From Logical to Grammatical Framework

Aarne Ranta
Department of Computer Science and Engineering
University of Gothenburg and Chalmers University of Technology
and
Digital Grammars AB

HCM Workshop: Mathematical Language and Practical Type Theory

Bonn 1-4 February 2020

digitalGrammars
Language technology to rely on.

1.
GF: formalize the relation between formal and informal language

LF

```
cat C Γ

fun f : T
```

LF example

```
cat Prop

cat Proof (A : Prop)

fun Conj : Prop -> Prop -> Prop

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

LF

```
cat C Γ

fun f : T
```

concrete syntax

```
lincat C = L

lin f = l
```

GF = LF + concrete syntax

abstract

```
cat Prop

cat Proof (A : Prop)

fun Conj : Prop -> Prop -> Prop

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

concrete

```
lincat Prop = Str

lincat Proof = Str

lin Conj A B = A ++ “\\&” ++ B

lin ConjI A B a b =
  “\\infer[ConjI]{“ ++
    A ++ “\\&” ++ B ++ “}{“ ++
    “{“ ++ a ++ b ++ “}”
```

Types for LF

Basic type formation.

$$\frac{\textbf{cat}\ C\ (x_1 : A_1)\cdots(x_n : A_n) \quad a_1 : A_1 \quad \ldots \quad a_n : A_n(x_1 := a_1, \ldots, x_{n-1} := a_{n-1})}{C\, a_1 \ldots a_n : \mathit{Type}}$$

Basic object formation.

$$\frac{\textbf{fun}\ f : A}{f : A}$$

Function type formation, application, and abstraction.

$$\frac{A : \mathit{Type} \quad \begin{matrix}(x : A)\\ B : \mathit{Type}\end{matrix}}{(x : A) \to B : \mathit{Type}} \qquad \frac{f : (x : A) \to B \quad a : A}{f\, a : B(x := a)} \qquad \frac{\begin{matrix}(x : A)\\ b : B\end{matrix}}{\lambda x \to b : (x : A) \to B}$$

AR, “Grammatical Framework: A Type-Theoretical Grammar Formalism”, *Journal of Functional Programming*, 2004

Types for concrete syntax, 1

Str

Types for concrete syntax, 2

Str

Str + precedence

abstract

```
cat Prop

fun Conj : Prop -> Prop -> Prop
```

concrete

```
lincat Prop = PrecExp

lin Conj A B = infixl p3 “\\&” A B

oper infixl :
  Prec -> Str ->
    PrecExp -> PrecExp -> PrecExp
  = ...
```

Types for concrete syntax, 3

Str

Str + precedence = {s : Str ; p : Prec}

Str^n

- tables
- records

```
cat Set

fun Pt : Set
```

```
lincat Set = Number => Str

lin Pt = table {
  Sg => “point” ;
  Pl => “points”
  }

param Number = Sg | Pl
```

Parameter types and constructors.

$$\frac{\textbf{param}\ P\ =\ \ldots}{P\ :\ PType} \qquad \frac{P\ :\ PType}{P\ :\ Type} \qquad \frac{\textbf{param}\ P\ =\ \ldots\ |\ C\ P_1\ \ldots\ P_n\ |\ \ldots}{C\ :\ P_1\ \to\ \ldots\ \to\ P_n\ \to\ P}$$

Record type formation.

$$\frac{T_1, \ldots, T_n\ :\ Type}{\{r_1\ :\ T_1\ ;\ \ldots\ ;\ r_n\ :\ T_n\}\ :\ Type}$$

Record formation and projection.

$$\frac{t_1\ :\ T_1\quad \ldots\quad t_n\ :\ T_n}{\{r_1 = t_1\ ;\ \ldots\ ;\ r_n = t_n\}\ :\ \{r_1\ :\ T_1\ ;\ \ldots\ ;\ r_n\ :\ T_n\}} \qquad \frac{c\ :\ \{\ldots\ ;\ r\ :\ T\ ;\ \ldots\}}{c.r\ :\ T}$$

Projection computation.

$$\{\ldots ; r\ =\ t; \ldots\}.r = t$$

Table type formation.

$$\frac{P\ :\ PType\quad T\ :\ Type}{P\ \Rightarrow\ T\ :\ Type}$$

Table formation and selection.

$$\frac{t_1\ :\ T\ \Gamma_P\, p_1\quad \ldots\quad t_n\ :\ T\ \Gamma_P\, p_n\quad [p_1, \ldots, p_n \text{ exhaustive for } P]}{\textbf{table}\ \{p_1\ \Rightarrow\ t_1\ ;\ \ldots\ ;\ p_n\ \Rightarrow\ t_n\}\ :\ P\ \Rightarrow\ T} \qquad \frac{c\ :\ P\ \Rightarrow\ T\quad p\ :\ P}{c\ !\ p\ :\ T}$$

AR: GF, JFP 2004

```
AxIa1
  : (a:Pt) -> (n1':((DistPt ((id Pt) a)) ((id Pt) a))) -> data{},

AxIa2
  : (l:Ln) -> ((EqLn ((id Ln) l)) ((id Ln) l)),

AxIa3
  : (l:Ln) -> (n1':((Int ((id Ln) l)) ((id Ln) l))) -> data{},

AxIb1
  : (a:Pt) -> (b:Pt) -> (h1':((DiPt ((id Pt) a)) ((id Pt) b))) -> (c:Pt) -> data{$i1' (x1':((DistPt
((id Pt) a)) ((id Pt) c))), $i2' (x1':((DistPt ((id Pt) b)) ((id Pt) c)))},

AxIb2
  : (l:Ln) -> (m:Ln) -> (h1':((DiLn ((id Ln) l)) ((id Ln) m))) -> (n:Ln) -> data{$i1' (x1':((DiLn
((id Ln) l)) ((id Ln) n))), $i2' (x1':((DiLn ((id Ln) m)) ((id Ln) n)))},

AxIb3
  : (l:Ln) -> (m:Ln) -> (h1':((Int ((id Ln) l)) ((id Ln) m))) -> (n:Ln) -> data{$i1' (x1':((Int
((id Ln) l)) ((id Ln) n))), $i2' (x1':((Int ((id Ln) m)) ((id Ln) n)))},

AxII1
  : (x:Pt) -> (y:((Mod Pt) (\ x0' -> ((DistPt x0') ((id Pt) x))))) -> sig{l1':((Inc ((id Pt) x))
((ln ((id Pt) x)) ((id ((Mod Pt) (\ x1' -> ((DistPt x1') ((id Pt) x))))) y))), l2':((Inc ((id Pt)
(((proj ?) ?) y))) ((ln ((id Pt) x)) ((id ((Mod Pt) (\ x1' -> ((DistPt x1') ((id Pt) x))))) y)))},

AxII2
  : (x:Ln) -> (y:((Mod Ln) (\ x0' -> ((Int ((id Ln) x)) x0')))) -> sig{l1':((Cont ((id Ln) x)) ((pt
((id Ln) x)) ((id ((Mod Ln) (\ x1' -> ((Int ((id Ln) x)) x1')))) y))), l2':((Cont ((id Ln) (((proj
?) ?) y))) ((pt ((id Ln) x)) ((id ((Mod Ln) (\ x1' -> ((Int ((id Ln) x)) x1')))) y)))},

AxIII
  : (a:Pt) -> (b:Pt) -> (l:Ln) -> (m:Ln) -> (h1':sig{l1':((DiPt ((id Pt) a)) ((id Pt) b)),
l2':((DistLn ((id Ln) l)) ((id Ln) m))}) -> data{$i1' (x1':data{$i1' (x2':((Apt ((id Pt) a)) ((id
Ln) l))), $i2' (x2':((Apt ((id Pt) a)) ((id Ln) m)))}), $i2' (x1':data{$i1' (x2':((Apt ((id Pt) b))
((id Ln) l))), $i2' (x2':((Apt ((id Pt) b)) ((id Ln) m)))})},

AxIV1
  : (a:Pt) -> (l:Ln) -> (h1':((Apt ((id Pt) a)) ((id Ln) l))) -> (b:Pt) -> data{$i1' (x1':((DiPt
((id Pt) a)) ((id Pt) b))), $i2' (x1':((Apt ((id Pt) b)) ((id Ln) l)))},

AxIV2
  : (a:Pt) -> (l:Ln) -> (h1':((Apt ((id Pt) a)) ((id Ln) l))) -> (m:Ln) -> data{$i1' (x1':((DiLn
((id Ln) l)) ((id Ln) m))), $i2' (x1':((Apt ((id Pt) a)) ((id Ln) m)))},

AxIV3
  : (l:Ln) -> (m:Ln) -> (h1':((Int ((id Ln) l)) ((id Ln) m))) -> (n:Ln) -> data{$i1' (x1':((DiLn
((id Ln) m)) ((id Ln) n))), $i2' (x1':((Int ((id Ln) l)) ((id Ln) n)))}
```

AxIa1. There exists no point a such that a is distinct from a.

AxIa2. For all lines l, l and l are coincident.

AxIa3. Whatever the line l, l does not intersect l.

AxIb2. For all lines l and m, if l and m are distinct and if n is a line, then l and n are distinct or m and n are distinct.

AxIb3. If l and m are lines and if l intersects m, then whatever the line n, either l or m intersects n.

AxII1. For all points x, if y is a point distinct from x, then both x and y are incident with the connecting line of x and y.

AxII2. If x is a line, then whatever the line y which x intersects, both x and y contain the intersection point of x and y.

AxIII. For all points a and b, for all lines l and m, if a and b are distinct points and l is distinct from m, then a or b is apart from l or m.

AxIV1. If a is a point, if l is a line and if a is apart from l, then whatever the point b, a and b are distinct or b is apart from l.

AxIV2. If a is a point, if l is a line and if a is apart from l, then whatever the line m, l and m are distinct or a is apart from m.

AxIV3. If l and m are lines and if l intersects m, then whatever the line n, m and n are distinct lines or l intersects n.

1996: ALF formalization of J. von Plato, The Axioms of Constructive Geometry, *Annals of Pure and Applied Logic* 76, 1995.

2.
Über eine bisher noch nicht benützte Anwendung des formalen Standpunktes

2.
Über eine bisher noch nicht benützte Anwendung des formalen Standpunktes:

- vary the concrete syntax to target several languages

LF abstracts away from

- shape of words
- order of words
- number of words
- morphological variation
- discontinuous constituents

```
cat Set


fun Pt : Set
```

```
lincat Set = {
  s : Number => Case => Str ;
  g : Gender
  }
lin Pt = {
  s = table {
    Sg => table {
      Nom | Acc => “Punkt” ;
      Dat => “Punkt” | “Punkte” ;
      Gen => “Punktes” | “Punkts”
      } ;
    Pl => table {
      Dat => “Punkten” ;
      _ => “Punkte”
      }
    } ;
    g = Masc
  }

param Number = Sg | Pl
param Gender = Masc | Fem | Neutr
param Case = Nom | Acc | Gen | Dat
```

	Singular	Plural
Nominativ	der Punkt	die Punkte
Genitiv	des Punktes des Punkts	der Punkte
Dativ	dem Punkt dem Punkte	den Punkten
Akkusativ	den Punkt	die Punkte

$$A \to A$$

Snow is white.

Schnee ist weiß.

If snow is white, snow is white.

Wenn Schnee weiß ist, ist Schnee weiß.

```
cat Prop

fun Inc :
  Elem Pt -> Elem Ln -> Prop
```

```
lincat Prop = Order => Str

lin Inc x y =
  let
    subj = x.s ! Nom ;
    verb = sein x.n ;
    compl = “inzident” ++ “mit” ++ y.s ! Dat
  in table {
    Main => subj ++ verb ++ compl ;
    Inv  => verb ++ subj ++ compl ;
    Sub  => subj ++ compl ++ verb
    }

param Order = Main | Inv | Sub
```

AxIa1. Es gibt keinen Punkt a derart, daß a verschieden von a ist.

