Grammars for the Working Programmer: BNFC and GF

Aarne Ranta

University of Gothenburg and Digital Grammars AB

SPLST-2015

Tampere, 9 October 2015

Outline

BNFC: a compiler compiler compiler

GF: compiling natural language

BNFC

compiler

compiler

parser

compiler

compiler compiler

- YACC

parser

parser generator

compiler

compiler compiler

compiler compiler compiler

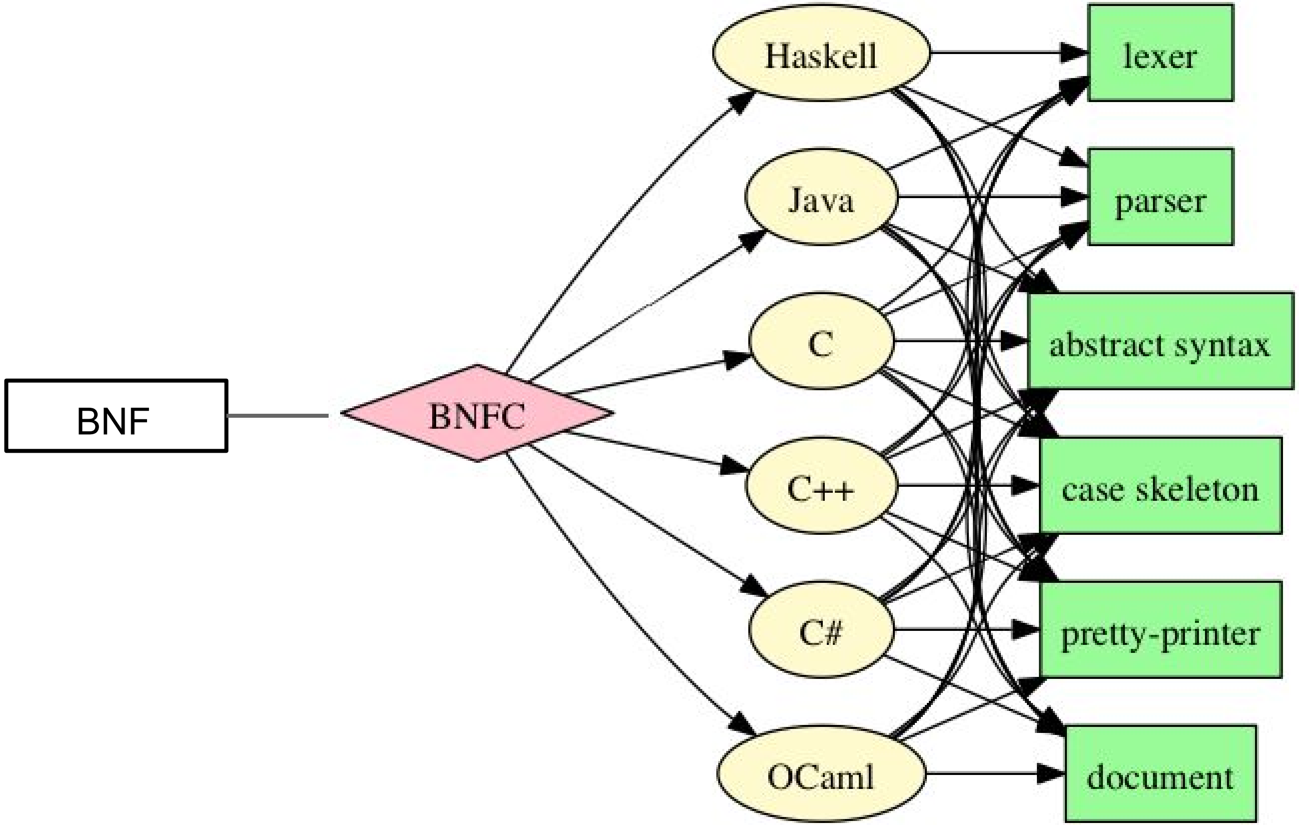
parser

parser generator

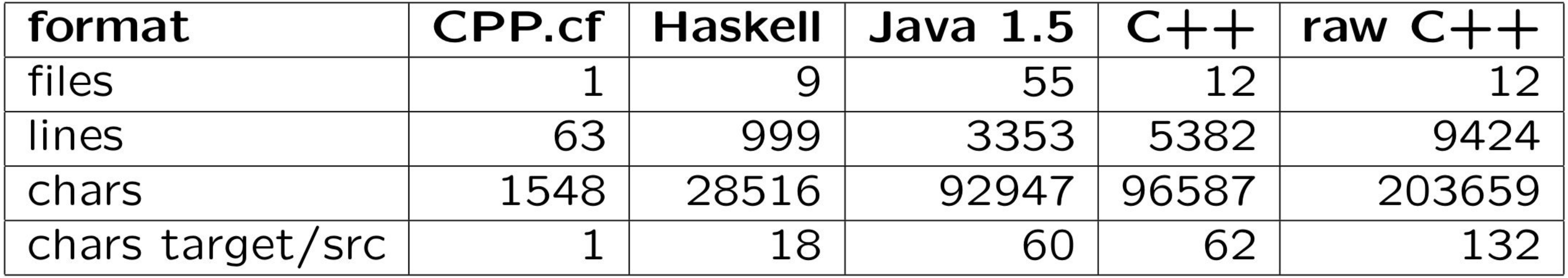
parser generator generator

compiler	parser
compiler compiler	parser generator
compiler compiler compiler	parser lexer abstract syntax translator documentation generator

BNFC = BNF Converter



Conciseness



format	CPP.cf	Haskell	Java 1.5	C++	raw C++
files	1	9	55	12	12
lines	63	999	3353	5382	9424
chars	1548	28516	92947	96587	203659
chars target/src	1	18	60	62	132

The number of bugs per line is independent of programming language.

Eric S. Raymond, The Art of Unix Programming

BNFC source

```
-- file Calc.bnf

EAdd. Exp0 ::= Exp0 "+" Exp1 ;
ESub. Exp0 ::= Exp0 "-" Exp1 ;
EMul. Exp1 ::= Exp1 "*" Exp2 ;
EDiv. Exp1 ::= Exp1 "/" Exp2 ;
EInt. Exp2 ::= Integer ;

coercions Exp 2 ;
```

Running BNFC

```
% bnfc -m --haskell Calc.bnf

  writing file AbsCalc.hs    # abstract syntax
  writing file LexCalc.x     # lexer
  writing file ParCalc.y     # parser
  writing file DocCalc.tex   # language document
  writing file SkelCalc.hs   # syntax-directed translation skeleton
  writing file PrintCalc.hs  # pretty-printer
  writing file TestCalc.hs   # top-level test program
  writing file ErrM.hs       # monad for error handling
  writing file Makefile      # Makefile
```

Testing the parser in Haskell

```
% make



% echo "5 + 6 * 7" | ./TestCalc
  Parse Successful!

  [Abstract Syntax]
  EAdd (EInt 5) (EMul (EInt 6) (EInt 7))

  [Linearized tree]
  5 + 6 * 7
```

Running BNFC for Java

```
%  bnfc -m --java Calc.cf
  Calc/Absyn/Exp.java        # abstract syntax
  Calc/Absyn/EAdd.java       # also ESub, EMul, EDiv, EInt
  Calc/PrettyPrinter.java    # pretty-printer
  Calc/VisitSkel.java        # syntax-directed translation skeleton
  Calc/ComposVisitor.java    # utilities for syntax-directed translation
  Calc/AbstractVisitor.java
  Calc/FoldVisitor.java
  Calc/AllVisitor.java
  Calc/Test.java             # top-level test file
  Calc/Yylex                 # lexer
  Calc/Calc.cup              # parser
  Calc.tex                   # language document
  Makefile                   # Makefile
```

Testing the parser in Java

```
% make



% echo "5 + 6 * 7" | java Calc/Test
  Parse Successful!

  [Abstract Syntax]
  EAdd (EInt 5) (EMul (EInt 6) (EInt 7))

  [Linearized tree]
  5 + 6 * 7
```

C, C++, C#, OCaml

```
bnfc -m --c      Calc.bnf
bnfc -m --cpp    Calc.bnf
bnfc -m --csharp Calc.bnf
bnfc -m --ocaml  Calc.bnf
```

The “theory” behind BNFC

Abstract syntax from grammar:

- ignore precedence levels
- ignore terminals

Parser returns abstract syntax

Later phases by **pattern matching** on abstract syntax

A Appel, *Modern Compiler Implementation in ML/C/Java*, CUP 1998.

```
EAdd. Exp0 ::= Exp0 "+" Exp1 ;
ESub. Exp0 ::= Exp0 "-" Exp1 ;
EMul. Exp1 ::= Exp1 "*" Exp2 ;
EDiv. Exp1 ::= Exp1 "/" Exp2 ;
EInt. Exp2 ::= Integer ;

coercions Exp 2 ;
```

```
EAdd. Exp  ::= Exp  "+" Exp1 ;
ESub. Exp  ::= Exp  "-" Exp1 ;
EMul. Exp1 ::= Exp1 "*" Exp2 ;
EDiv. Exp1 ::= Exp1 "/" Exp2 ;
EInt. Exp2 ::= Integer ;


coercions Exp 2 ;
```

```
data Exp =
   EAdd Exp Exp
 | ESub Exp Exp
 | EMul Exp Exp
 | EDiv Exp Exp
 | EInt Integer
```

Syntax-directed translation skeleton

```
transExp :: Exp -> Result
transExp x = case x of
  EAdd exp1 exp2  -> failure x
  ESub exp1 exp2  -> failure x
  EMul exp1 exp2  -> failure x
  EDiv exp1 exp2  -> failure x
  EInt n  -> failure x
```

Interpreter

```
eval :: Exp -> Integer
eval x = case x of
  EAdd exp1 exp2  -> eval exp1 + eval exp2
  ESub exp1 exp2  -> eval exp1 - eval exp2
  EMul exp1 exp2  -> eval exp1 * eval exp2
  EDiv exp1 exp2  -> eval exp1 `div` eval exp2
  EInt n  -> n
```

Abstract syntax in Java

```
public abstract class Exp implements java.io.Serializable {
  public abstract <R,A> R accept(Exp.Visitor<R,A> v, A arg);
  public interface Visitor <R,A> {
    public R visit(Calc.Absyn.EAdd p, A arg);
    // etc for ESub, EMul, ...
public class EAdd extends Exp {
  public final Exp exp_1, exp_2;
  public EAdd(Exp p1, Exp p2) { exp_1 = p1; exp_2 = p2; }

  public <R,A> R accept(Calc.Absyn.Exp.Visitor<R,A> v, A arg) {
    return v.visit(this, arg); }

// etc for ESub, EMul, ...
```

Java skeleton: visitor

```
public class VisitSkel
{
  public class ExpVisitor<R,A> implements Exp.Visitor<R,A>
  {
    public R visit(Calc.Absyn.EAdd p, A arg)
    {
      p.exp_1.accept(new ExpVisitor<R,A>(), arg);
      p.exp_2.accept(new ExpVisitor<R,A>(), arg);
      return null;
    }
    public R visit(Calc.Absyn.ESub p, A arg)
    {
```

Interpreter in Java

```
public class Interpreter {
  public Integer eval(Exp e) {
    return e.accept(new Value(), null ) ;
  }
  private class Value implements Exp. Visitor<Integer, Object> {
    public Integer visit (EAdd p, Object arg) {
      return eval(p.exp_1) + eval(p.exp_2) ;
    }
    public Integer visit (ESub p, Object arg) {
      return eval(p.exp_1) - eval(p.exp_2) ;
    }
```

Parsing

LALR(1) conversion: Happy, Bison, JavaCup

GLR (Tomita) available in Haskell

In principle, any BNF (context-free) method

Lexing

Finite automata: Alex, FLex, JLex

Predefined token types

```
  Integer Double Char String Ident
```

User-defined token types (regular expressions)

Rule format: basic labelled BNF

```
SWhile. Stm ::= "while" "(" Exp ")" Stm
```

generates

```
Stm = … | SWhile Exp Stm | …
```

Rule format: precedence numbers

```
EAdd. Exp0 ::= Exp0 "+" Exp1
EMul. Exp1 ::= Exp1 "*" Exp2
```

generates

```
Exp = … | EAdd Exp Exp | EMul Exp Exp | …
```

Rule format: precedence coercions

```
coercions Exp 2 ;
```

generates

```
_. Exp0 ::= Exp1
_. Exp1 ::= Exp2
_. Exp2 ::= "(" Exp1 ")"
```

Rule format: lists

```
terminator Stm ";"
```

generates

```
[].  [Stm] ::=
(:). [Stm] ::= Stm ";" [Stm]
```

Other rule formats

```
separator Exp ","

token UIdent (upper (letter | digit | ‘_’)*)

comment "/*" "*/"
```

Nothing much more

+ keep it simple
+ generate many host languages
+ encourage modular compiler design

- restricted to “well-behaved languages”

Use cases

Design of new languages

Multiple host languages (e.g. Haskell + C)

Teaching (at Chalmers 2003-, other places later)

Legacy languages? C, Java, SQL,...

