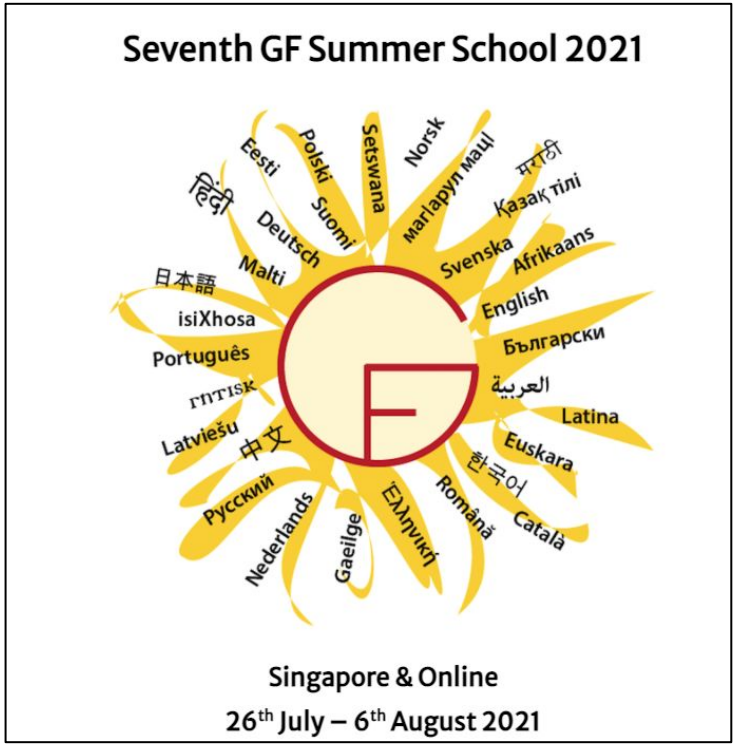
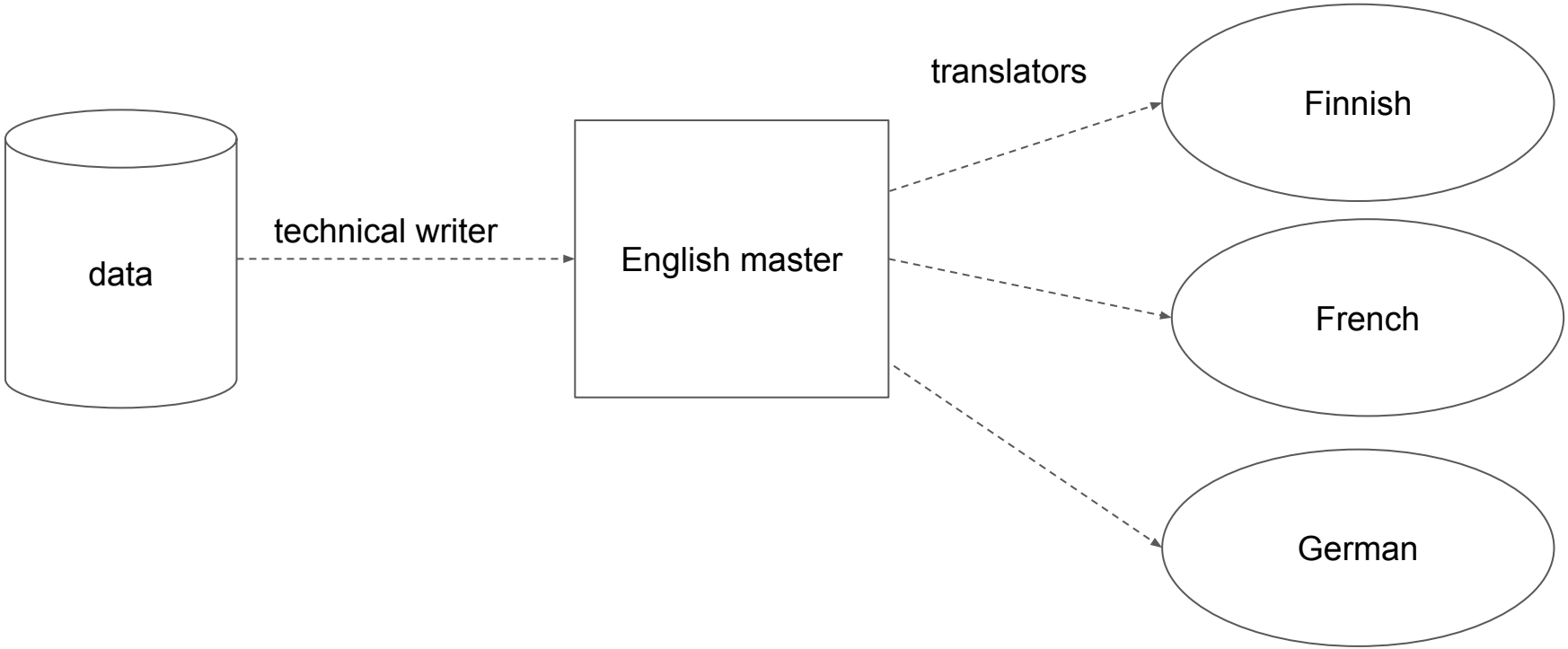
GF for NLG