AxIa2. Für alle Geraden l gilt es, daß l und l gleich sind.

AxIa3. Welche auch die Gerade l sein mag, schneidet l nicht l.

AxIb1. Für alle Punkte a und b gilt es, daß wenn a und b verschieden sind, so ist entweder a oder b verschieden von c, welcher auch der Punkt c sein mag.

AxIb2. Für alle Geraden l und m gilt es, daß wenn l und m verschieden sind und wenn n eine Gerade ist, so sind l und n verschieden oder m und n sind verschieden.

AxIb3. Wenn l und m Geraden sind und wenn l m schneidet, so schneidet entweder l oder m n, welche auch die Gerade n sein mag.

AxII1. Für alle Punkte x gilt es, daß wenn y ein von x verschiedener Punkt ist, so sind sowohl x als auch y inzident mit der Verbindungsgeraden zwischen x und y.

AxII2. Wenn x eine Gerade ist, so enthalten sowohl x als auch y den Schnittpunkt von x und y, welche auch die Gerade y die x schneidet sein mag.

AxIII. Für alle Punkte a und b gilt es, daß es für alle Geraden l und m gilt, daß wenn a und b verschiedene Punkte sind und l verschieden von m ist, so ist a oder b außerhalb l oder m.

AxIV1. Wenn a ein Punkt ist, wenn l eine Gerade ist und wenn a außerhalb l ist, so sind a und b verschieden oder b ist außerhalb l, welcher auch der Punkt b sein mag.

AxIV2. Wenn a ein Punkt ist, wenn l eine Gerade ist und wenn a außerhalb l ist, so sind l und m verschieden oder a ist außerhalb m, welche auch die Gerade m sein mag.

AxIV3. Wenn l und m Geraden sind und wenn l m schneidet, so sind m und n verschiedene Geraden oder l schneidet n, welche auch die Gerade n

AxIa1. Es gibt keinen Punkt a derart, daß a verschieden von a ist.

AxIa2. Für alle Geraden l gilt es, daß l und l gleich sind.

AxIa3. Welche auch die Gerade l sein mag, schneidet l nicht l.

AxIb1. Für alle Punkte a und b gilt es, daß wenn a und b verschieden sind, so ist entweder a oder b verschieden von c, welcher auch der Punkt c sein mag.

AxIb2. Für alle Geraden l und m gilt es, daß wenn l und m verschieden sind und wenn n eine Gerade ist, so sind l und n verschieden oder m und n sind verschieden.

AxIb3. Wenn l und m Geraden sind und wenn l m schneidet, so schneidet entweder l oder m n, welche auch die Gerade n sein mag.

AxII1. Für alle Punkte x gilt es, daß wenn y ein von x verschiedener Punkt ist, so sind sowohl x als auch y inzident mit der Verbindungsgeraden zwischen x und y.

AxII2. Wenn x eine Gerade ist, so enthalten sowohl x als auch y den Schnittpunkt von x und y, welche auch die Gerade y die x schneidet sein mag.

AxIII. Für alle Punkte a und b gilt es, daß es für alle Geraden l und m gilt, daß wenn a und b verschiedene Punkte sind und l verschieden von m ist, so ist a oder b außerhalb l oder m.

AxIV1. Wenn a ein Punkt ist, wenn l eine Gerade ist und wenn a außerhalb l ist, so sind a und b verschieden oder b ist außerhalb l, welcher auch der Punkt b sein mag.

AxIV2. Wenn a ein Punkt ist, wenn l eine Gerade ist und wenn a außerhalb l ist, so sind l und m verschieden oder a ist außerhalb m, welche auch die Gerade m sein mag.

AxIV3. Wenn l und m Geraden sind und wenn l m schneidet, so sind m und n verschiedene Geraden oder l schneidet n, welche auch die Gerade n

AxIa1. Il n'existe aucun point a tel que a soit distinct de a.

AxIa2. Pour toute droite l, l et l sont coïncidentes.

AxIa3. Quelle que soit la droite l, l ne coupe pas l.

AxIb1. Pour tous les points a et b, si a et b sont distincts, alors quel que soit le point c, ou a ou b est distinct de c.

AxIb2. Pour toutes les droites l et m, si l et m sont distinctes et si n est une droite, alors l et n sont distinctes ou m et n sont distinctes.

AxIb3. Si l et m sont des droites et si l coupe m, alors quelle que soit la droite n, ou l ou m coupe n.

AxII1. Pour tout point x, si y est un point distinct de x, alors et x et y sont incidents avec la ligne de connection entre x et y.

AxII2. Si x est une droite, alors quelle que soit la droite y que x coupe, et x et y contiennent le point d'intersection de x et de y.

AxIII. Pour tous les points a et b, pour toutes les droites l et m, si a et b sont des points distincts et l est distincte de m, alors a ou b est extérieur à l ou à m.

AxIV1. Si a est un point, si l est une droite et si a est extérieur à l, alors quel que soit le point b, a et b sont distincts ou b est extérieur à l.

AxIV2. Si a est un point, si l est une droite et si a est extérieur à l, alors quelle que soit la droite m, l et m sont distinctes ou a est extérieur à m.

AxIV3. Si l et m sont des droites et si l coupe m, alors quelle que soit la droite n, m et n sont des droites distinctes ou l coupe n.

AxIa1. Non esiste nessun punto a tale che a sia distinto di a.

AxIa2. Per ogni linea l, l e l sono coincidenti.

AxIa3. Per una linea l qualsiasi, l non taglia l.

AxIb1. Per tutti i punti a e b, se a e b sono distinti, allora per un punto c qualsiasi, o a o b è distinto di c.

AxIb2. Per tutte le linee l e m, se l e m sono distinte e se n è una linea, allora l e n sono distinte o m e n sono distinte.

AxIb3. Se l e m sono delle linee e se l taglia m, allora per una linea n qualsiasi, o l o m taglia n.

AxII1. Per ogni punto x, se y è un punto distinto di x, allora e x e y sono incidenti colla linea di connessione di x e y.

AxII2. Se x è una linea, allora per una linea y che x taglia qualsiasi, e x e y contengono il punto d'intersezione di x e y.

AxIII. Per tutti i punti a e b, per tutte le linee l e m, se a e b sono dei punti distinti e l è distinta di m, allora a o b è esterno a l o a m.

AxIV1. Se a è un punto, se l è una linea e se a è esterno a l, allora per un punto b qualsiasi, a e b sono distinti o b è esterno a l.

AxIV2. Se a è un punto, se l è una linea e se a è esterno a l, allora per una linea m qualsiasi, l e m sono distinte o a è esterno a m.

AxIV3. Se l e m sono delle linee e se l taglia m, allora per una linea n qualsiasi, m e n sono delle linee distinte o l taglia n.

AxIa1. Non esiste nessun punto a tale che a sia distinto di a.

AxIa2. Per ogni linea l, l e l sono coincidenti.

AxIa3. Per una linea l qualsiasi, l non taglia l.

AxIb1. Per tutti i punti a e b, se a e b sono distinti, allora per un punto c qualsiasi, o a o b è distinto di c.

AxIb2. Per tutte le linee l e m, se l e m sono distinte e se n è una linea, allora l e n sono distinte o m e n sono distinte.

AxIb3. Se l e m sono delle linee e se l taglia m, allora per una linea n qualsiasi, o l o m taglia n.

AxII1. Per ogni punto x, se y è un punto distinto di x, allora e x e y sono incidenti colla linea di connessione di x e y.

AxII2. Se x è una linea, allora per una linea y che x taglia qualsiasi, e x e y contengono il punto d'intersezione di x e y.

AxIII. Per tutti i punti a e b, per tutte le linee l e m, se a e b sono dei punti distinti e l è distinta di m, allora a o b è esterno a l o a m.

AxIV1. Se a è un punto, se l è una linea e se a è esterno a l, allora per un punto b qualsiasi, a e b sono distinti o b è esterno a l.

AxIV2. Se a è un punto, se l è una linea e se a è esterno a l, allora per una linea m qualsiasi, l e m sono distinte o a è esterno a m.

AxIV3. Se l e m sono delle linee e se l taglia m, allora per una linea n qualsiasi, m e n sono delle linee distinte o l taglia n.

AxIa1. Millekään pisteelle a ei päde, että a olisi eri kuin a.

AxIa2. Kaikille suorille l pätee, että l ja l ovat samoja.

AxIa3. Oli suora l mikä tahansa, l ei leikkaa l:ä.

AxIb1. Kaikille pisteille a ja b pätee, että jos a ja b ovat eri, niin oli piste c mikä tahansa, joko a tai b on eri kuin c.

AxIb2. Kaikille suorille l ja m pätee, että jos l ja m ovat eri ja jos n on suora, niin l ja n ovat eri tai m ja n ovat eri.

AxIb3. Jos l ja m ovat suoria ja jos l leikkaa m:n, niin oli suora n mikä tahansa, joko l tai m leikkaa n:n.

AxII1. Kaikille pisteille x pätee, että jos y on piste eri kuin x, niin sekä x että y ovat insidenttejä x:n ja y:n yhdyssuoran kanssa.

AxII2. Jos x on suora, niin oli suora y jonka x leikkaa mikä tahansa, sekä x että y sisältävät x:n ja y:n leikkauspisteen.

AxIII. Kaikille pisteille a ja b pätee, että kaikille suorille l ja m pätee, että jos a ja b ovat eri pisteitä ja l on eri kuin m, niin a tai b on l:n tai m:n ulkopuolinen.

AxIV1. Jos a on piste, jos l on suora ja jos a on l:n ulkopuolinen, niin oli piste b mikä tahansa, a ja b ovat eri tai b on l:n ulkopuolinen.

AxIV2. Jos a on piste, jos l on suora ja jos a on l:n ulkopuolinen, niin oli suora m mikä tahansa, l ja m ovat eri tai a on m:n ulkopuolinen.

AxIV3. Jos l ja m ovat suoria ja jos l leikkaa m:n, niin oli suora n mikä tahansa, m ja n ovat eri suoria tai l leikkaa n:n.