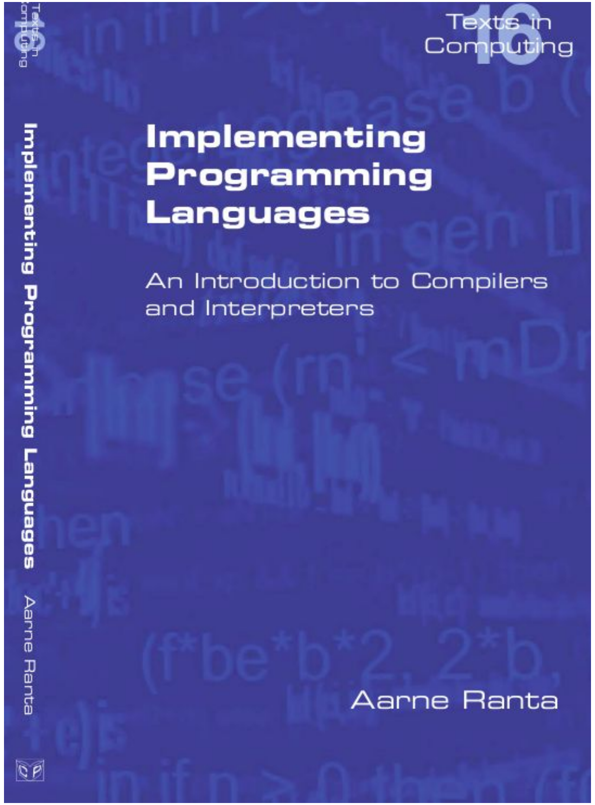
```
CQuery.           Command ::= Table ;

CInsert.          Command ::= "INSERT" "INTO" Ident VALUES ;

CUpdate.          Command ::= "UPDATE" Ident "SET" [Setting] WHERE ;

CDelete.          Command ::= "DELETE" STAR "FROM" Ident WHERE ;

CCreateDatabase.  Command ::= "CREATE" "DATABASE" Ident ;

CCreateTable.     Command ::= "CREATE" "TABLE" Ident "(" [Typing] ")" ;

CAlterTable.      Command ::= "ALTER" "TABLE" Ident Alter ;

CCreateView.      Command ::= "CREATE" "VIEW" Ident "AS" Table ;

CCreateAssertion. Command ::= "CREATE" "ASSERTION" Ident "CHECK" "(" Condition ")" ;

CDescribe.        Command ::= "DESCRIBE" Ident ;


QSelect.          Query   ::= "SELECT" TOP DISTINCT Columns "FROM" Table1 WHERE GROUP HAVING ORDER ;

QSelectWith.      Query   ::= "WITH" [Definition] Query ;


CCAll.   Columns ::= "*" ;

CCExps.  Columns ::= [Exp] ;


separator nonempty Ident "," ;   -- used in insert column names

separator nonempty Exp "," ;     -- used in insert values and in IN lists


WNone.      WHERE ::= ;

WCondition. WHERE ::= "WHERE" Condition ;


TName.         Table2 ::= Ident ;

TNameAlias.    Table2 ::= Table2 "AS" Ident ;

TNameAlias.    Table2 ::= Table2 Ident ;                  --- deprecated in standard SQL
```

```
TNameAlias.    Table2 ::= Table2 Ident ;                  --- deprecated in standard SQL

TProduct.      Table1 ::= Table1 "," Table2 ;

TUnion.        Table1 ::= Table1 "UNION" ALL Table2 ;

TIntersect.    Table1 ::= Table1 "INTERSECT" ALL Table2 ; --- ALL not in Oracle

TExcept.       Table1 ::= Table1 "EXCEPT" ALL Table2 ;    --- ALL not in Oracle

TJoin.         Table1 ::= Table1 "JOIN" Table2 ON ;

TNatJoin.      Table1 ::= Table1 "NATURAL" "JOIN" Table2 ;

TNatFullJoin.  Table1 ::= Table1 "NATURAL" "FULL" "OUTER" "JOIN" Table2 ;

TLeftJoin.     Table1 ::= Table1 "LEFT" "OUTER" "JOIN" Table2 ON ;

TRightJoin.    Table1 ::= Table1 "RIGHT" "OUTER" "JOIN" Table2 ON ;

TQuery.        Table  ::= Query ;


coercions Table 2 ;


EName.      Exp8 ::= Ident ;

EQual.      Exp8 ::= Ident "." Ident ;

ENameAlias. Exp8 ::= Exp8 "AS" Ident ;

EQuery.     Exp8 ::= "(" Query ")" ;

EInt.       Exp8 ::= Integer ;

EFloat.     Exp8 ::= Double ;

EStr.       Exp8 ::= Str ;      -- single quotes

EString.    Exp8 ::= String ;   -- double quotes

ENull.      Exp8 ::= "NULL" ;

EList.      Exp8 ::= "(" Exp "," [Exp] ")" ; --- City IN ('Paris','Berlin')

EAggr.      Exp8 ::= AggrOper "(" DISTINCT Exp ")" ;

EAggrAll.   Exp8 ::= AggrOper "(" DISTINCT "*" ")" ;

EDef.       Exp8 ::= "DEFAULT" ;
```

Resources

http://bnfc.digitalgrammars.com/

https://github.com/BNFC/bnfc


Texts in Computing
16
Implementing Programming Languages
An Introduction to Compilers and Interpreters
Aarne Ranta

Aarne Ranta and Markus Forsberg, *Implementing Programming Languages. An Introduction to Compilers and Interpreters*, College Publications, London, 2012.

BNFC is no rocket science

- just a piece of useful engineering.

GF

How to link two grammars?

```
EAdd. Exp  ::= Exp  “+” Exp1
EMul. Exp1 ::= Exp1 “*” Exp2
EInt. Exp2 ::= Integer
```

```
EAdd. Exp ::= Exp Exp “iadd”
EMul. Exp ::= Exp Exp “imul”
EInt. Exp ::= “ldc” Integer
```

Common abstract syntax

```
EAdd. Exp  ::= Exp  “+” Exp1
EMul. Exp1 ::= Exp1 “*” Exp2
EInt. Exp2 ::= Integer
```

```
EAdd. Exp ::= Exp Exp “iadd”
EMul. Exp ::= Exp Exp “imul”
EInt. Exp ::= “ldc” Integer
```

```
EAdd. Exp ::= Exp Exp
EMul. Exp ::= Exp Exp
EInt. Exp ::= Integer
```

Let us define them separately

```
concrete CalcJava of Calc
lin EAdd a b = a ++ “+” ++ b
lin EMul a b = a ++ “*” ++ b
lin EInt n = n
```

```
concrete CalcJVM of Calc
lin EAdd a b = a ++ b ++ “iadd”
lin EMul a b = a ++ b ++ “imul”
lin EInt n = “ldc” ++ n
```

```
abstract Calc
fun EAdd : Exp -> Exp -> Exp
fun EMul : Exp -> Exp -> Exp
fun EInt : Integer -> Exp
```

