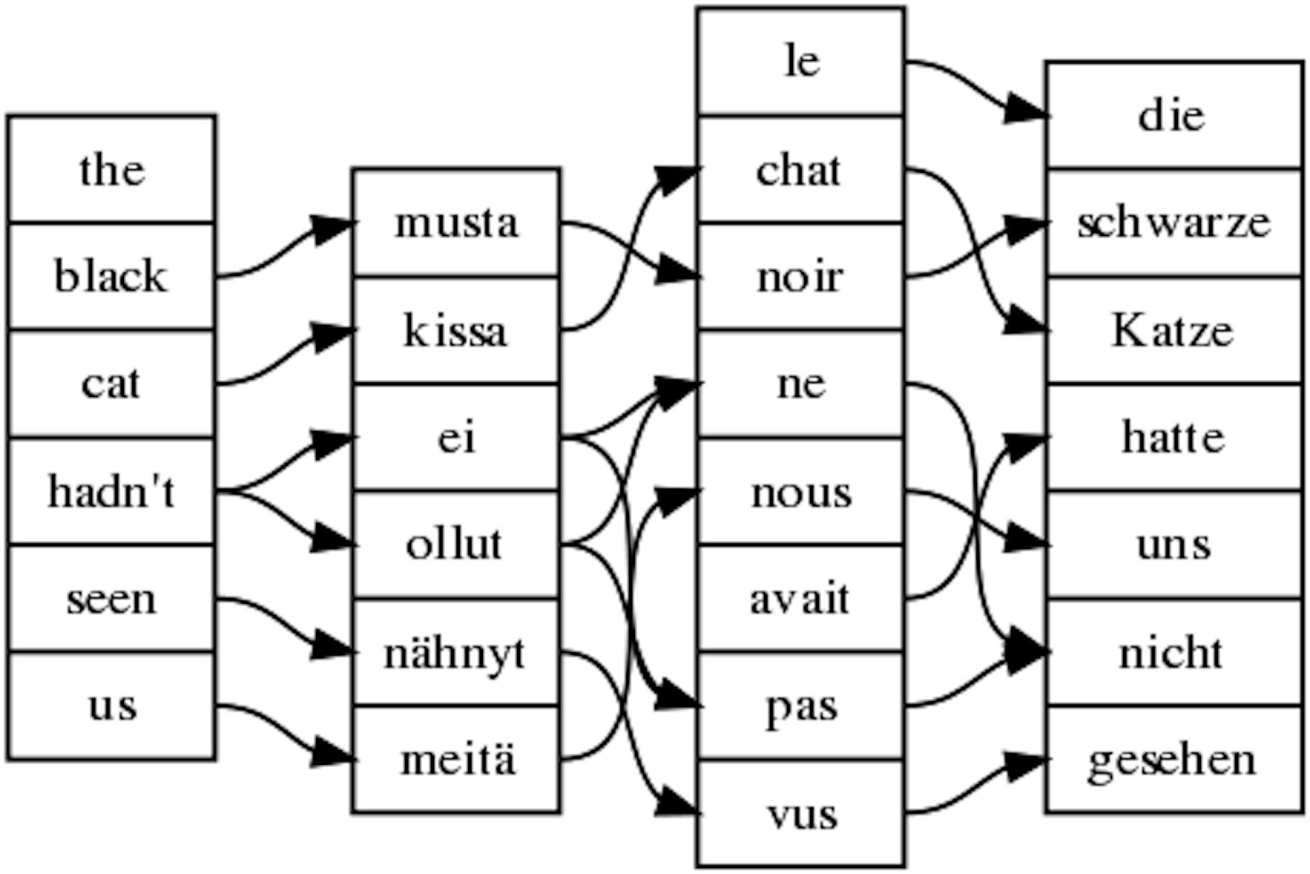
multilingual grammar = abstract syntax + many concrete syntaxes

Curry 1961: tectogrammar + phenogrammars

Possible in GF by the use of

- word order variation
- language-dependent parameters for inflection and agreement
- discontinuous constituents

Example of shared structure and word alignment




```
UseCl Anter Past Neg (Pred (Def (Mod black_A cat_N)) (Compl see_V2 we_NP))
```

Romance verb phrase

```
lincat VP = {
  s      : Verb ;
  agr    : VPAgr ;                      -- dit/dite dep. on verb, subj, and cl
  neg    : Polarity => (Str * Str) ;    -- ne-pas
  clit1  : Str ;                        -- le/se
  clit2  : Str ;                        -- lui
  clit3  : Str ;                        -- y en
  comp   : Agr => Str ;                 -- content(e) ; à ma mère ; hier
  ext    : Polarity => Str ;            -- que je dors / que je dorme
  }
```