3.
Hide low-level linguistic details in a Resource Grammar API

```
lincat Set = N

lincat Prop = Cl

lin Pt = mkN “point”

lin Inc x y =
 mkCl x (mkA2 (mkA “incident”) with_Prep) y
```

```
lincat Set = N

lincat Prop = Cl

lin Pt = mkN “Punkt” “Punkte” masculine

lin Inc x y =
 mkCl x (mkA2 (mkA “inzident”) mit_Prep) y
```

Smart paradigms with regular expressions

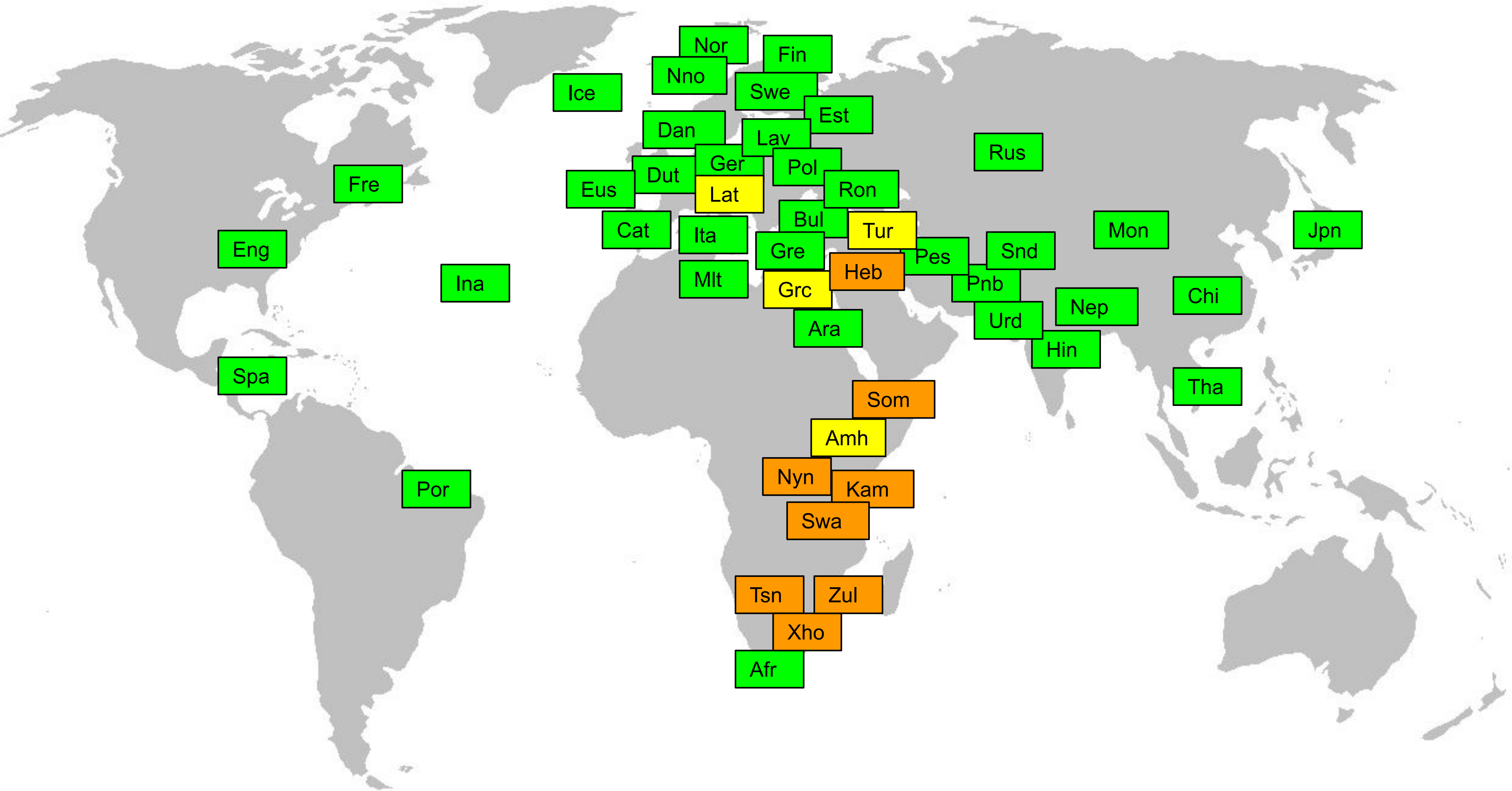
```
smartNoun : Str -> Noun = \sg -> case sg of {
   _ + ("ay"|"ey"|"oy"|"uy") => regNoun sg ;              -- boys
   x + "y"                   => mkNoun sg (x + "ies") ; -- babies
   _ + ("ch"|"sh"|"s"|"o")   => mkNoun sg (sg + "es") ; -- bushes
   _                         => regNoun sg              -- points
   }
```

RGL = Resource Grammar Library

http://www.grammaticalframework.org/lib/doc/synopsis.html

mkCl	NP -> V2 -> NP -> Cl	she loves him
mkCl	NP -> V3 -> NP -> NP -> Cl	she sen
mkCl	NP -> VV -> VP -> Cl	she war
mkCl	NP -> VS -> S -> Cl	she say
mkCl	NP -> VQ -> QS -> Cl	she wor
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 -> VPSlash -> NP -> 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 n
mkCl	NP -> AP -> Cl	she is v
mkCl	NP -> NP -> Cl	she is th
mkCl	NP -> N -> Cl	she is a
mkCl	NP -> CN -> Cl	she is a
mkCl	NP -> Adv -> Cl	she is h
mkCl	NP -> VP -> Cl	she alw
mkCl	N -> Cl	there is
mkCl	CN -> Cl	there is
mkCl	NP -> Cl	there ar
mkCl	NP -> RS -> Cl	it is she

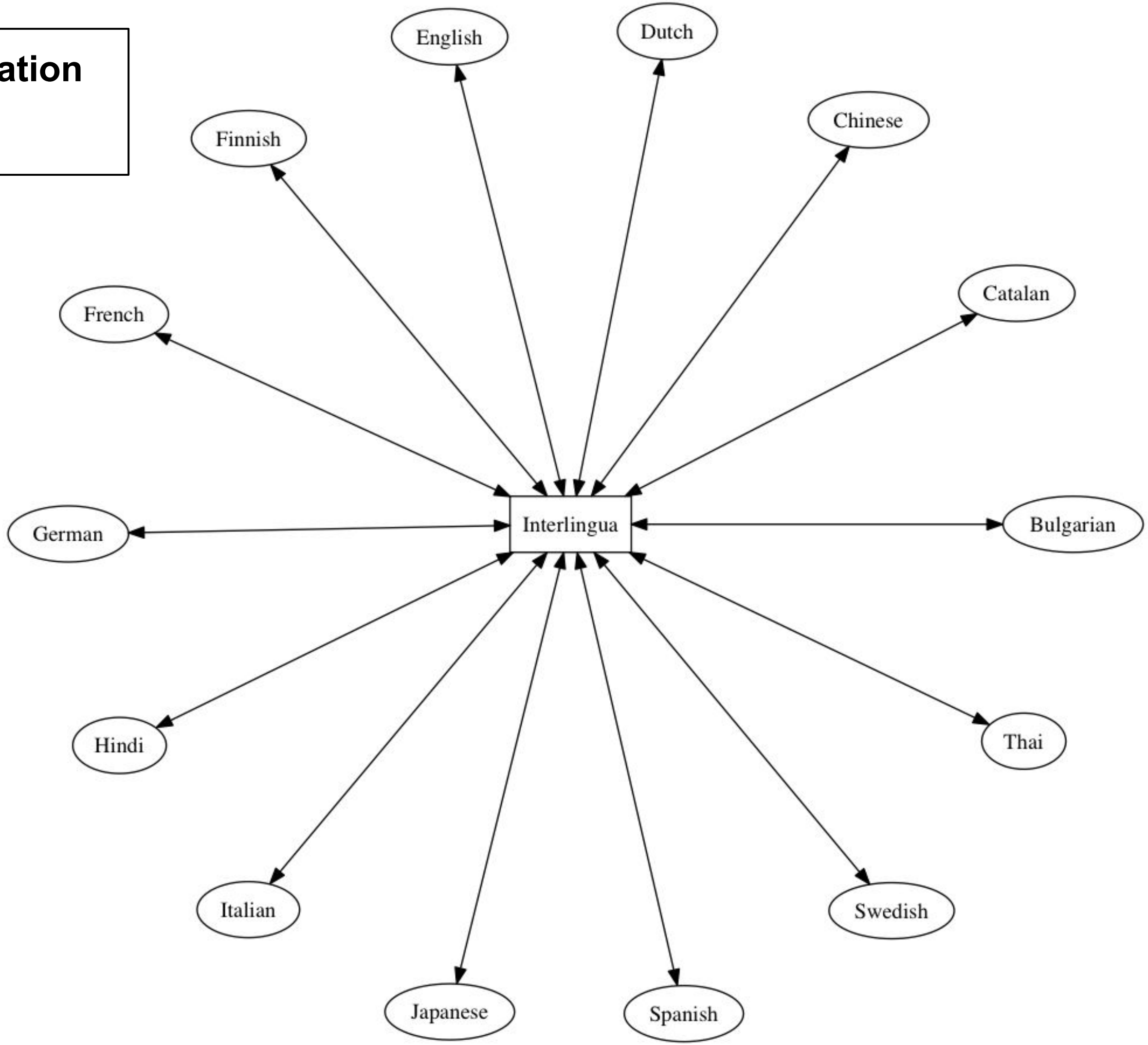
- API: `mkUtt (mkCl she_NP love_V2 he_NP)`
- Afr: *sy het hom lief*
- Ara: *تُحِبُّهُ هِيَ*
- Bul: *тя го обича*
- Cat: *ella el estima*
- Chi: *她爱他*
- Dan: *hun elsker ham*
- Dut: *zij houdt van hem*
- Eng: *she loves him*
- Est: *tema armastab teda*
- Eus: *hark hura maite du*
- Fin: *hän rakastaa häntä*
- Fre: *elle l'aime*
- Ger: *sie liebt ihn*
- Gre: *αυτή τον αγαπά*
- Hin: *वह उस को प्यार करती है*
- Ice: *hún elskar hann*
- Ita: *lei lo ama*
- Jpn: *彼女は彼を愛する*
- Lav: *viņa viņu mīl*
- Mlt: *hi tħobbu*
- Mon: *түүний түүнийг хайрладаг нь*
- Nep: *उनी उ लाई माया गर्छिन्*
- Nno: *ho elskar han*
- Nor: *hun elsker ham*
- Pes: *او او را دوست دارد*
- Pnb: *او اونوں پیار کردی اے*
- Pol: *ona kocha jego*
- Por: *ela o ama*
- Ron: *ea îl iubeşte*
- Rus: *она любит его*
- Snd: *هوء هو سان عشق ڪري ٿي*
- Spa: *ella lo ama*
- Swe: *hon älskar honom*
- Tha: *หล่อนรักเขา*
- Urd: *وہ اس کو پیار کرتی ہے*



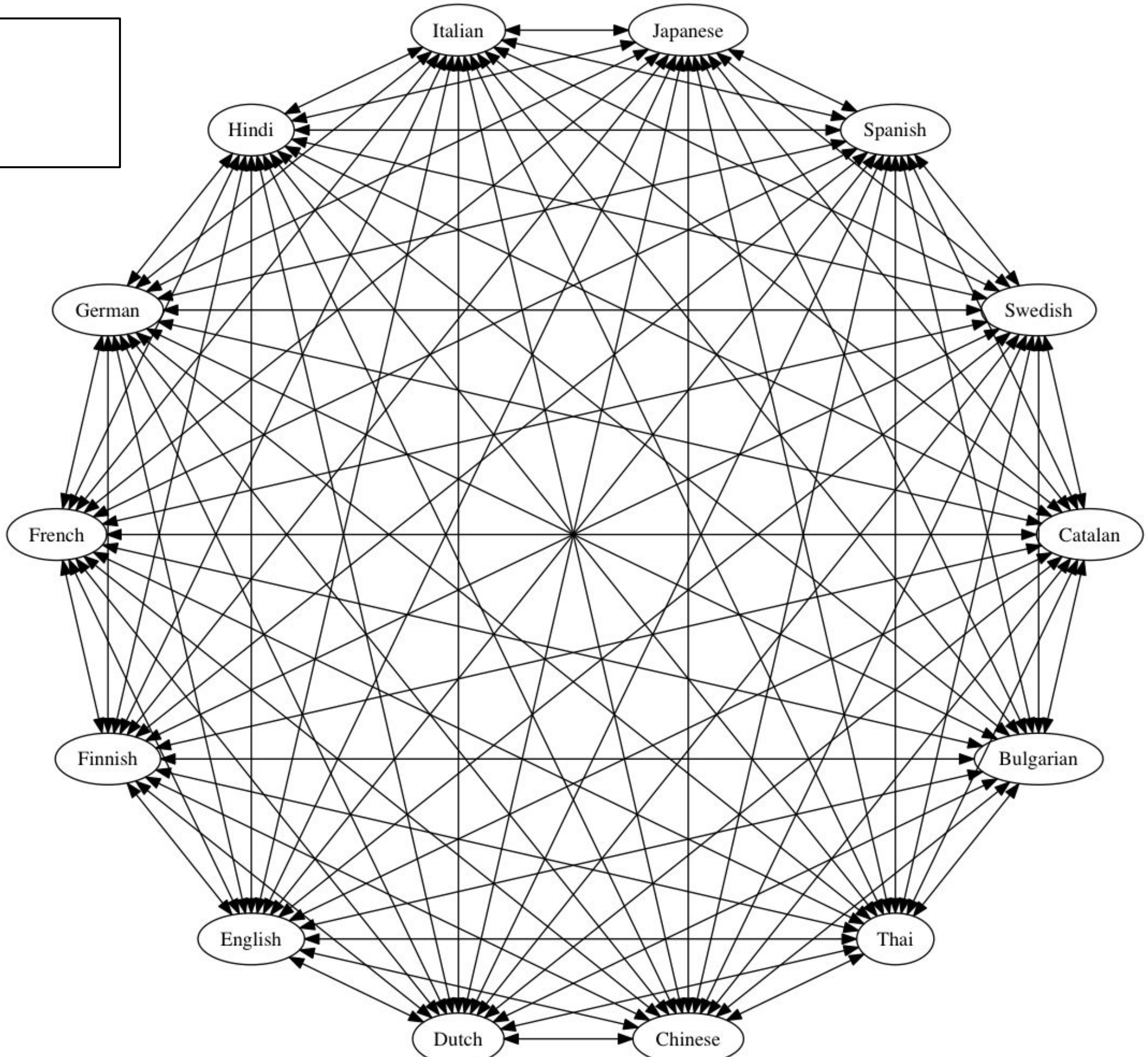
https://commons.wikimedia.org/wiki/Maps_of_the_world#/media/File:BlankMap-World-noborders.png