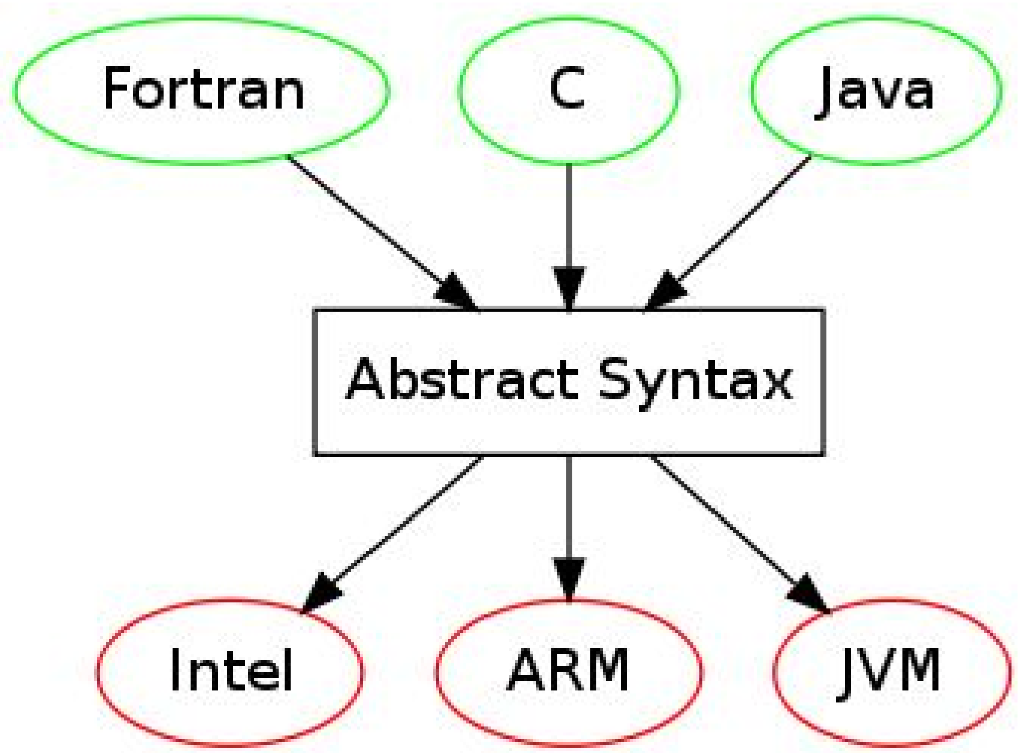
GF = Grammatical Framework

multilingual grammar =

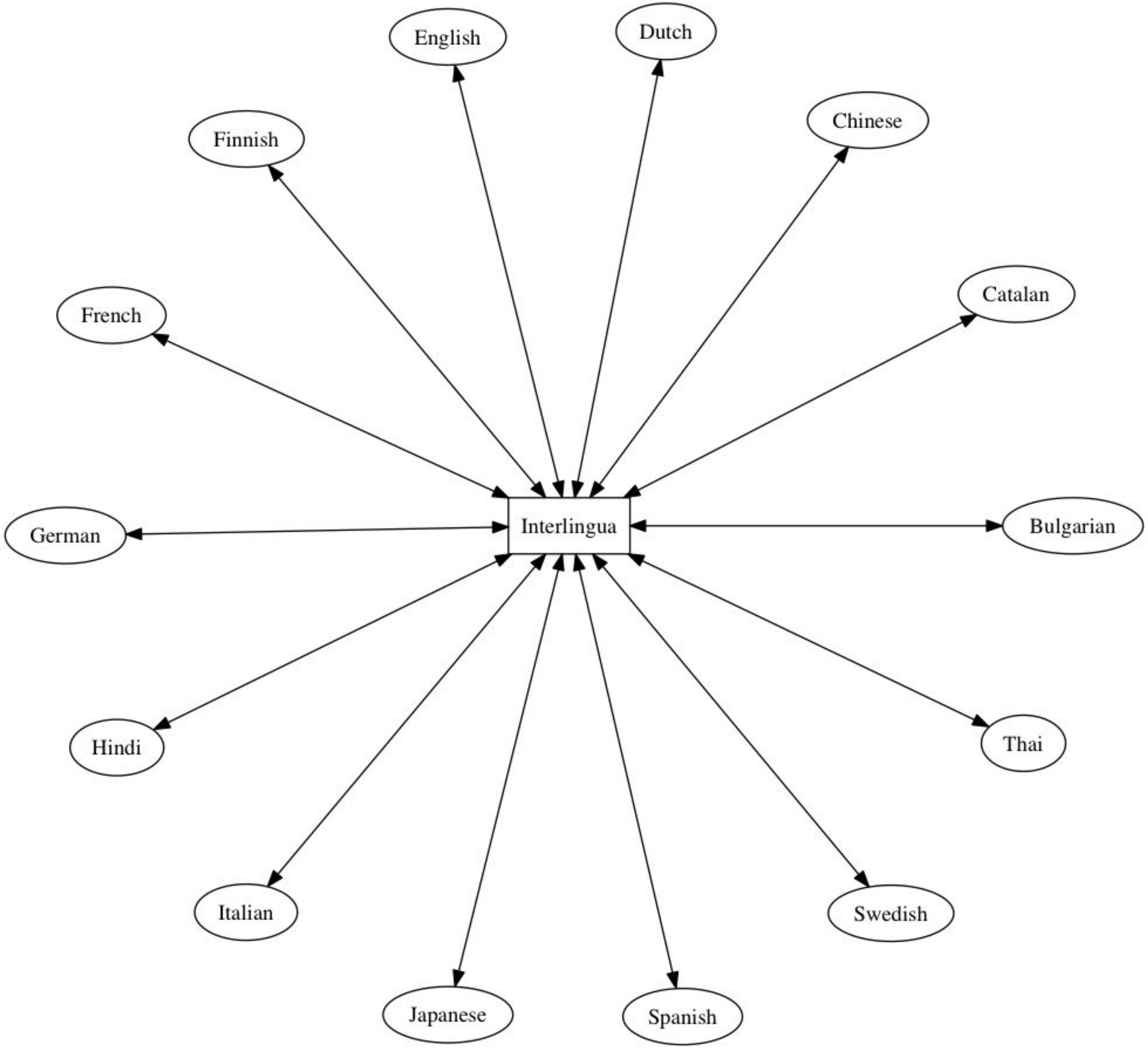
abstract syntax + concrete syntaxes

GF = Logical Framework + concrete syntax

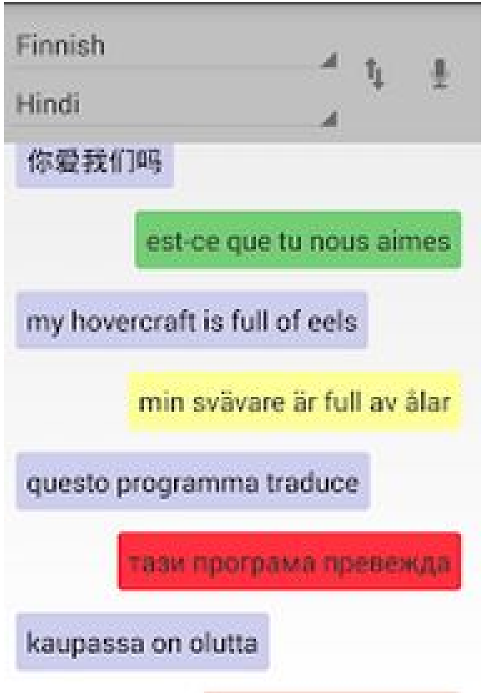
Translation model: multi-source multi-target compiler



Compiling natural language



Demo: GF Offline Translation



你愛我们吗

est-ce que tu nous aimes

my hovercraft is full of eels

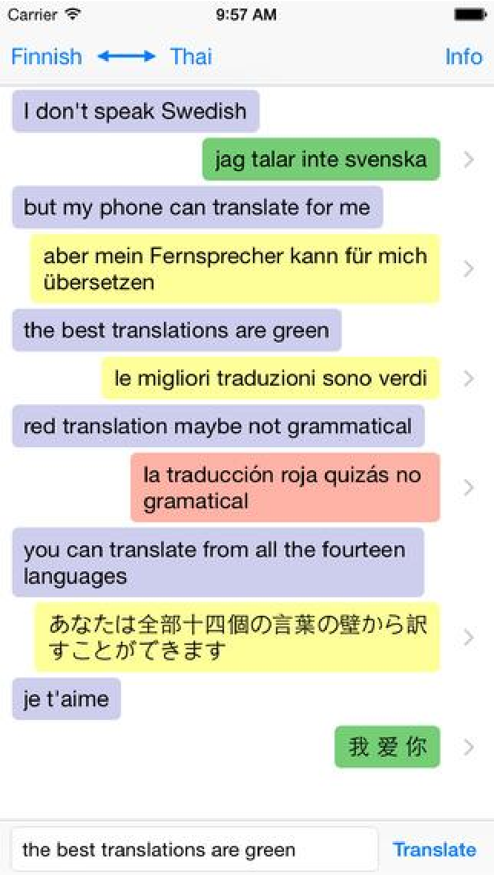
min svävare är full av ålar

questo programma traduce

тази програма превежда

kaupassa on olutta

https://play.google.com/store/apps/details?id=org.grammaticalframework.ui.android