Languages

Indo-European

- Germanic: Afrikaans, Danish, English, Dutch, German, Norwegian, Swedish
- Romance: Catalan, French, Italian, (Latin,) Romanian, Spanish
- Slavic: Bulgarian, Polish, Russian
- Baltic: Latvian
- Indo-Iranian: (Hindi,) Nepali, Persian, Punjabi, Urdu

Fenno-Ugric: (Estonian,) Finnish
Semitic: (Arabic, Hebrew, Maltese)
East-Asian: (Chinese, Japanese, Thai)
Other: (Swahili, Turkish)

mkCl	NP -> V -> Cl	*she sleeps*
mkCl	NP -> V2 -> NP -> Cl	*she loves him*
mkCl	NP -> V3 -> NP -> NP -> Cl	*she sends it to him*
mkCl	NP -> VV -> VP -> Cl	*she wants to sleep*
mkCl	NP -> VS -> S -> Cl	*she say*
mkCl	NP -> VQ -> QS -> Cl	*she wo*
mkCl	NP -> VA -> A -> Cl	*she bec*
mkCl	NP -> VA -> AP -> Cl	*she bec*
mkCl	NP -> V2A -> NP -> A -> Cl	*she pai*
mkCl	NP -> V2A -> NP -> AP -> Cl	*she pai*
mkCl	NP -> V2S -> NP -> S -> Cl	*she ans*
mkCl	NP -> V2Q -> NP -> QS -> Cl	*she ask*
mkCl	NP -> V2V -> NP -> VP -> Cl	*she beg*
mkCl	NP -> A -> Cl	*she is o*
mkCl	NP -> A -> NP -> Cl	*she is o*
mkCl	NP -> A2 -> NP -> Cl	*she is m*
mkCl	NP -> AP -> Cl	*she is very old*
mkCl	NP -> NP -> Cl	*she is the woman*

- API: `mkCl she_NP want_VV (mkVP sleep_V)`
- Bul: *тя иска да спи*
- Cat: *ella vol dormir*
- Dan: *hun vil sove*
- Dut: *ze wil slapen*
- Eng: *she wants to sleep*
- Fin: *hän tahtoo nukkua*
- Fre: *elle veut dormir*
- Ger: *sie will schlafen*
- Ita: *lei vuole dormire*
- Nor: *hun vil sove*
- Pol: *ona chce spać*
- Ron: *ea vrea să doarmă*
- Rus: *она хочет спать*
- Spa: *ella quiere dormir*
- Swe: *hon vill sova*
- Urd: وہ سونا چاہتی ہے

Demo

Parsing and generating with the Resource Grammar

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

Select Grammar -> MiniGrammar.pgf

Translating pronouns

(Hutchins & Somers 1992)

the monkey ate the banana because it was hungry	*er* (der Affe)
the monkey ate the banana because it was ripe	*sie* (die Banane)
the monkey ate the banana because it was tea-time	*es*

So we need to find the referent of *it*, or at least its type.

What is semantics

Translation equivalence

Convey the proper meaning (on some level of abstraction)

Literal meaning

Use the same syntactic structure (Resource Grammar abstract syntax)

Preservation of information

Literal translation:

if a man owns a donkey he beats it

si un homme possède un âne il le bat

But this also means

if a man owns a donkey it beats him

To remove ambiguity, we should choose

si un homme possède un âne il bat l'âne

Idiomaticity

Eng. *what is your name*

Ger. *was ist dein Name -> wie heißt du*

Fre. *quel est ton nom -> comment t'appelles-tu*

Language games

English	**Swedish**	**German**
A beer please.	*En öl tack.*	*Ein Bier bitte.*
Here we are.	*Var så god.*	*Bitte.*
Thank you.	*Tack.*	*Danke.*
You're welcome.	*Var så god.*	*Bitte.*

Compositional semantics

We *can* give a compositional semantics to

s'il vous plaît

via its abstract syntax tree

```
SubjS si_Subj (UseCl Pres Simul Pos (Pred il_NP (Compl plaire_V2 vous_NP)))
```

But this is usually *not* what we want!

Demo

The MOLTO phrasebook:

- idiomatic travel phrases
- 15 languages
- disambiguation via grammar-generated feedback

http://www.grammaticalframework.org/demos/phrasebook/

Also for Android phones: Phrasedroid,

http://www.androidzoom.com/android_applications/travel_and_local/phrasedr