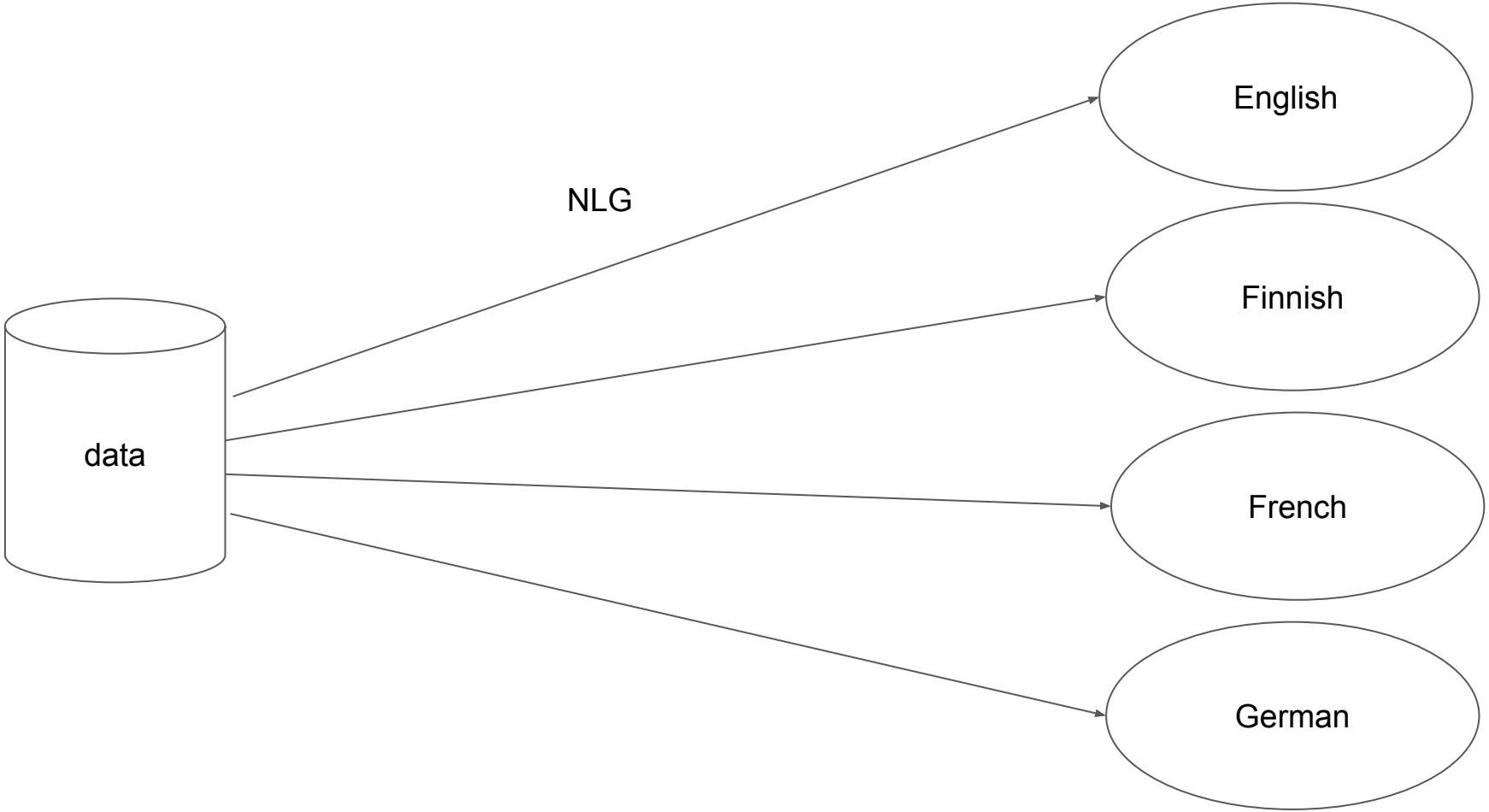
A tutorial on multilingual text generation from (Wiki)data