I don't speak Swedish

jag talar inte svenska

but my phone can translate for me

aber mein Fernsprecher kann für mich übersetzen

the best translations are green

le migliori traduzioni sono verdi

red translation maybe not grammatical

la traducción roja quizás no gramatical

you can translate from all the fourteen languages

あなたは全部十四個の言葉の壁から訳すことができます

je t'aime

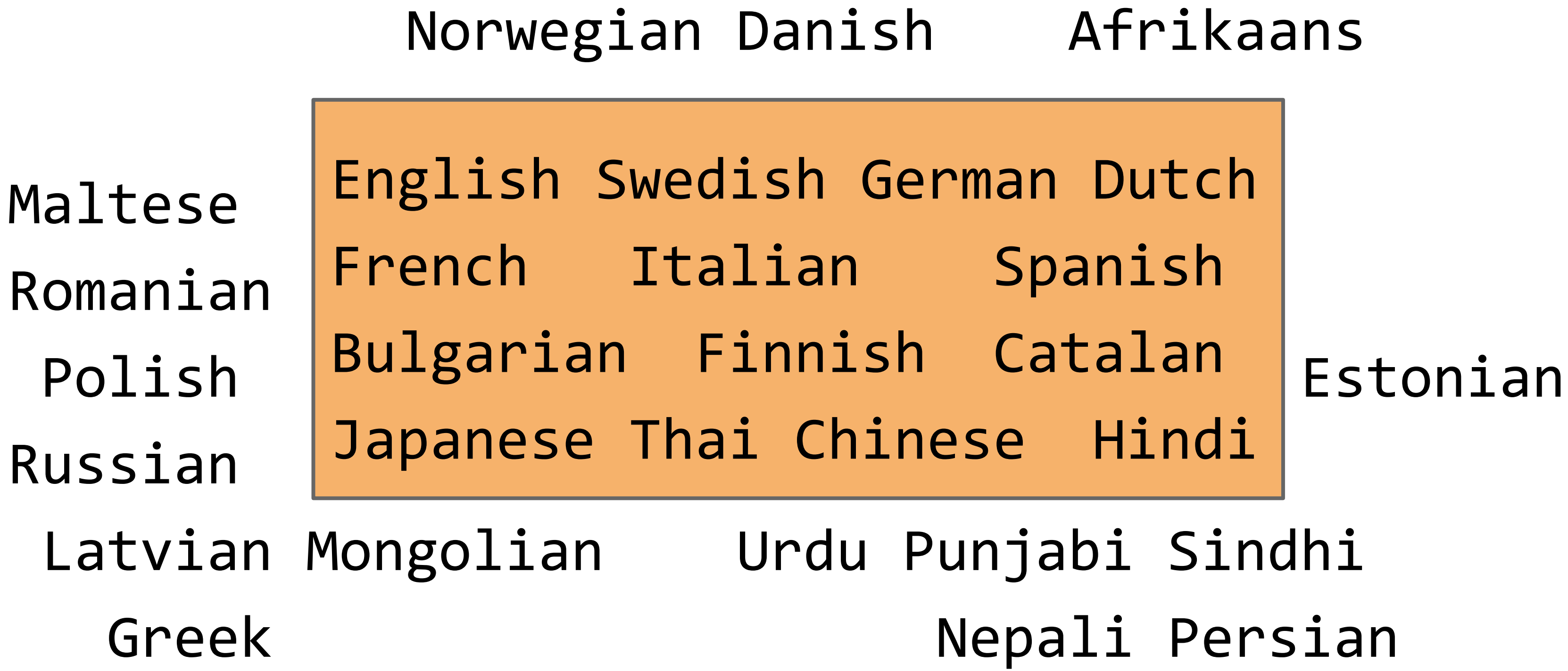
我 爱 你

the best translations are green

https://itunes.apple.com/us/app/gf-offline-translator/id1023328422?mt=8

K. Angelov, B. Bringert & A. Ranta, Speech-enabled hybrid multilingual translation for mobile devices, EACL 2014.

GF resources: 30 languages



Norwegian Danish Afrikaans

English Swedish German Dutch

French Italian Spanish

Bulgarian Finnish Catalan

Japanese Thai Chinese Hindi

Maltese

Romanian

Polish

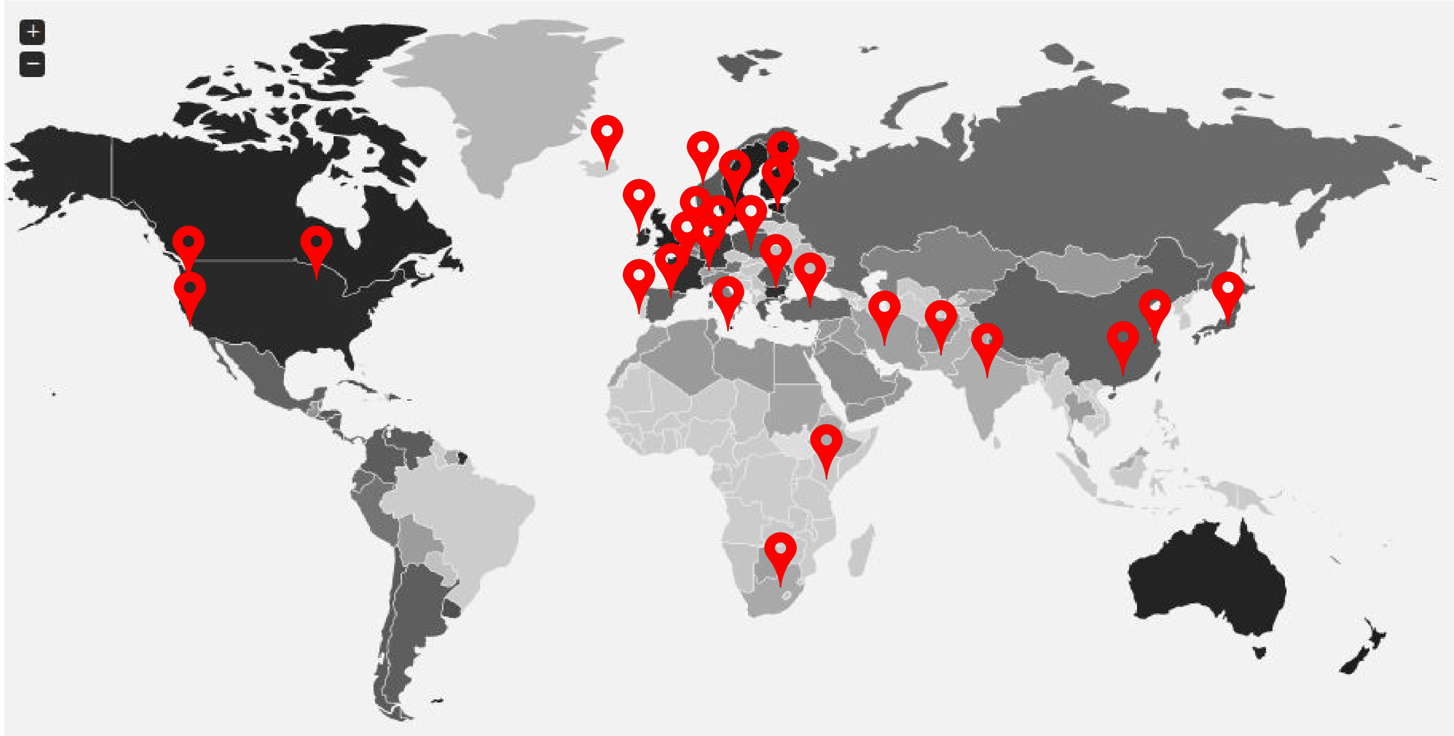
Russian

Estonian

Latvian Mongolian Urdu Punjabi Sindhi

Greek Nepali Persian

GF community: 150+ members



How it works: an example

you have 1 new messages

How it works: an example

you have 1 new messages

you have 1 new message(s)

How it works: an example

you have 1 new messages

you have 1 new message(s)

you have 1 new message
you have 2 new messages

```
abstract Mail = {

cat
  Welcome ;
  Number ;

fun
  YouHave : Number -> Welcome ;
  One, Two : Number ;

}
```

```
abstract Mail = {

cat
  Welcome ;
  Number ;

fun
  YouHave : Number -> Welcome ;
  One, Two : Number ;

}
```

```
concrete MailEng of Mail = {

lincat
  Welcome = Str ;
  Number = {s : Str ; n : Num} ;

lin
  YouHave k =
    “you have” ++ k.s ++ “new” ++
      case k.n of {
        Sg => “message” ;
        Pl => “messages”
        } ;
  One = {s = “1” ; n = Sg} ;
  Two = {s = “2” ; n = Pl} ;

param Num = Sg | Pl ;
}
```

This was just the beginning

sinulla on 1 uusi viesti

sinulla on 2 uutta viestiä

1 message	رِسَالَةٌ	*risālatun*
2 messages	رِسَالَتَانِ	*risālatāni*
(3–10) messages	رَسَائِلَ	*rasāʾila*
(11–99) messages	رِسَالَةً	*risālatan*
x100 messages	رِسَالَةٍ	*risālatin*

Library-based solution

```
-- RGL API (Resource Grammar Library)

oper mkNP : Numeral -> N -> NP
oper mkCl : NP -> V2 -> NP -> Cl
```

Library-based solution

```
-- RGL API (Resource Grammar Library)
oper mkNP : Numeral -> N -> NP
oper mkCl : NP -> V2 -> NP -> Cl

-- application grammar

lin YouHave n =
  mkCl you_NP have_V2
    (mkNP n (mkCN new_A message_N))
```

Library-based solution

```
-- RGL API (Resource Grammar Library)
oper mkNP : Numeral -> N -> NP
oper mkCl : NP -> V2 -> NP -> Cl

-- application grammar

lin YouHave n =
  mkCl you_NP have_V2
    (mkNP n (mkCN uusi_A viesti_N))
```

Library-based solution

```
-- RGL API (Resource Grammar Library)
oper mkNP : Numeral -> N -> NP
oper mkCl : NP -> V2 -> NP -> Cl

-- application grammar

lin YouHave n =
  mkCl you_NP have_V2
    (mkNP n (mkCN jadid_A risala_N))
```

Expressivity of GF

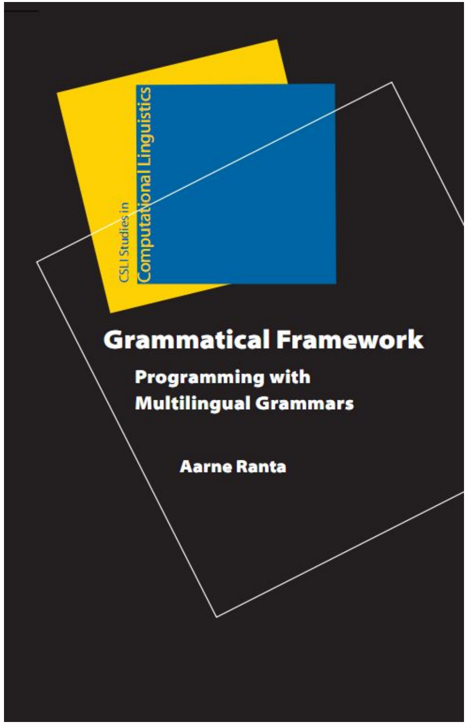
GF source language

- type theory + functional programming
- static type checking
- module system (inheritance, functors)

PGF machine language

- Portable Grammar Format (binary)
- PMCFG (Parallel Multiple Context-Free Grammar)

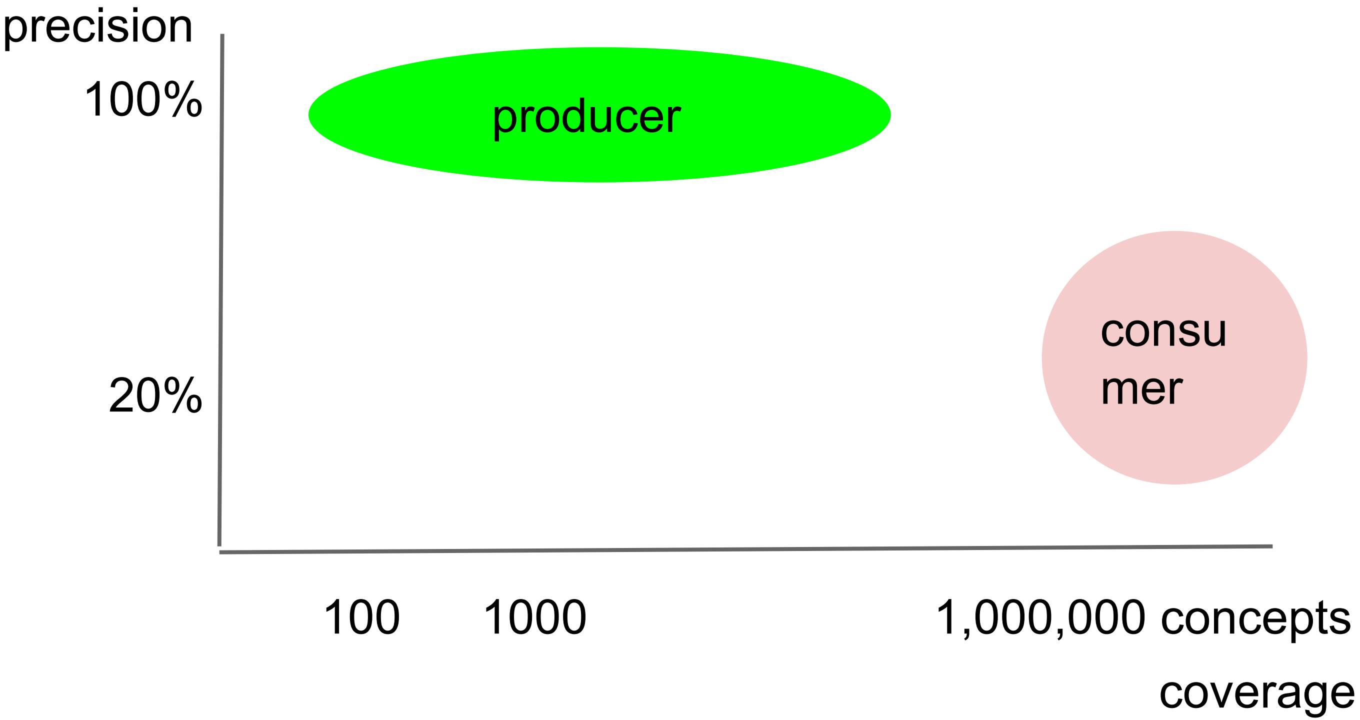
The GF book



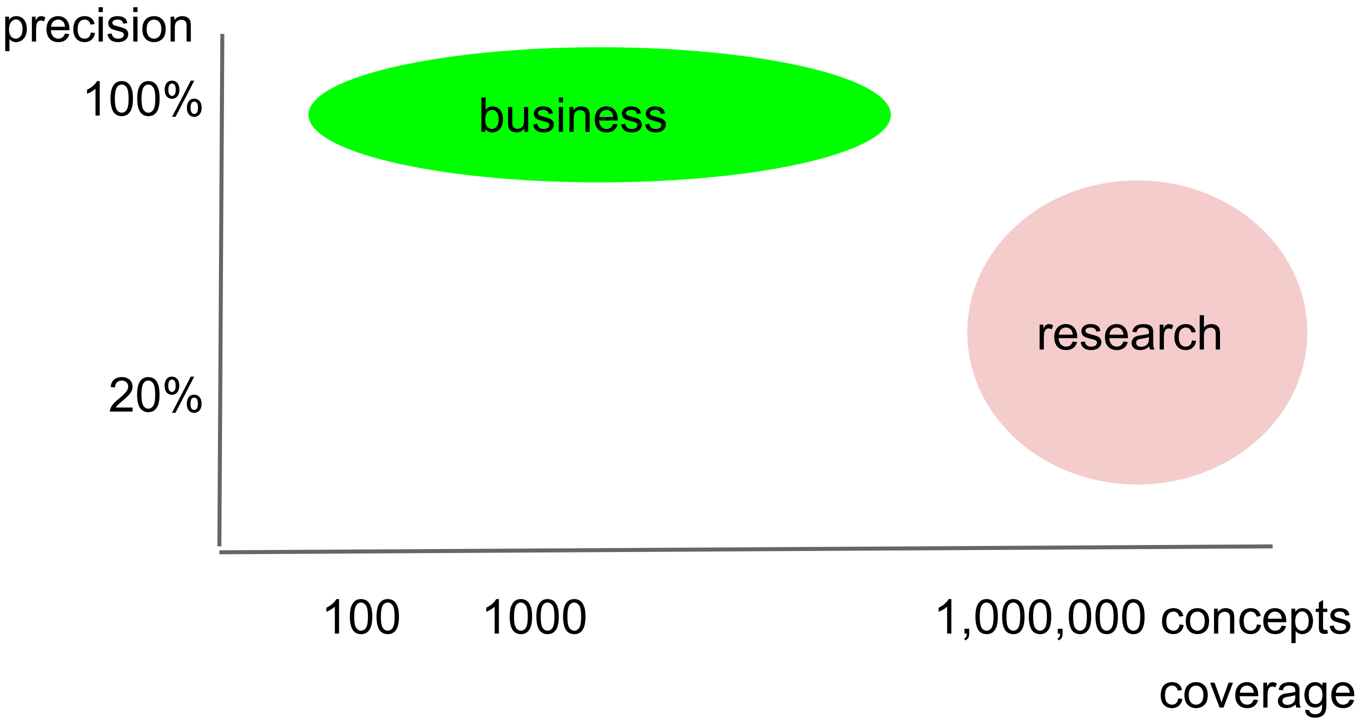
A. Ranta. *Grammatical Framework: Programming with Multilingual Grammars*, CSLI, Stanford, 2011. Chinese translation by Prof. Yan Tian: 语法框架 为多种自然语言语法编程, Shanghai Jiao Tong University Press, 2014.