4.
Use LF as a general purpose framework for interlinguas

Interlingual translation



Linear scale-up



digitalGrammars
Language technology to rely on.

altran

SU Ambulans

Lex Cogitans LLC

VÄSTRA GÖTALANDSREGIONEN
SAHLGRENSKA UNIVERSITETSSJUKHUSET

DART

TD

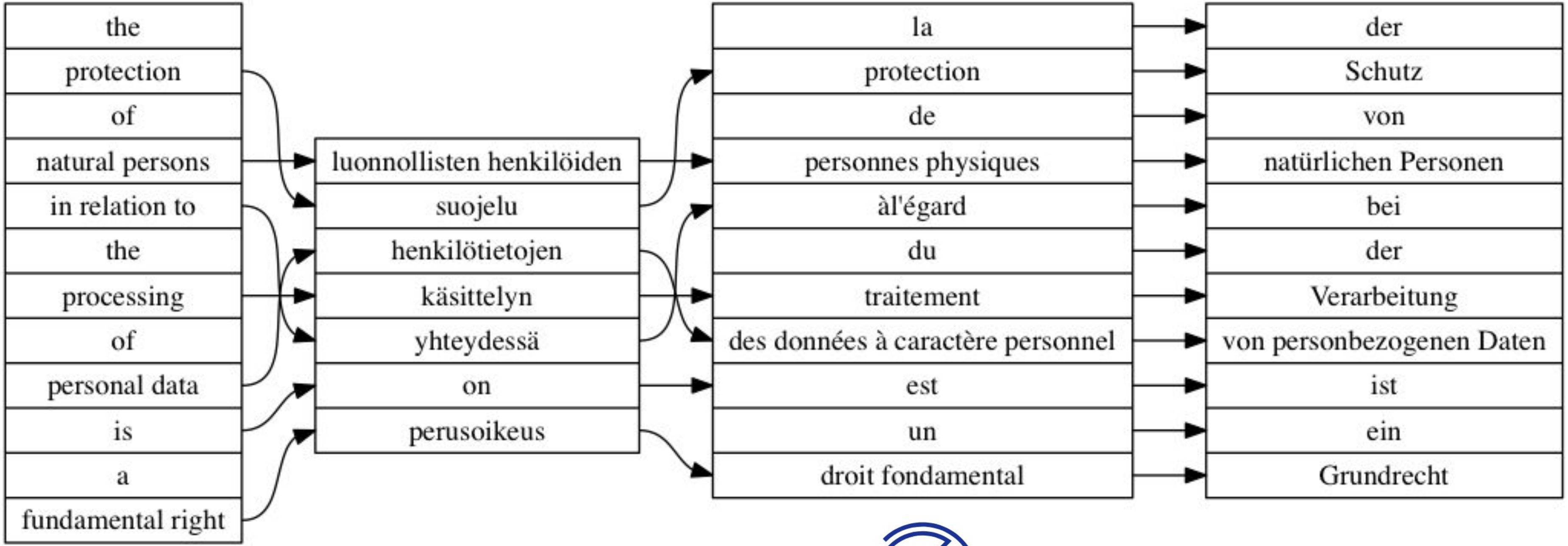
Talkamatic
NEXT GENERATION OF CONVERSATION

SIGNATU

TEXTUAL

2014 2015 2016 2017 2018 2019

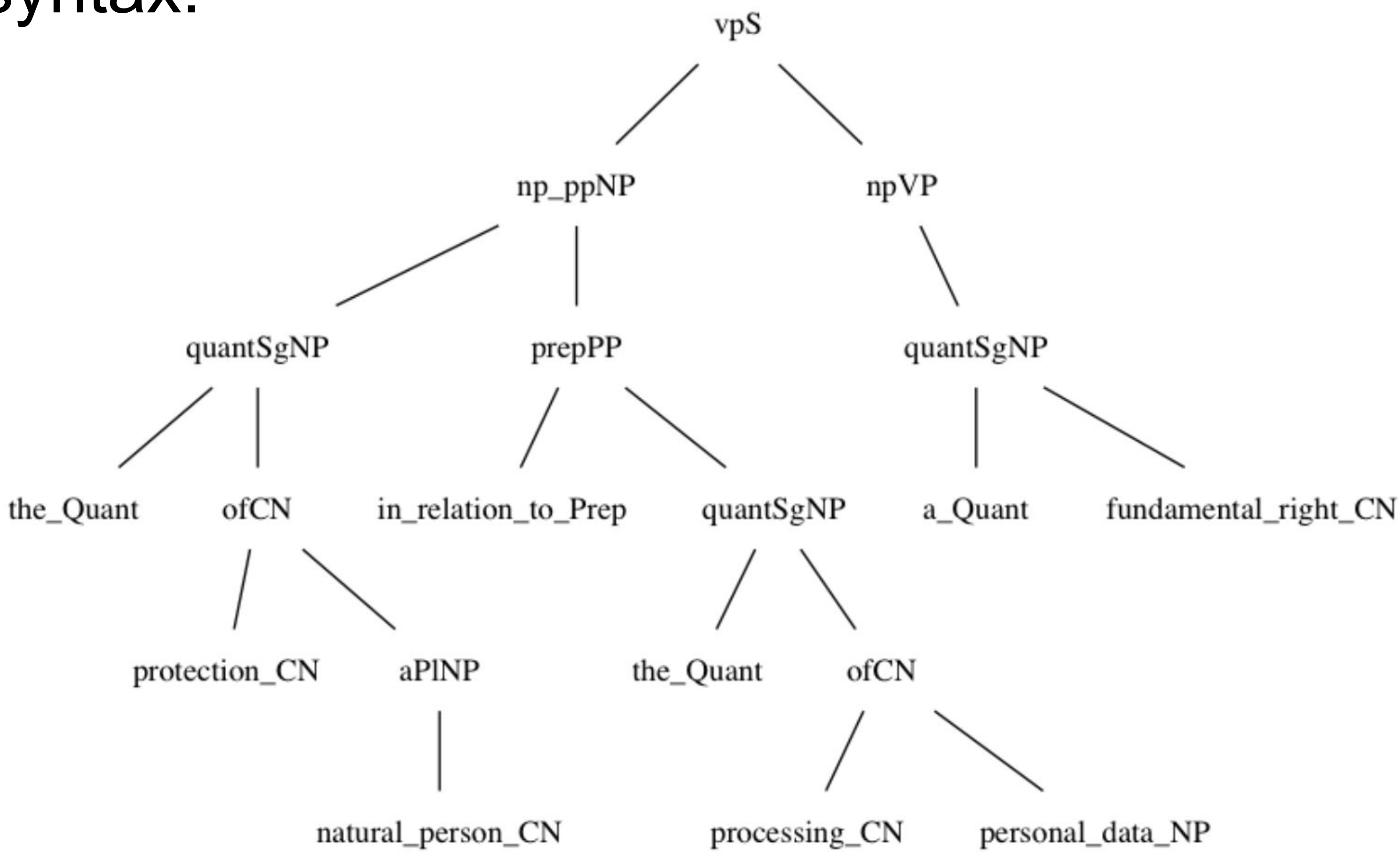
Concept alignments in the first sentence of the GDPR



SIGNATU

digitalGrammars
Language technology to rely on.

Abstract syntax: GDPR



Concept alignment from parallel texts

encourage the establishment of data protection certification mechanisms pursuant to Article 42(1), and approve the criteria of certification pursuant to Article 42(5)

die Einführung von Datenschutzzertifizierungsmechanismen nach Artikel 42 Absatz 1 anregen und Zertifizierungskriterien nach Artikel 42 Absatz 5 billigen

encourage la mise en place de mécanismes de certification en matière de protection des données en application de l'article 42, paragraphe 1, et approuve les critères de certification en application de l'article 42, paragraphe 5

German was a good starting point for identifying these!

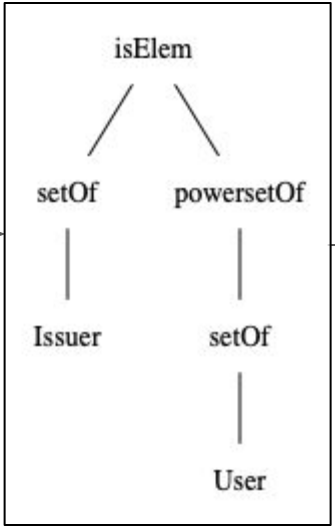
Air Traffic Control - iFACTS

- Tactical support to controllers:
  - Trajectory Prediction
  - Conflict Detection
  - Flight Path Monitoring
- Size of the iFACTS Z specification:
  - Initially developed by a team of 12 engineers.
  - Actively developed or maintained over 10 years.
  - Over 40,000 lines of Z.
  - 287,000 lines of canned "fuzzlisp".
  - The key Z documents amount to over 3,000 pages.
- Improves airspace efficiency.

altran digital Grammars Language technology to rely on.