GF Applications and Business

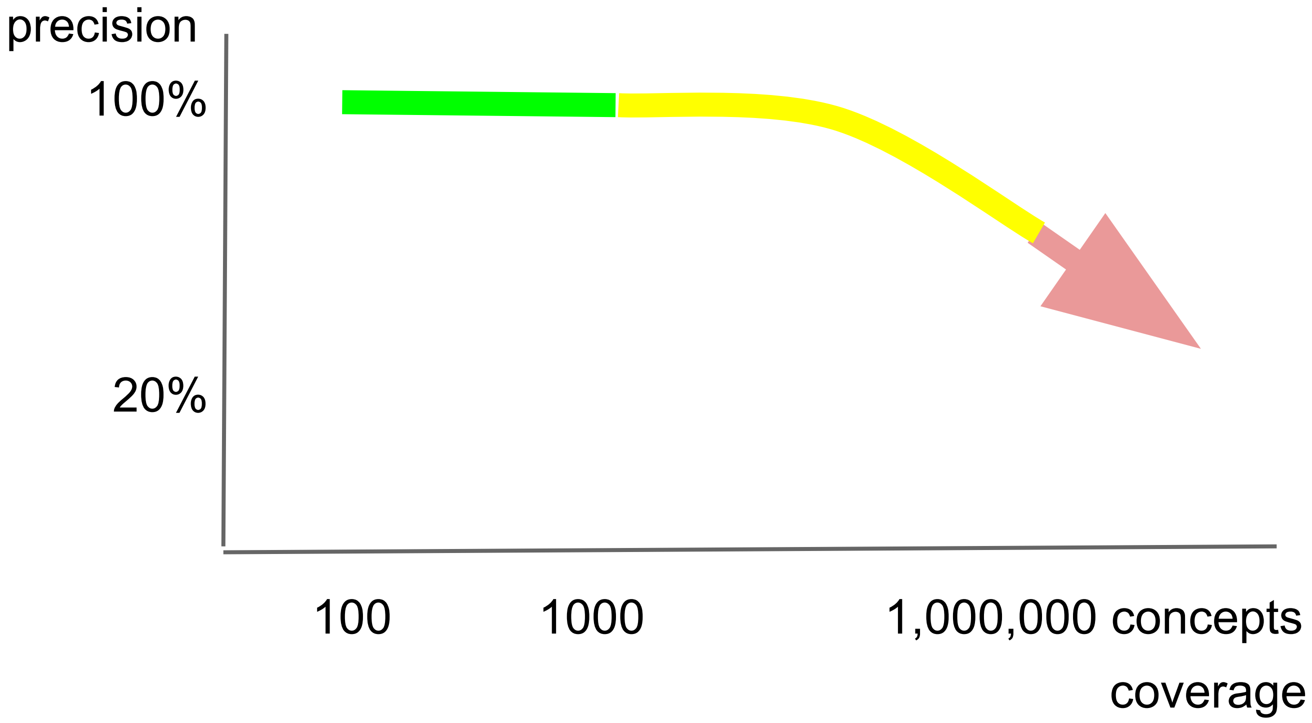
Producer vs. consumer translation



Maturity level



Graceful degradation



An example

I am hungry.

Minulla on nälkä.

meaning

The vice dean kicked the bucket.

Pahedekaani potkaisi ämpäriä.

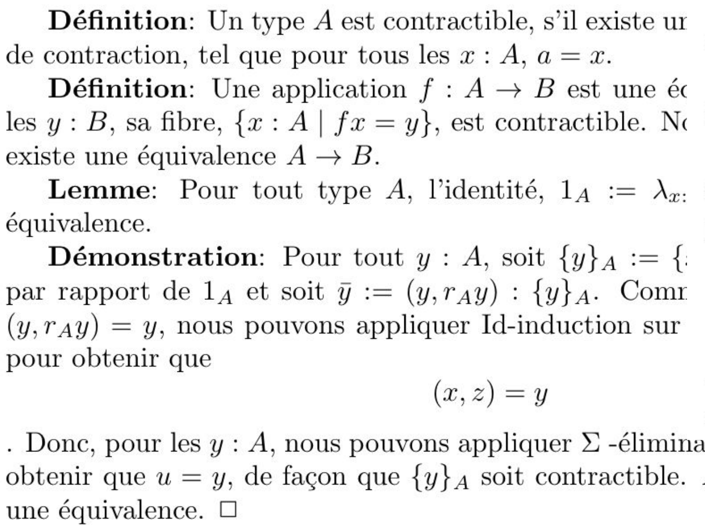
syntax

Little boy eat big snake.

Pieni poika syödä iso käärme.

chunks

```
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)) (Ap
BaseInd) (ExpProp (equalExp (VarExp u_Var) (VarExp y_Var))))) (PredProp contractible_Pre
ConclusionParagraph (PropConclusion (PredProp equivalence_Pred (FunInd (ExpFun (Type
QEDParagraph
```



Définition: Un type A est contractible, s'il existe ur
de contraction, tel que pour tous les $x : A$, $a = x$.

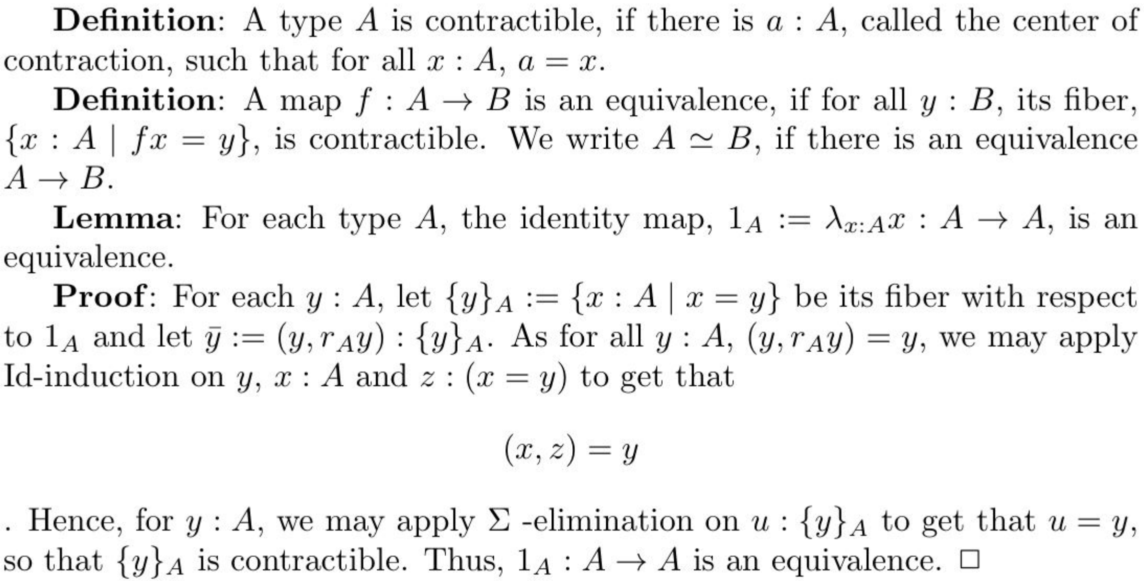
Définition: Une application $f : A \to B$ est une éc
les $y : B$, sa fibre, $\{x : A \mid fx = y\}$, est contractible. Nc
existe une équivalence $A \to B$.

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

Démonstration: Pour tout $y : A$, soit $\{y\}_A := \{$
par rapport de 1_A et soit $\bar{y} := (y, r_A y) : \{y\}_A$. Comr
$(y, r_A y) = y$, nous pouvons appliquer Id-induction sur
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 contractible. Alors, $1_A : A \to A$ est une équivalence. □



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

$$(x, z) = y$$

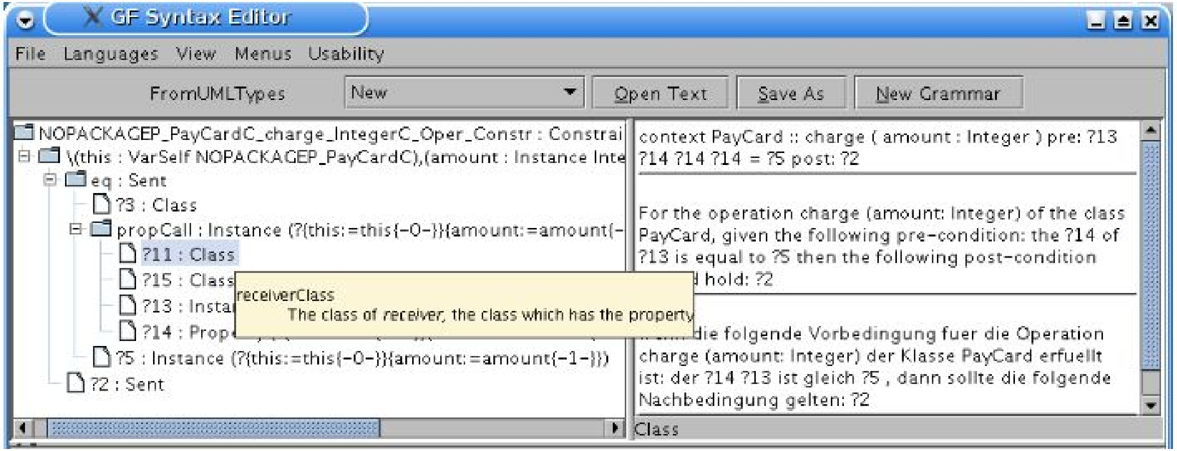
. Hence, for $y : A$, we may apply Σ -elimination on $u : \{y\}_A$ to get that $u = y$, so that $\{y\}_A$ is contractible. Thus, $1_A : A \to A$ is an equivalence. □

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

GF-KeY

- if the try counter is equal to 0 then this implies that the result is equal to false
- if the following conditions are true
  - the try counter is greater than 0
  - *pin* is not equal to null
  - *offset* is at least 0
  - *length* is at least 0
  - *offset* plus *length* is at most the size of *pin*
  - the query `arrayCompare ( the pin , 0 , ` *`pin`* ` , ` *`offset`* ` , ` *`length`* ` )`[1] `on Util is` equal to 0

  then this implies that the following conditions are true
  - the result is equal to true
  - this owner PIN is validated
  - the try counter is equal to the maxin



```
NOPACKAGEP_PayCardC_charge_IntegerC_Oper_Constr : Constrai
  \(this : VarSelf NOPACKAGEP_PayCardC),(amount : Instance Inte
    eq : Sent
      ?3 : Class
      propCall : Instance (?{this:=this{-0-}}{amount:=amount{-
        ?11 : Class
        ?15 : Class
        ?13 : Insta
        ?14 : Prop
      ?5 : Instance (?{this:=this{-0-}}{amount:=amount{-1-}})
    ?2 : Sent
```

receiverClass
The class of *receiver*, the class which has the property

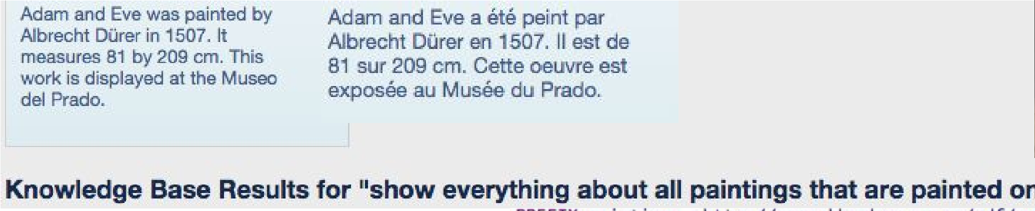
context PayCard :: charge (amount : Integer) pre: ?13 ?14 ?14 ?14 = ?5 post: ?2

For the operation charge (amount: Integer) of the class PayCard, given the following pre-condition: the ?14 of ?13 is equal to ?5 then the following post-condition hold: ?2

die folgende Vorbedingung fuer die Operation charge (amount: Integer) der Klasse PayCard erfuellt ist: der ?14 ?13 ist gleich ?5 , dann sollte die folgende Nachbedingung gelten: ?2

K. Johannisson, Formal and Informal Software Specifications, PhD Thesis, 2005

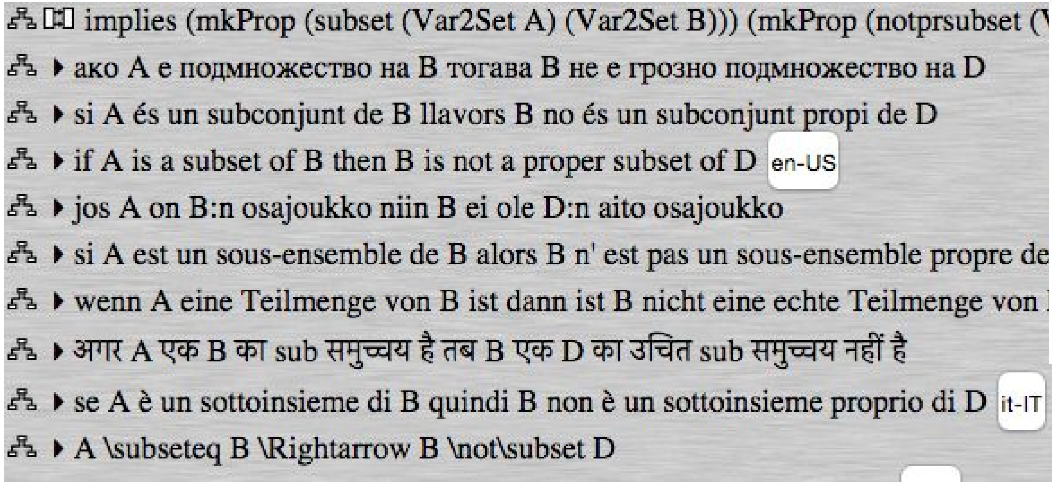
2010-2013: MOLTO



Adam and Eve was painted by Albrecht Dürer in 1507. It measures 81 by 209 cm. This work is displayed at the Museo del Prado.

Adam and Eve a été peint par Albrecht Dürer en 1507. Il est de 81 sur 209 cm. Cette oeuvre est exposée au Musée du Prado.

Knowledge Base Results for "show everything about all paintings that are painted on canvas" (100 of many)



implies (mkProp (subset (Var2Set A) (Var2Set B))) (mkProp (notprsubset (V

ако A е подмножество на B тогава B не е грозно подмножество на D

si A és un subconjunt de B llavors B no és un subconjunt propi de D

if A is a subset of B then B is not a proper subset of D en-US

jos A on B:n osajoukko niin B ei ole D:n aito osajoukko

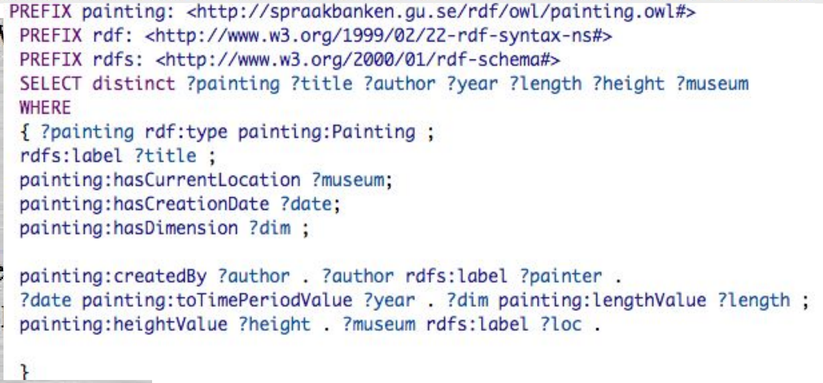
si A est un sous-ensemble de B alors B n' est pas un sous-ensemble propre de

wenn A eine Teilmenge von B ist dann ist B nicht eine echte Teilmenge von

अगर A एक B का sub समुच्चय है तब B एक D का उचित sub समुच्चय नहीं है

se A è un sottoinsieme di B quindi B non è un sottoinsieme proprio di D it-IT it-IT

A \subseteq B \Rightarrow B \not\subset D



```
PREFIX painting: <http://spraakbanken.gu.se/rdf/owl/painting.owl#>
 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
 SELECT distinct ?painting ?title ?author ?year ?length ?height ?museum
 WHERE
 { ?painting rdf:type painting:Painting ;
 rdfs:label ?title ;
 painting:hasCurrentLocation ?museum;
 painting:hasCreationDate ?date;
 painting:hasDimension ?dim ;

 painting:createdBy ?author . ?author rdfs:label ?painter .
 ?date painting:toTimePeriodValue ?year . ?dim painting:lengthValue ?length ;
 painting:heightValue ?height . ?museum rdfs:label ?loc .

 }
```

MOLTO Multilingual Online Translation

Non multa, sed multum not quantity but quality

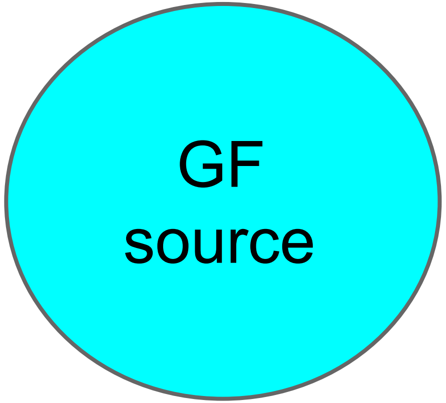
FP7-ICT-247914

- Svängrumsytan utanför dörren lutar 2% i sidled.
- The turning space outside the gate tilts 2% sideways.
- Kääntymätila oven ulkopuolella kallistuu 2% sivusuunnassa.

UttSTD (PredUttTD (AdvNPTD (DetCNTD (DetQuant DefArt NumSg) (UseNTD svängrumsyta_NTD)) (PrepNPTD utanför_Prep (DetCNTD (DetQuant DefArt NumSg) (UseNTD dörr_NTD)))) (AdvVPTD (luta_VPTD (ProcentMeasure 2)) i_sidled_AdvTD))