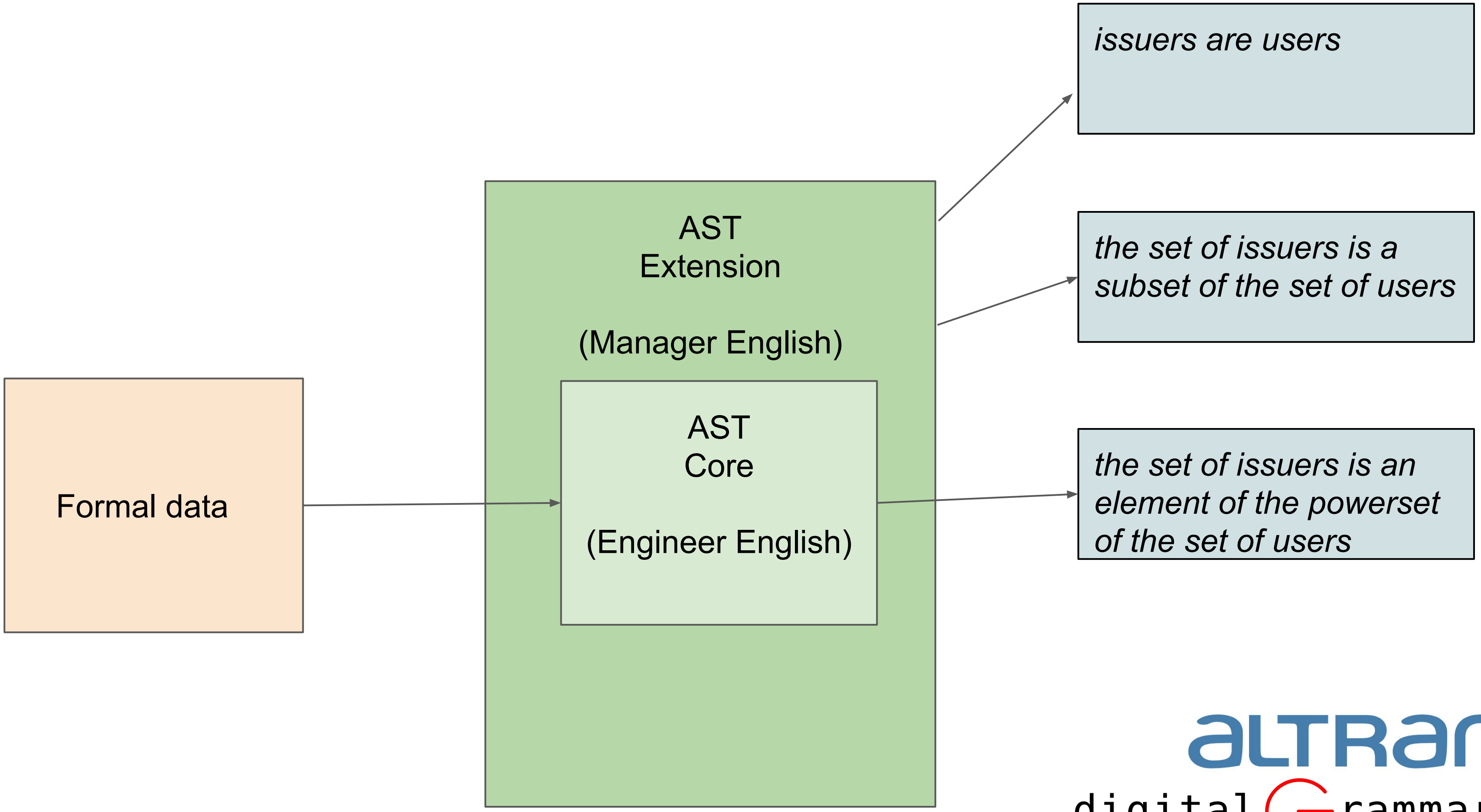
zed2eng

Issuer $\in$ ***P***(User)



the set of issuers is an element of the powerset of the set of users

http://www.grammaticalframework.org/~aarne/final-dg-zed.pdf


Formal data
AST
Extension
(Manager English)
AST
Core
(Engineer English)
issuers are users
the set of issuers is a subset of the set of users
the set of issuers is an element of the powerset of the set of users

NLG functions on the Manager level

inline simple bulleted lists: one of *A *B → A or B

predicate idioms: x is equal to non-exist → x does not exist

X’ = X → X does not change

A = {x} → x is the only A

A ∩ B = {x} → x is the only A in B

expression idioms: optional element of X → optional X

the sum of x, y and z → x plus y plus z

definition idioms: X is a subset of Y → Xs are Ys

X is a member of {A,B,C} → X is A, B or C

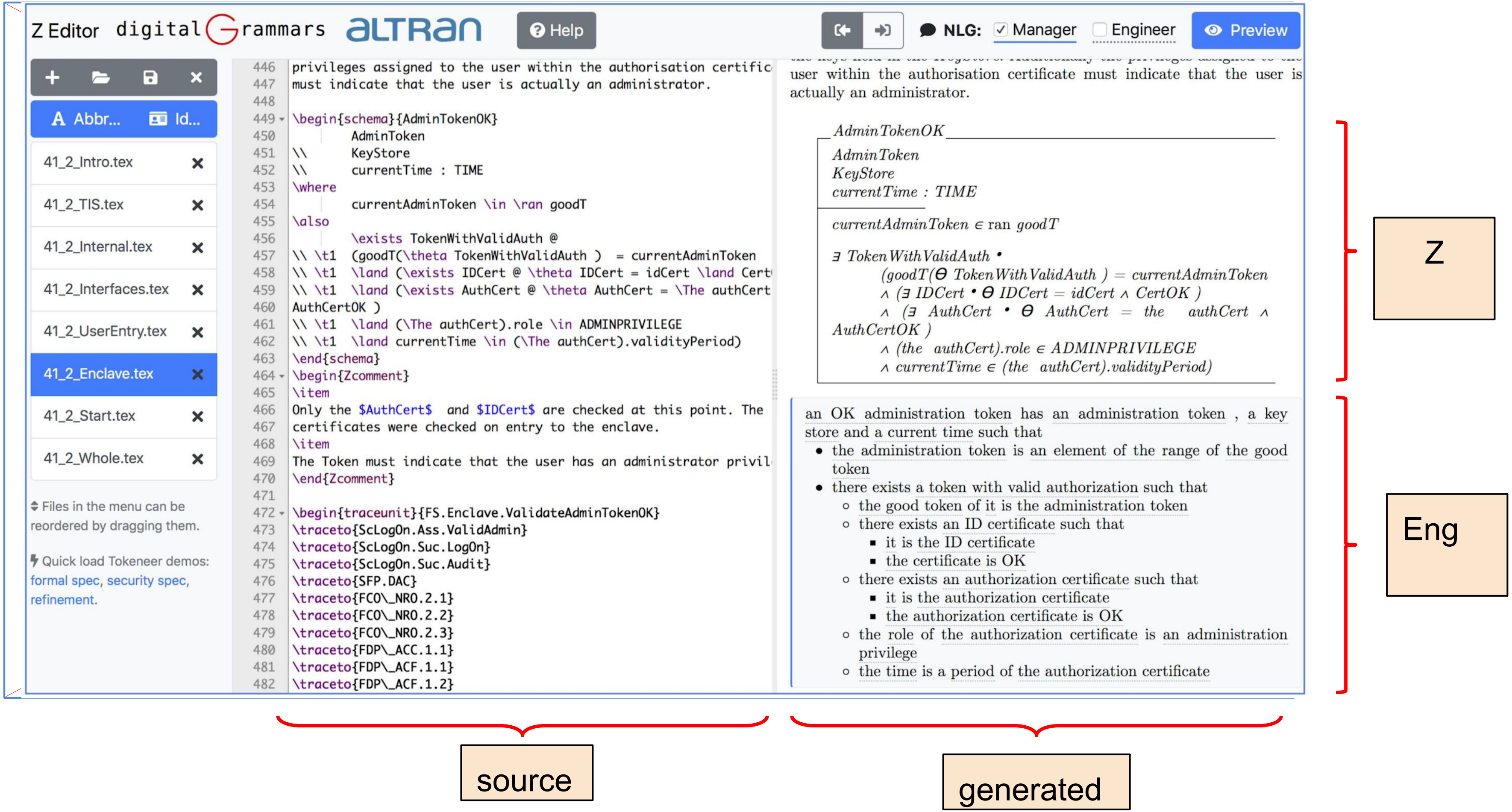
NLG functions on the Manager level: lossy ones

hide declaration types: `x in A → x`

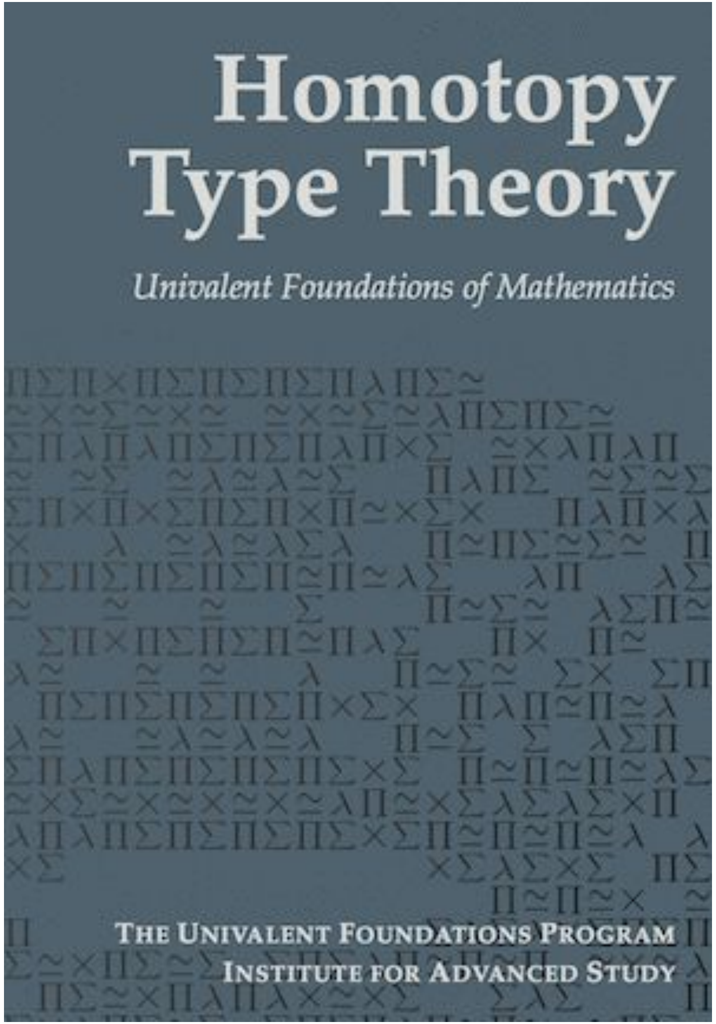
select shorter words: `administration operation → operation`

pronouns: `the operation → it`

`the validation key of it → its validation key`


Z Editor
digital Grammars
altran
Help
NLG:
Manager
Engineer
Preview
Abbr...
Id...
41_2_Intro.tex
41_2_TIS.tex
41_2_Internal.tex
41_2_Interfaces.tex
41_2_UserEntry.tex
41_2_Enclave.tex
41_2_Start.tex
41_2_Whole.tex
Files in the menu can be reordered by dragging them.
Quick load Tokeneer demos: formal spec, security spec, refinement.
privileges assigned to the user within the authorisation certific
must indicate that the user is actually an administrator.
\begin{schema}{AdminTokenOK}
AdminToken
\\ KeyStore
\\ currentTime : TIME
\where
currentAdminToken \in \ran goodT
\also
\exists TokenWithValidAuth @
\\ \t1 (goodT(\theta TokenWithValidAuth) = currentAdminToken
\\ \t1 \land (\exists IDCert @ \theta IDCert = idCert \land Cert
\\ \t1 \land (\exists AuthCert @ \theta AuthCert = \The authCert
AuthCertOK)
\\ \t1 \land (\The authCert).role \in ADMINPRIVILEGE
\\ \t1 \land currentTime \in (\The authCert).validityPeriod)
\end{schema}
\begin{Zcomment}
\item
Only the $AuthCert$ and $IDCert$ are checked at this point. The
certificates were checked on entry to the enclave.
\item
The Token must indicate that the user has an administrator privil
\end{Zcomment}
\begin{traceunit}{FS.Enclave.ValidateAdminTokenOK}
\traceto{ScLogOn.Ass.ValidAdmin}
\traceto{ScLogOn.Suc.LogOn}
\traceto{ScLogOn.Suc.Audit}
\traceto{SFP.DAC}
\traceto{FCO_NRO.2.1}
\traceto{FCO_NRO.2.2}
\traceto{FCO_NRO.2.3}
\traceto{FDP_ACC.1.1}
\traceto{FDP_ACF.1.1}
\traceto{FDP_ACF.1.2}
user within the authorisation certificate must indicate that the user is actually an administrator.
AdminTokenOK
AdminToken
KeyStore
currentTime : TIME
currentAdminToken ∈ ran goodT
∃ TokenWithValidAuth •
(goodT(θ TokenWithValidAuth) = currentAdminToken
∧ (∃ IDCert • θ IDCert = idCert ∧ CertOK)
∧ (∃ AuthCert • θ AuthCert = the authCert ∧ AuthCertOK)
∧ (the authCert).role ∈ ADMINPRIVILEGE
∧ currentTime ∈ (the authCert).validityPeriod)
an OK administration token has an administration token , a key store and a current time such that
the administration token is an element of the range of the good token
there exists a token with valid authorization such that
the good token of it is the administration token
there exists an ID certificate such that
it is the ID certificate
the certificate is OK
there exists an authorization certificate such that
it is the authorization certificate
the authorization certificate is OK
the role of the authorization certificate is an administration privilege
the time is a period of the authorization certificate
Z
Eng
source
generated

5.
Scale up from CNL to legacy text and gradually to open text



First edition 2013

Idea: develop texts parallel to formal proofs (Agda and Coq).