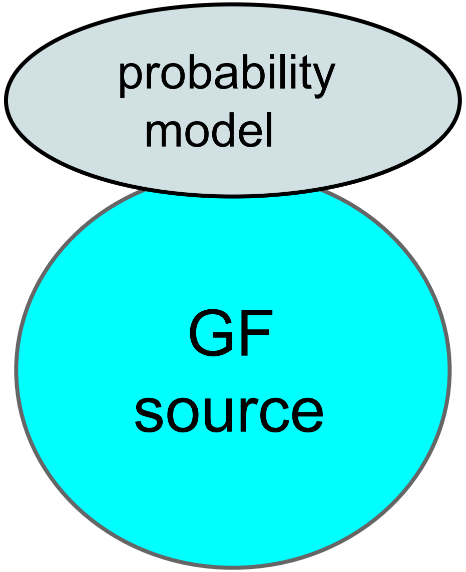
http://www.t-d.se/sv/TD2/

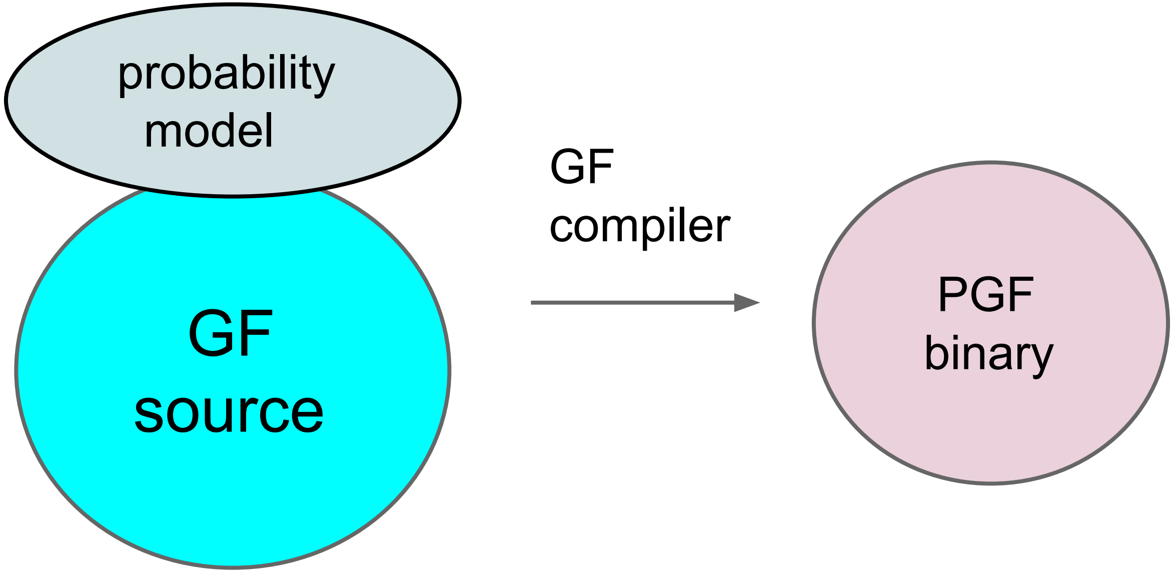
Building GF applications

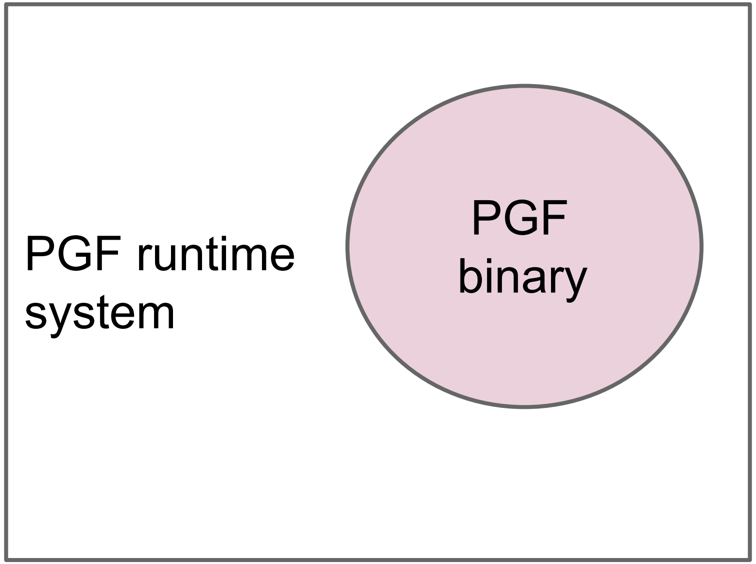


GF

source

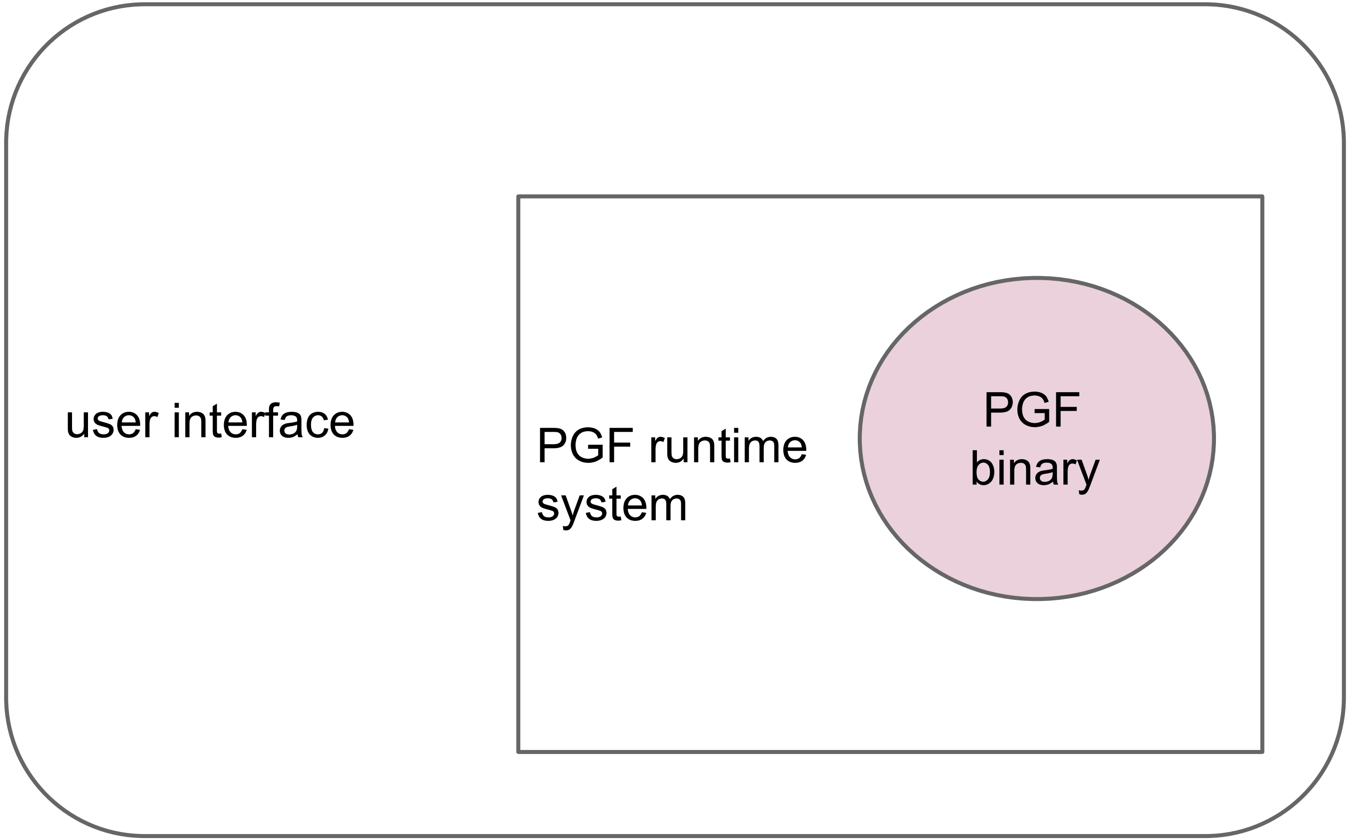

probability
model
GF
source


probability model
GF source
GF compiler
PGF binary



PGF runtime system

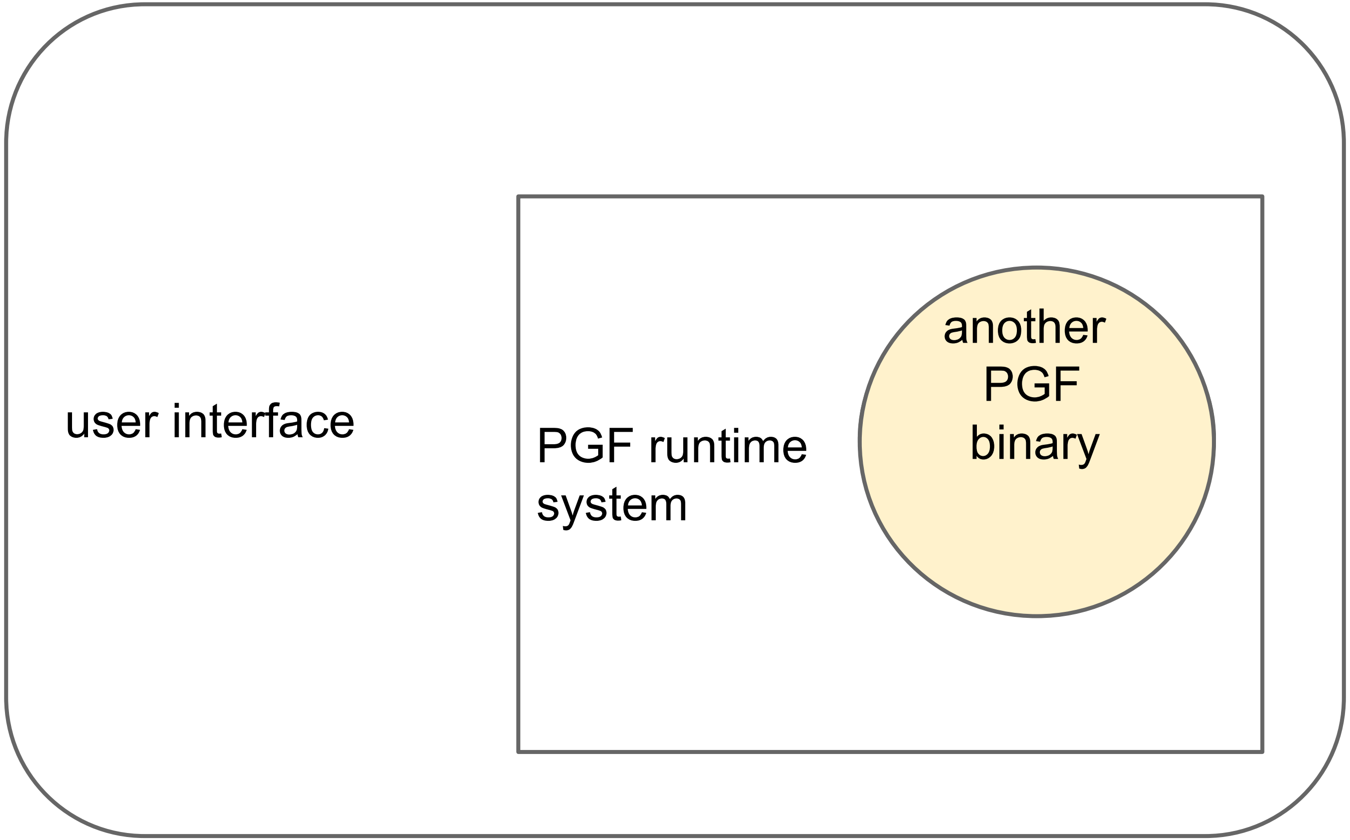
PGF binary



user interface

PGF runtime system

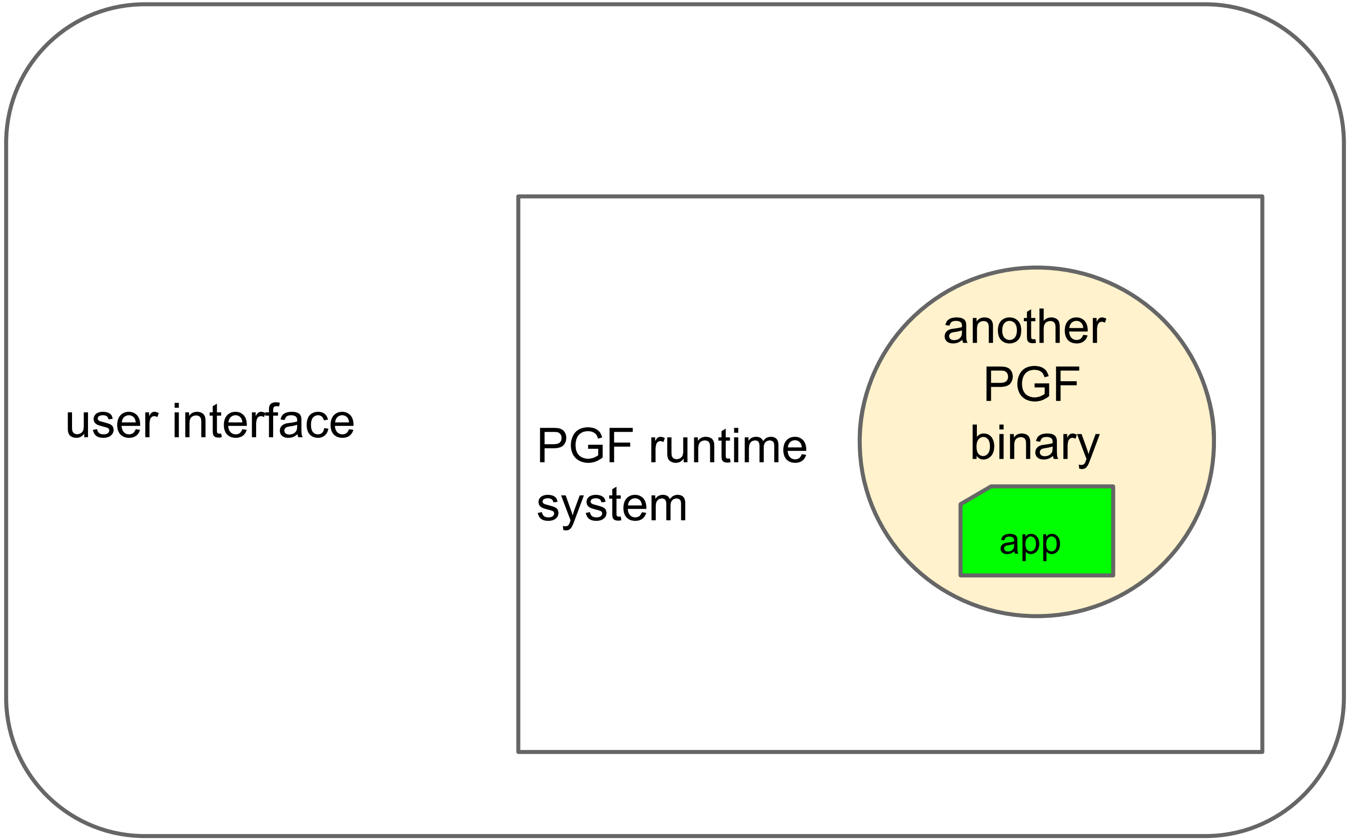
PGF binary

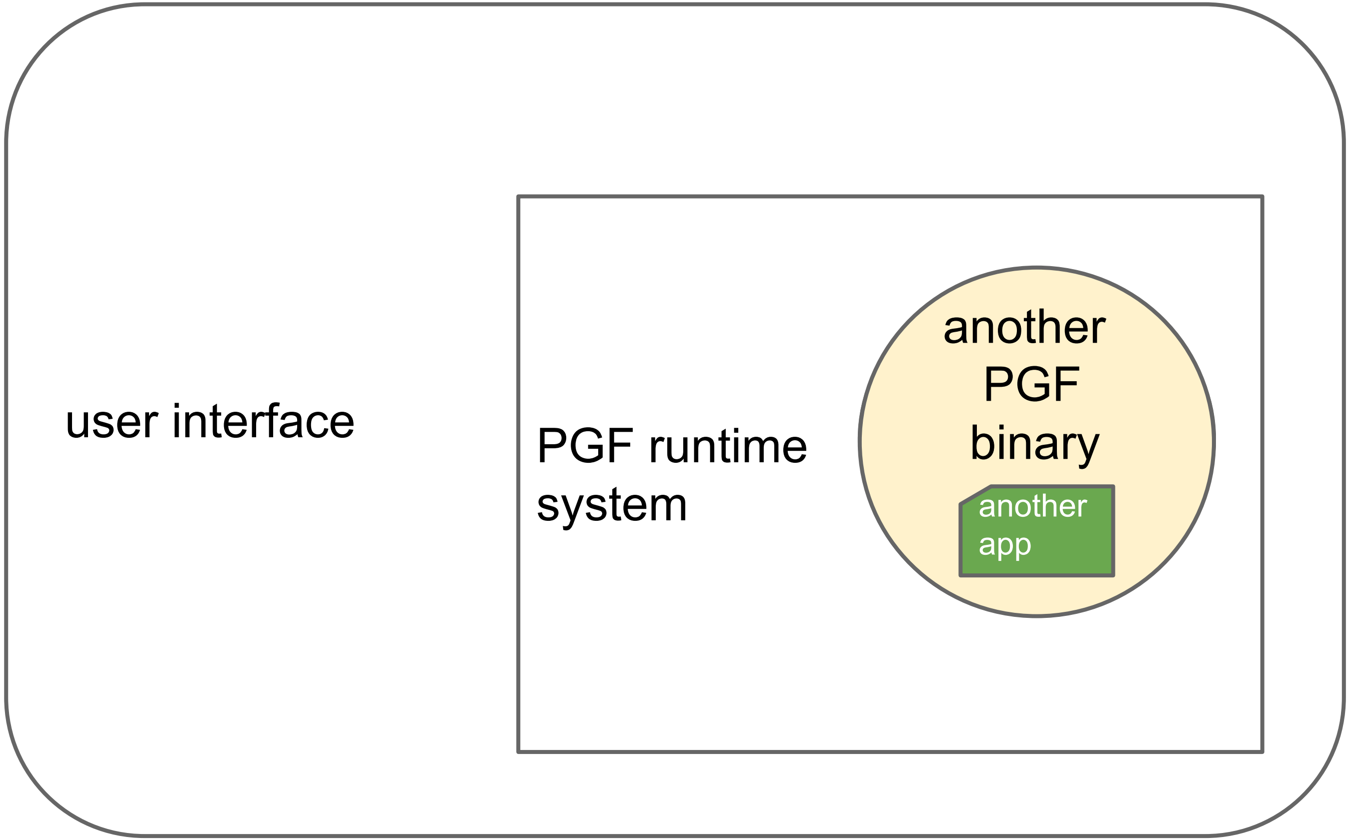


user interface

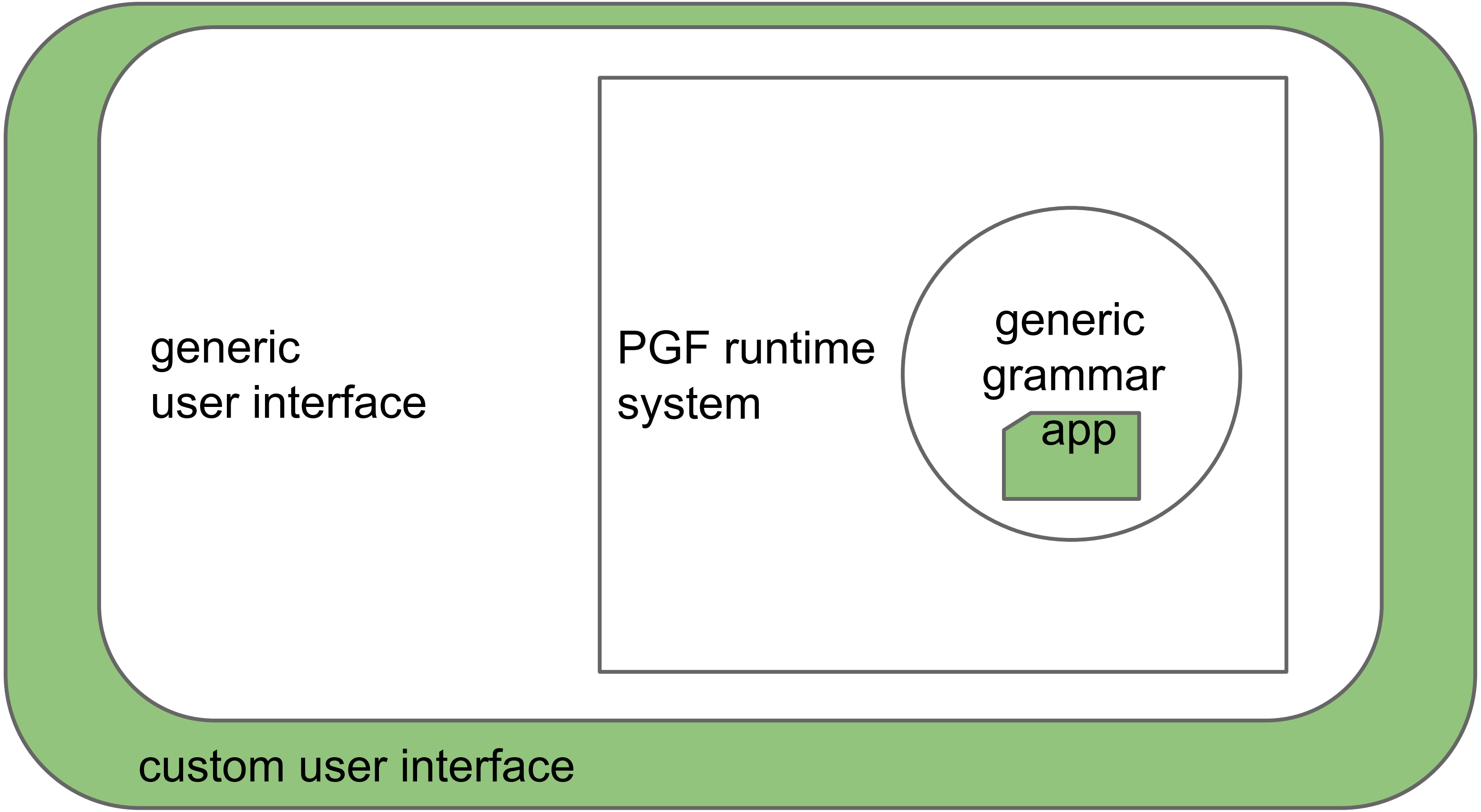
PGF runtime system

another PGF binary


user interface
PGF runtime system
another PGF binary
app


user interface
PGF runtime system
another PGF binary
another app

White: free, open-source (LGPL/BSD). **Green**: a business idea



App size

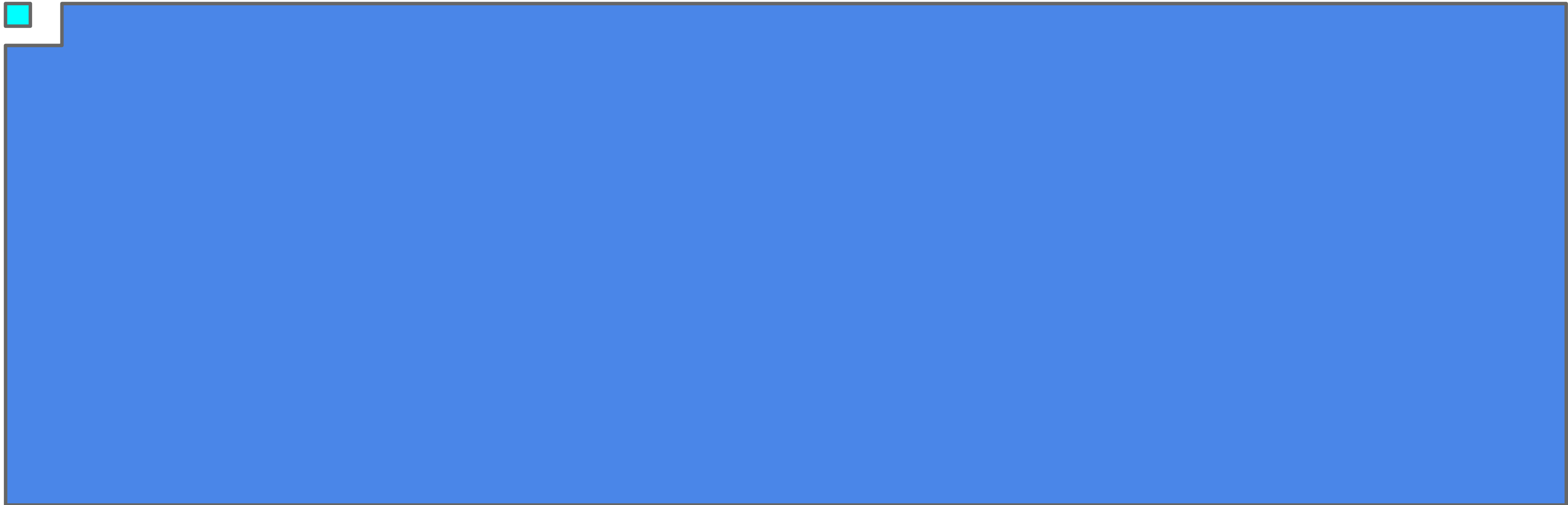
For 14 languages

- 15 modules total 30 MB

App size

For 14 languages

- 15 modules, 30 MB in total
- Google translate offline: 182 modules, 150 MB each



Take home

Grammars:

- declarative models of languages
- useful in engineering

BNFC: programming language grammars

GF: natural language grammars

Hands on: let us build a grammar

http://cloud.grammaticalframework.org/gfse/

cloud-based grammar editor

Some useful Finnish phrases perhaps?