Abstractions in text ~ abstractions in LF

However, no formal relation between formal proof and text yet

Our project:

- parse the text and generate checkable proofs;
- see how far you can get
- develop a library of proof idioms as you go

(starting as MSc thesis of Warrick Macmillan)

PDF

Definition: A type A is contractible, if there is $a : A$, called the center of contraction, such that for all $x : A$, $a = x$.

Definition: A map $f : A \to B$ is an equivalence, if for all $y : B$, its fiber, $\{x : A \mid fx = y\}$, is contractible. We write $A \simeq B$, if there is an equivalence $A \to B$.

Lemma: For each type A, the identity map, $1_A := \lambda_{x:A} x : A \to A$, is an equivalence.

Proof: For each $y : A$, let $\{y\}_A := \{x : A \mid x = y\}$ be its fiber with respect to 1_A and let $\bar{y} := (y, r_A y) : \{y\}_A$. As for all $y : A$, $(y, r_A y) = y$, we may apply Id-induction on y, $x : A$ and $z : (x = y)$ to get that

$$(x, z) = y$$

. Hence, for $y : A$, we may apply Σ -elimination on $u : \{y\}_A$ to get that $u = y$, so that $\{y\}_A$ is contractible. Thus, $1_A : A \to A$ is an equivalence. $\square$

Corollary: If U is a type universe, then, for $X, Y : U$,

$$(*) X = Y \to X \simeq Y$$

.

Proof: We may apply the lemma to get that for $X : U$, $X \simeq X$. Hence, we may apply Id-induction on $X, Y : U$ to get that $(*)$. $\square$

Definition: A type universe U is univalent, if for $X, Y : U$, the map $E_{X,Y} : X = Y \to X \simeq Y$ in $(*)$ is an equivalence.

```latex
\documentclass{article}
 \usepackage[utf8]{inputenc}
 \usepackage{latexsym}
 \usepackage{amsfonts}
 \begin{document}
 \input{macros}

 \textbf{Definition}:
 A type $A$ is contractible, if there is $a : A$, called the center of contraction, such that for all $x : A$,
$\equalH {a}{x}$.

 \textbf{Definition}:
 A map $f : \arrowH {A}{B}$ is an equivalence, if for all $y : B$, its fiber, $\comprehensionH {x}{A}{\equalH {\appH
{f}{x}}{y}}$, is contractible.
 We write $\equivalenceH {A}{B}$, if there is an equivalence $\arrowH {A}{B}$.

 \textbf{Lemma}:
 For each type $A$, the identity map, $\defineH {\idMapH {A}}{\typingH {\lambdaH {x}{A}{x}}{\arrowH {A}{A}}}$, is an
equivalence.

 \textbf{Proof}:
 For each $y : A$, let $\defineH {\fiberH {y}{A}}{\comprehensionH {x}{A}{\equalH {x}{y}}}$ be its fiber with respect
to $\idMapH {A}$ and let $\defineH {\barH {y}}{\typingH {\pairH {y}{\reflexivityH {A}{y}}}{\fiberH {y}{A}}}$.
 As for all $y : A$, $\equalH {\pairH {y}{\reflexivityH {A}{y}}}{y}$, we may apply Id-induction on $y$, $\typingH
{x}{A}$ and $\typingH {z}{\idPropH {x}{y}}$ to get that \[\equalH {\pairH {x}{z}}{y}\].
 Hence, for $y : A$, we may apply $\Sigma$ -elimination on $\typingH {u}{\fiberH {y}{A}}$ to get that $\equalH
{u}{y}$, so that $\fiberH {y}{A}$ is contractible.
 Thus, $\typingH {\idMapH {A}}{\arrowH {A}{A}}$ is an equivalence.
  $\Box$

 \textbf{Corollary}:
 If $U$ is a type universe, then, for $X , Y : U$, \[(*)\arrowH {\equalH {X}{Y}}{\equivalenceH {X}{Y}}\].

 \textbf{Proof}:
 We may apply the lemma to get that for $X : U$, $\equivalenceH {X}{X}$.
 Hence, we may apply Id-induction on $\typingH {X , Y}{U}$ to get that $(*)$.
  $\Box$


 \textbf{Definition}:
 A type universe $U$ is univalent, if for $X , Y : U$, the map $\equivalenceMapH {X}{Y}: \arrowH {\equalH
{X}{Y}}{\equivalenceH {X}{Y}}$ in $(*)$ is an equivalence.

 \end{document}
```

LaTeX

AST

```
FormatParagraph DocumentStyleFormat
FormatParagraph BeginDocumentFormat
FormatParagraph InputMacrosFormat
TitleParagraph DefinitionTitle
DefPredParagraph type_Sort A_Var contractible_Pred (ExistCalledProp a_Var (ExpSort (VarExp A_Var)) (FunInd centre_of_contraction_Fun) (ForAllProp (BaseVar x_Var)
(ExpSort (VarExp A_Var)) (ExpProp (equalExp (VarExp a_Var) (VarExp x_Var)))))
FormatParagraph EmptyLineFormat
TitleParagraph DefinitionTitle
DefPredParagraph (mapSort (mapExp (VarExp A_Var) (VarExp B_Var))) f_Var equivalence_Pred (ForAllProp (BaseVar y_Var) (ExpSort (VarExp B_Var)) (PredProp
contractible_Pred (AliasInd (AppFunItInd fiber_Fun) (FunInd (ExpFun (ComprehensionExp x_Var (VarExp A_Var) (equalExp (AppExp f_Var (VarExp x_Var)) (VarExp
y_Var))))))))
DefPropParagraph (ExpProp (equivalenceExp (VarExp A_Var) (VarExp B_Var))) (ExistSortProp (equivalenceSort (mapExp (VarExp A_Var) (VarExp B_Var))))
FormatParagraph EmptyLineFormat
TitleParagraph LemmaTitle
TheoremParagraph (ForAllProp (BaseVar A_Var) type_Sort (PredProp equivalence_Pred (AliasInd (FunInd identity_map_Fun) (FunInd (ExpFun (DefExp (identityMapExp (VarExp
A_Var)) (TypedExp (BaseExp (lambdaExp x_Var (VarExp A_Var) (VarExp x_Var))) (mapExp (VarExp A_Var) (VarExp A_Var)))))))))
FormatParagraph EmptyLineFormat
TitleParagraph ProofTitle
AssumptionParagraph (ConsAssumption (ForAssumption y_Var (ExpSort (VarExp A_Var)) (LetAssumption (FunInd (ExpFun (DefExp (fiberExp (VarExp y_Var) (VarExp A_Var))
(ComprehensionExp x_Var (VarExp A_Var) (equalExp (VarExp x_Var) (VarExp y_Var)))))) (AppFunItInd (fiberWrt_Fun (FunInd (ExpFun (identityMapExp (VarExp A_Var))))))))
(BaseAssumption (LetExpAssumption (barExp (VarExp y_Var)) (TypedExp (BaseExp (pairExp (VarExp y_Var) (reflexivityExp (VarExp A_Var) (VarExp y_Var)))) (fiberExp (VarExp
y_Var) (VarExp A_Var))))))
ConclusionParagraph (AsConclusion (ForAllProp (BaseVar y_Var) (ExpSort (VarExp A_Var)) (ExpProp (equalExp (pairExp (VarExp y_Var) (reflexivityExp (VarExp A_Var)
(VarExp y_Var))) (VarExp y_Var)))) (ApplyLabelConclusion id_induction_Label (ConsInd (FunInd (ExpFun (VarExp y_Var))) (ConsInd (FunInd (ExpFun (TypedExp (BaseExp
(VarExp x_Var)) (VarExp A_Var)))) (ConsInd (FunInd (ExpFun (TypedExp (BaseExp (VarExp z_Var)) (idPropExp (VarExp x_Var) (VarExp y_Var))))) BaseInd))) (DisplayExpProp
(equalExp (pairExp (VarExp x_Var) (VarExp z_Var)) (VarExp y_Var)))))
ConclusionSoThatParagraph (ForConclusion (BaseVar y_Var) (ExpSort (VarExp A_Var)) (ApplyLabelConclusion sigma_elimination_Label (ConsInd (FunInd (ExpFun (TypedExp
(BaseExp (VarExp u_Var)) (fiberExp (VarExp y_Var) (VarExp A_Var))))) BaseInd) (ExpProp (equalExp (VarExp u_Var) (VarExp y_Var))))) (PredProp contractible_Pred (FunInd
(ExpFun (fiberExp (VarExp y_Var) (VarExp A_Var)))))
ConclusionParagraph (PropConclusion (PredProp equivalence_Pred (FunInd (ExpFun (TypedExp (BaseExp (identityMapExp (VarExp A_Var))) (mapExp (VarExp A_Var) (VarExp
A_Var)))))))
QEDParagraph
FormatParagraph EmptyLineFormat
TitleParagraph CorollaryTitle
TheoremParagraph (ForAllProp (BaseVar U_Var) type_universe_Sort (ForAllProp (ConsVar X_Var (BaseVar Y_Var)) (ExpSort (VarExp U_Var)) (DisplayLabelledExpProp StarLabel
(mapExp (equalExp (VarExp X_Var) (VarExp Y_Var)) (equivalenceExp (VarExp X_Var) (VarExp Y_Var))))))
FormatParagraph EmptyLineFormat
TitleParagraph ProofTitle
ConclusionParagraph (ApplyLabelConclusion the_lemma_Label BaseInd (ForAllProp (BaseVar X_Var) (ExpSort (VarExp U_Var)) (ExpProp (equivalenceExp (VarExp X_Var) (VarExp
X_Var)))))
ConclusionParagraph (ApplyLabelConclusion id_induction_Label (ConsInd (FunInd (ExpFun (TypedExp (ConsExp (VarExp X_Var) (BaseExp (VarExp Y_Var))) (VarExp U_Var))))
BaseInd) (LabelProp StarLabel))
QEDParagraph
TitleParagraph DefinitionTitle
DefPredParagraph type_universe_Sort U_Var univalent_Pred (ForAllProp (ConsVar X_Var (BaseVar Y_Var)) (ExpSort (VarExp U_Var)) (PredProp equivalence_Pred (IndInLabel
(TheSortInd (mapSort (mapExp (equalExp (VarExp X_Var) (VarExp Y_Var)) (equivalenceExp (VarExp X_Var) (VarExp Y_Var)))) (equivalenceMapExp (VarExp X_Var) (VarExp
Y_Var))) StarLabel)))
FormatParagraph EndDocumentFormat
```

```
-- Definition
 (A : Type)=> Contractible A := exist ((a : A))((x : A)-> Id (a)(x)); ; CenterContraction ;

 -- Definition
 (f : app (Map)(A -> B))=> Equivalence f := (y : B)-> (Contractible (fiber it)); ; ; fiber it := comprehension
(x)(A)(Id (f (x))(y))
 equivalence (A)(B):= Equivalence (A -> B);

 -- Lemma
 => (A : Type)-> (Equivalence (id)); ; ; id := idMap (A):= (\ x : A -> x): A -> A

 -- Proof
 <= (y : A)=> fiber (y)(A):= comprehension (x)(A)(Id (x)(y)):= fiber (idMap (A))it ; ; ; bar (y):= < y , refl
(A)(y)> : fiber (y)(A)
 => (y : A)-> Id (< y , refl (A)(y)>)(y)=> IdInd y x : A z : Id (x)(y): Id (< x , z >)(y);
 => (y : A)=> SigmaEl u : fiber (y)(A): Id (u)(y); (Contractible (fiber (y)(A)));
 => (Equivalence (idMap (A): A -> A));
 QED

 -- Corollary
 => (U : Universe)-> (X , Y : U)-> Id (X)(Y)-> equivalence (X)(Y); it := Id (X)(Y)-> equivalence (X)(Y)

 -- Proof
 => Lemma : (X : U)-> equivalence (X)(X)
 => IdInd X , Y : U : it
 QED


 -- Definition
 (U : Universe)=> Univalent U := (X , Y : U)-> (Equivalence (((EqMap (X)(Y): app (Map)(Id (X)(Y)-> equivalence
(X)(Y))): it)));
```

TT

French

```
-- Definition
 (A : Type)=> Contracti

 -- Definition
 (f : app (Map)(A -> B)
(x)(A)(Id (f (x))(y))
 equivalence (A)(B):= E

 -- Lemma
 => (A : Type)-> (Equiv

 -- Proof
 <= (y : A)=> fiber (y)
(A)(y)> : fiber (y)(A)
 => (y : A)-> Id (< y ,
 => (y : A)=> SigmaEl u
 => (Equivalence (idMap
 QED

 -- Corollary
 => (U : Universe)-> (X

 -- Proof
 => Lemma : (X : U)-> e
 => IdInd X , Y : U : i
 QED


 -- Definition
 (U : Universe)=> Univa
(X)(Y))): it)));
```

Définition: Un type A est contractile, s'il existe un $a : A$, nommé le centre de contraction, tel que pour tous les $x : A$, $a = x$.

Définition: Une application $f : A \to B$ est une équivalence, si pour tous les $y : B$, sa fibre, $\{x : A \mid fx = y\}$, est contractile. Nous écrivons $A \simeq B$, s'il existe une équivalence $A \to B$.

Lemme: Pour tout type A, l'identité, $1_A := \lambda_{x:A} x : A \to A$, est une équivalence.

Démonstration: Pour tout $y : A$, soit $\{y\}_A := \{x : A \mid x = y\}$ sa fibre par rapport de 1_A et soit $\bar{y} := (y, r_A y) : \{y\}_A$. Comme pour tous les $y : A$, $(y, r_A y) = y$, nous pouvons appliquer Id-induction sur y, $x : A$ et $z : (x = y)$ pour obtenir que

$$(x, z) = y$$

. Donc, pour les $y : A$, nous pouvons appliquer Σ -élimination sur $u : \{y\}_A$ pour obtenir que $u = y$, de façon que $\{y\}_A$ soit contractile. Alors, $1_A : A \to A$ est une équivalence. □

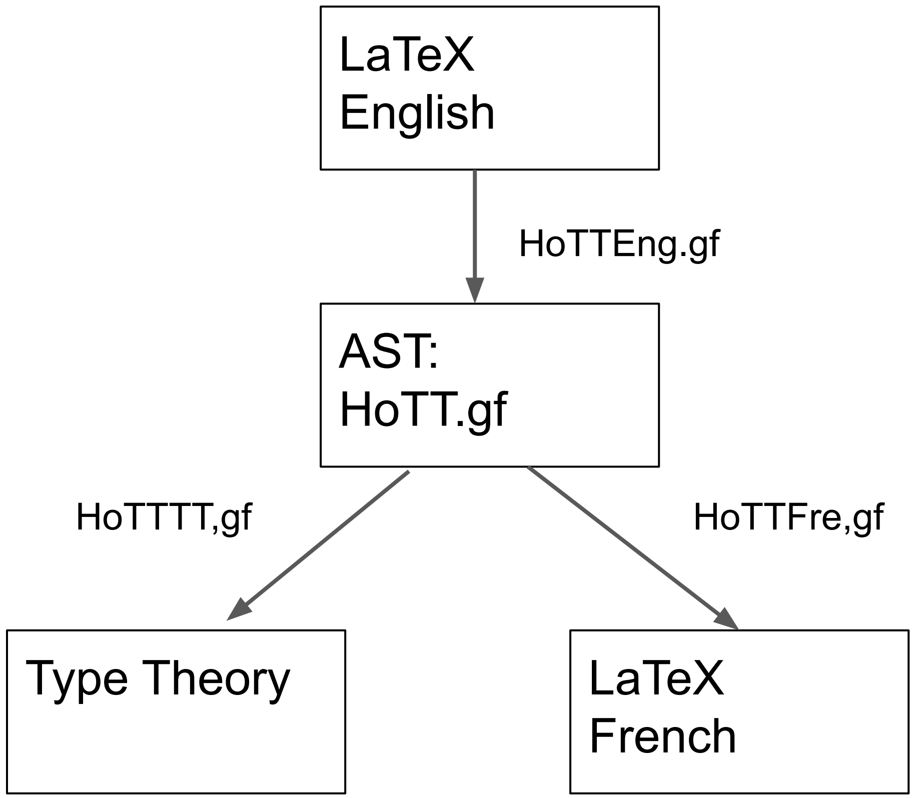
Corollaire: Si U est un univers, alors, pour les $X, Y : U$,

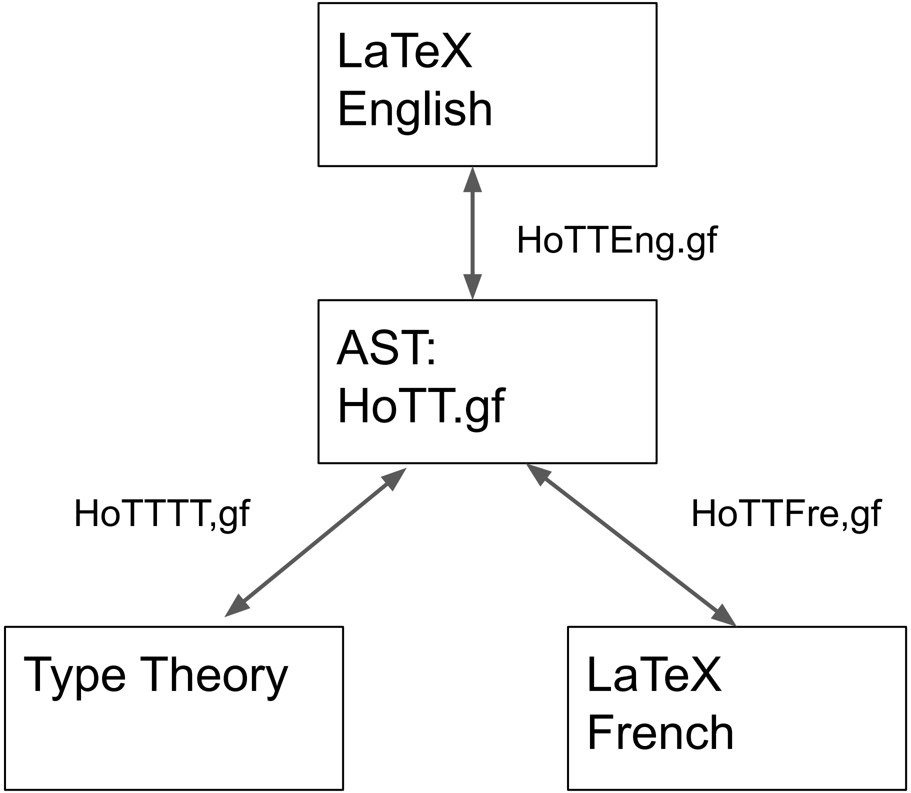
$$(*) X = Y \to X \simeq Y$$

.

Démonstration: Nous pouvons appliquer le lemme pour obtenir que pour les $X : U$, $X \simeq X$. Donc, nous pouvons appliquer Id-induction sur $X, Y : U$ pour obtenir que $(*)$. □

Définition: Un univers U est univalent, si pour les $X, Y : U$, l'application $E_{X,Y} : X = Y \to X \simeq Y$ dans $(*)$ est une équivalence.


LaTeX
English
HoTTEng.gf
AST:
HoTT.gf
HoTTTT,gf
HoTTFre,gf
Type Theory
LaTeX
French

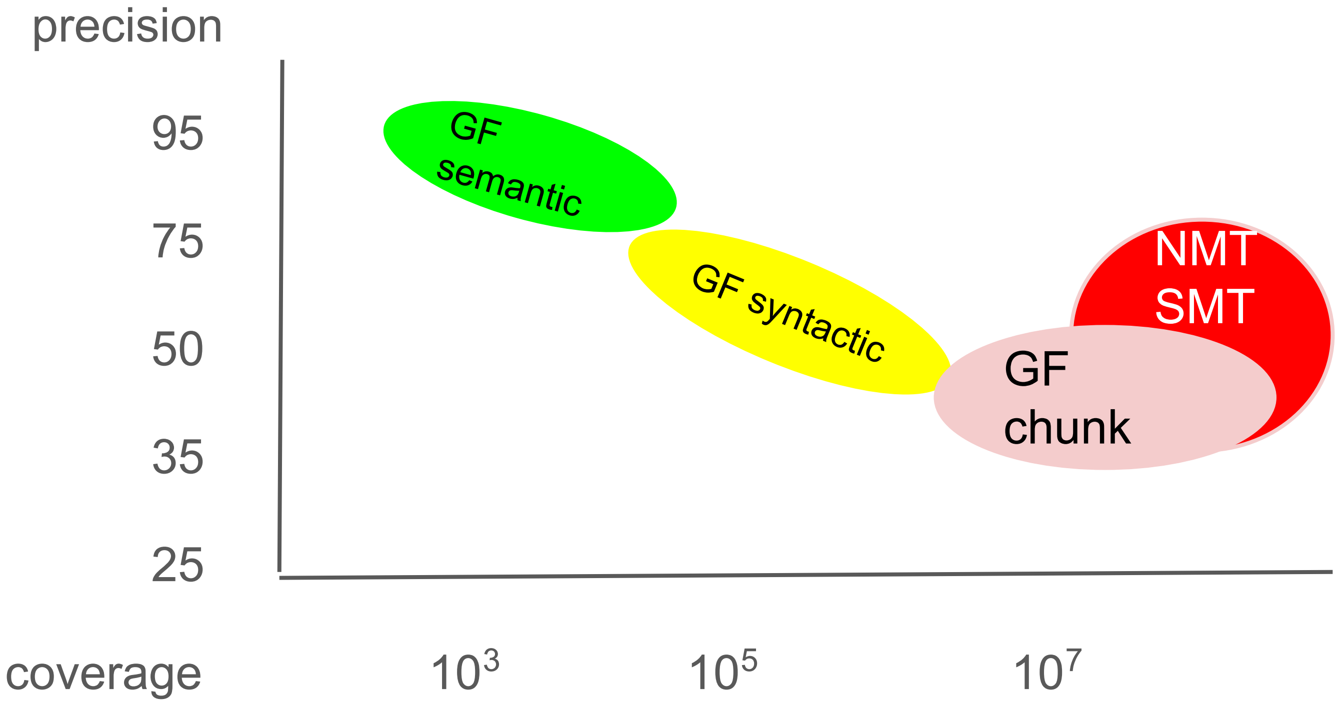

LaTeX
English
HoTTEng.gf
AST:
HoTT.gf
HoTTTT,gf
HoTTFre,gf
Type Theory
LaTeX
French

```
$ wc -l *.gf
      54 Formulas.gf
      90 FormulasLatex.gf
      85 FormulasLogic.gf
     104 Framework.gf
       6 FrameworkEng.gf
      22 FrameworkFre.gf
     163 FrameworkFunctor.gf
      67 FrameworkInstanceEng.gf
      68 FrameworkInstanceFre.gf
     127 FrameworkInterface.gf
     139 FrameworkLogic.gf
      22 HottLexicon.gf
      38 HottLexiconEng.gf
      38 HottLexiconFre.gf
      35 HottLexiconLogic.gf
      36 HottLexiconSwe.gf
    1094 total
```

```
concrete FormulasLatex of Formulas = open Prelude in {

lin
  mapExp x y = optLatex
    (macro2 "arrowH" a b)
    (a ++ macro "rightarrow" ++ b) ;

  ComprehensionExp x A P = optLatex
    (macro3 "comprehensionH" a b c)
    ("\\{" ++ a ++ ":" ++ b ++ macro "mid" ++ c ++ "\\}") ;


oper
  macro  : Str -> Str =
    \m -> "\\" + m ;
  macro2 : (f,x,y : Str) -> Str =
    \f,x,y -> "\\" + f ++ "{" ++ x ++ "} {" ++ y ++ "}" ;

-- convert flat expression b to macro expression a
  optLatex : Str -> Str -> Str = \a,b -> a | b ;
```

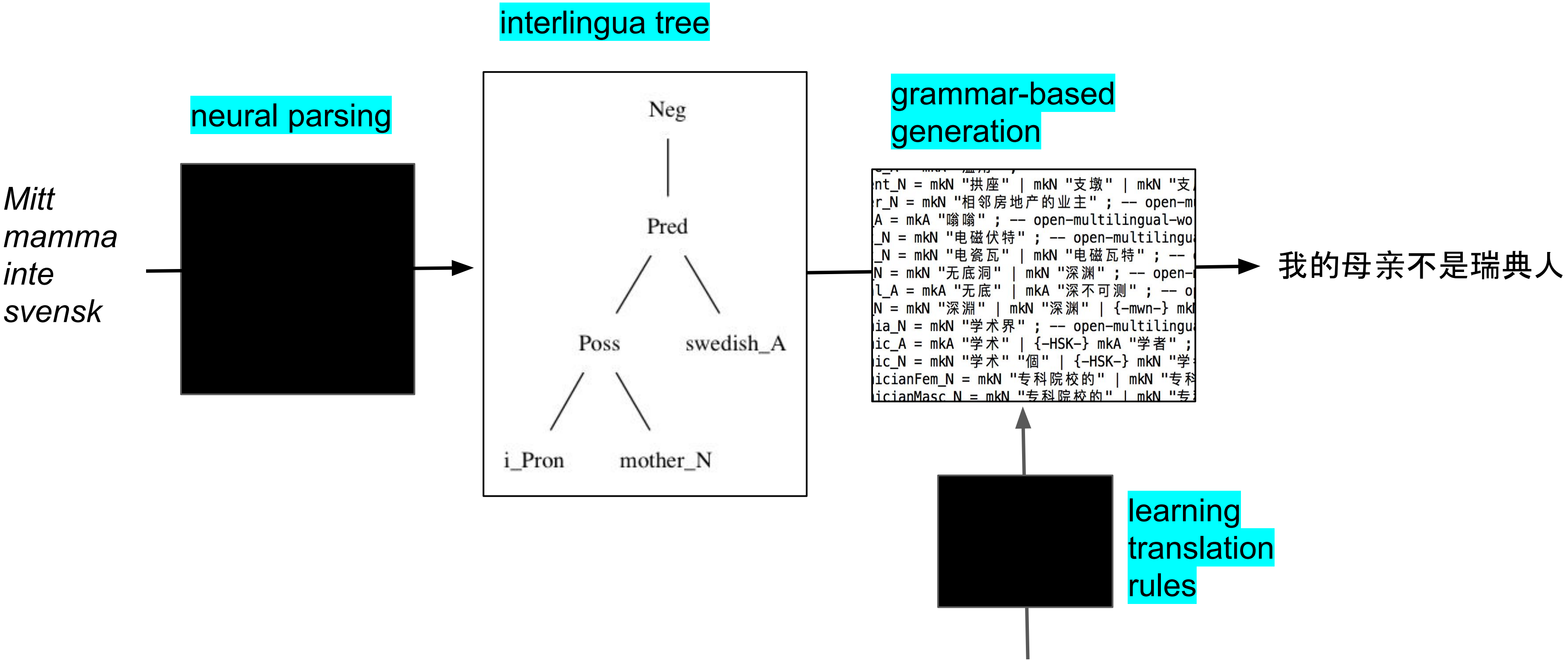
```
> p -cat=Exp "\\{ f : y \\mid u \\}" | l

\comprehensionH { f } { y } { u }
```

Graceful degradation



Wide-coverage translation

GF + black box machine learning



Take home

You can use GF to

Parse, generate, translate

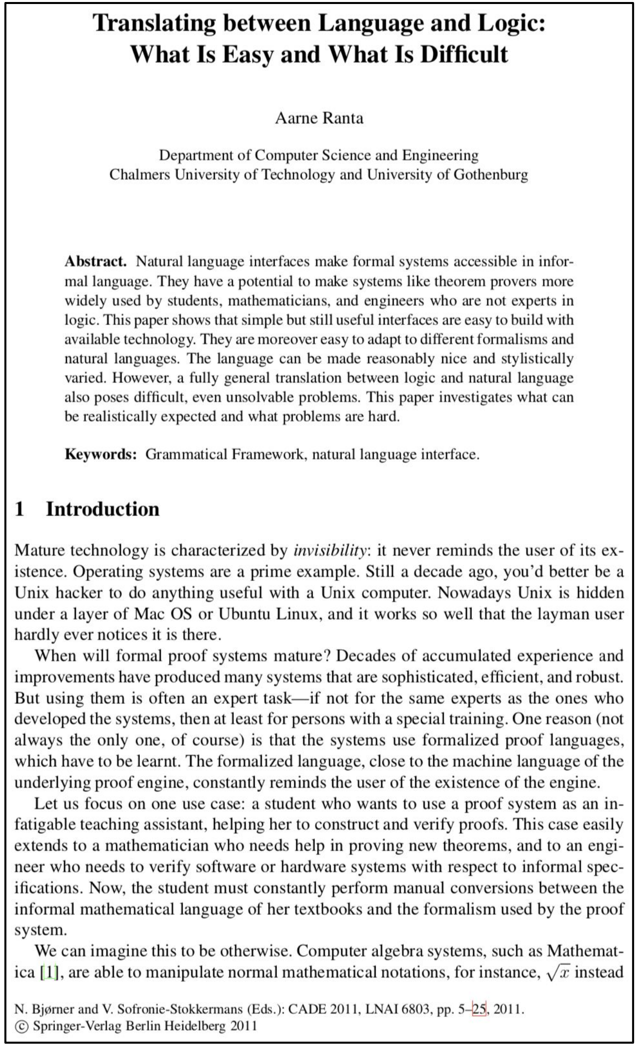
Get the details of 40 languages from the RGL

Build web and mobile interfaces with editing help

Define and process abstract syntax trees

Embed in Haskell, Python, Java, C, …

Integrate with your own system for input and output



Translating between Language and Logic: What Is Easy and What Is Difficult

Aarne Ranta

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract. Natural language interfaces make formal systems accessible in informal language. They have a potential to make systems like theorem provers more widely used by students, mathematicians, and engineers who are not experts in logic. This paper shows that simple but still useful interfaces are easy to build with available technology. They are moreover easy to adapt to different formalisms and natural languages. The language can be made reasonably nice and stylistically varied. However, a fully general translation between logic and natural language also poses difficult, even unsolvable problems. This paper investigates what can be realistically expected and what problems are hard.

Keywords: Grammatical Framework, natural language interface.

1 Introduction

Mature technology is characterized by *invisibility*: it never reminds the user of its existence. Operating systems are a prime example. Still a decade ago, you'd better be a Unix hacker to do anything useful with a Unix computer. Nowadays Unix is hidden under a layer of Mac OS or Ubuntu Linux, and it works so well that the layman user hardly ever notices it is there.

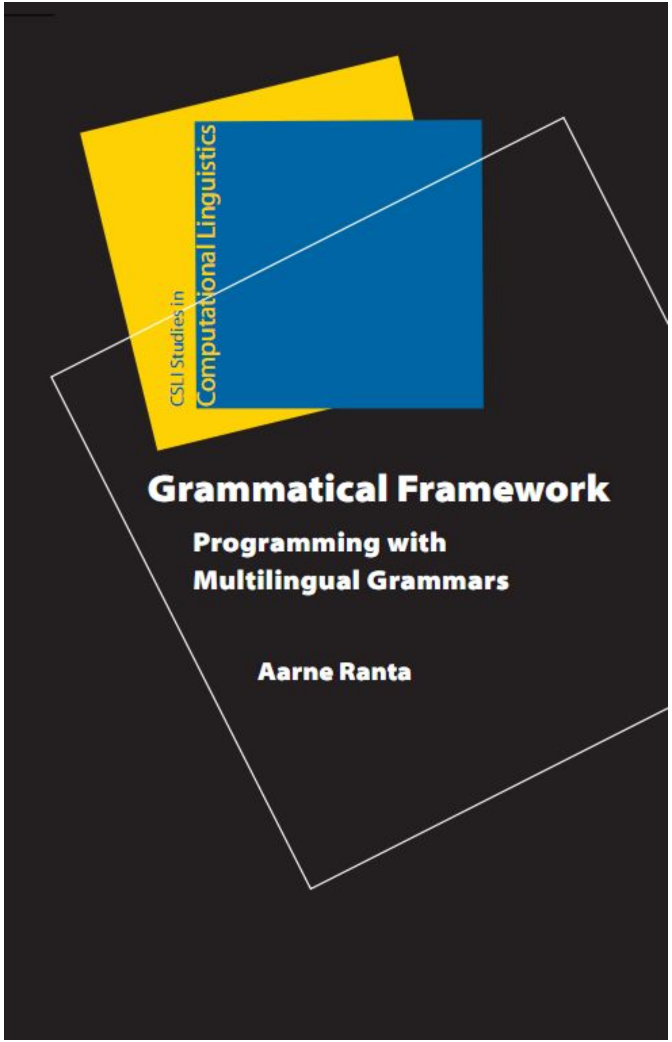
When will formal proof systems mature? Decades of accumulated experience and improvements have produced many systems that are sophisticated, efficient, and robust. But using them is often an expert task—if not for the same experts as the ones who developed the systems, then at least for persons with a special training. One reason (not always the only one, of course) is that the systems use formalized proof languages, which have to be learnt. The formalized language, close to the machine language of the underlying proof engine, constantly reminds the user of the existence of the engine.

Let us focus on one use case: a student who wants to use a proof system as an indefatigable teaching assistant, helping her to construct and verify proofs. This case easily extends to a mathematician who needs help in proving new theorems, and to an engineer who needs to verify software or hardware systems with respect to informal specifications. Now, the student must constantly perform manual conversions between the informal mathematical language of her textbooks and the formalism used by the proof system.

We can imagine this to be otherwise. Computer algebra systems, such as Mathematica [1], are able to manipulate normal mathematical notations, for instance, $\sqrt{x}$ instead

N. Bjørner and V. Sofronie-Stokkermans (Eds.): CADE 2011, LNAI 6803, pp. 5–25, 2011.

CADE 2011



CSLI, Stanford, 2011

Seventh GF Summer School 2020

Singapore

3rd–14th August 2020

About Registration Programme Accommodation Venue Organizers Contacts
Logistics Previous schools Bid

Material shown in the GF tutorial

Minibar on the cloud: http://cloud.grammaticalframework.org/minibar/minibar.html

On-line grammar editor: grammar Bonn in http://cloud.grammaticalframework.org/gfse/

The CADE grammar: https://github.com/GrammaticalFramework/gf-contrib/tree/master/cade-2011

The HoTT grammar: https://github.com/GrammaticalFramework/gf-contrib/tree/master/homotopy-typetheory