Aarne Ranta

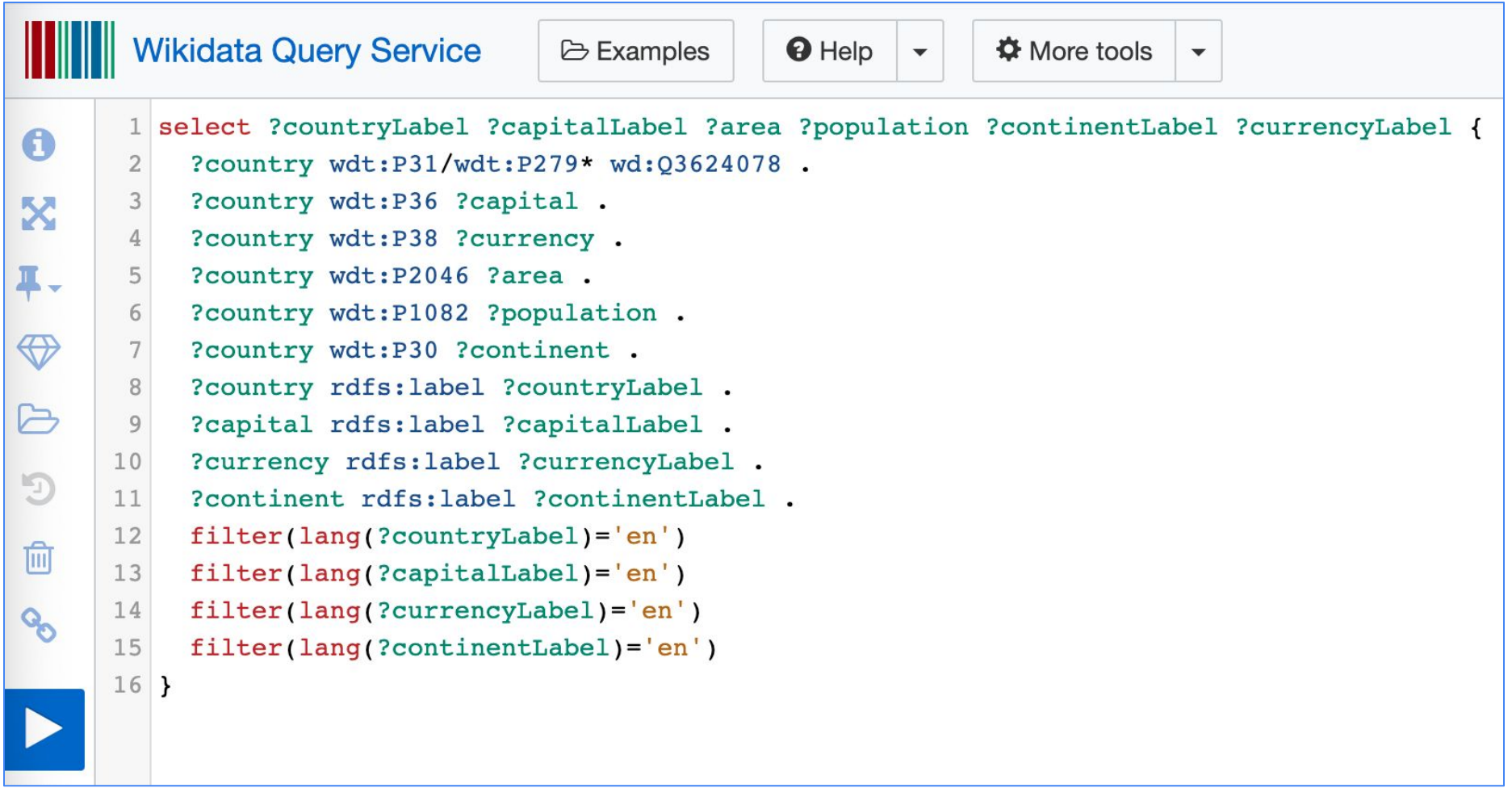


From translation to NLG


data
technical writer
English master
translators
Finnish
French
German


English
NLG
Finnish
data
French
German

Starting point: Wikidata



https://w.wiki/3fV$

country	capital	area	population	continent	currency
Afghanistan	Kabul	652230	36643815	Asia	Afghan afghani
Albania	Tirana	28748	3020209	Europe	Albanian lek
Algeria	Algiers	2381741	41318142	Africa	Algerian dinar
Andorra	Andorra la Vella	468	76177	Europe	euro
Angola	Luanda	1246700	29784193	Africa	kwanza
Argentina	Buenos Aires	2780400	44938712	South America	Argentine peso

Our plan

Bottom-up development

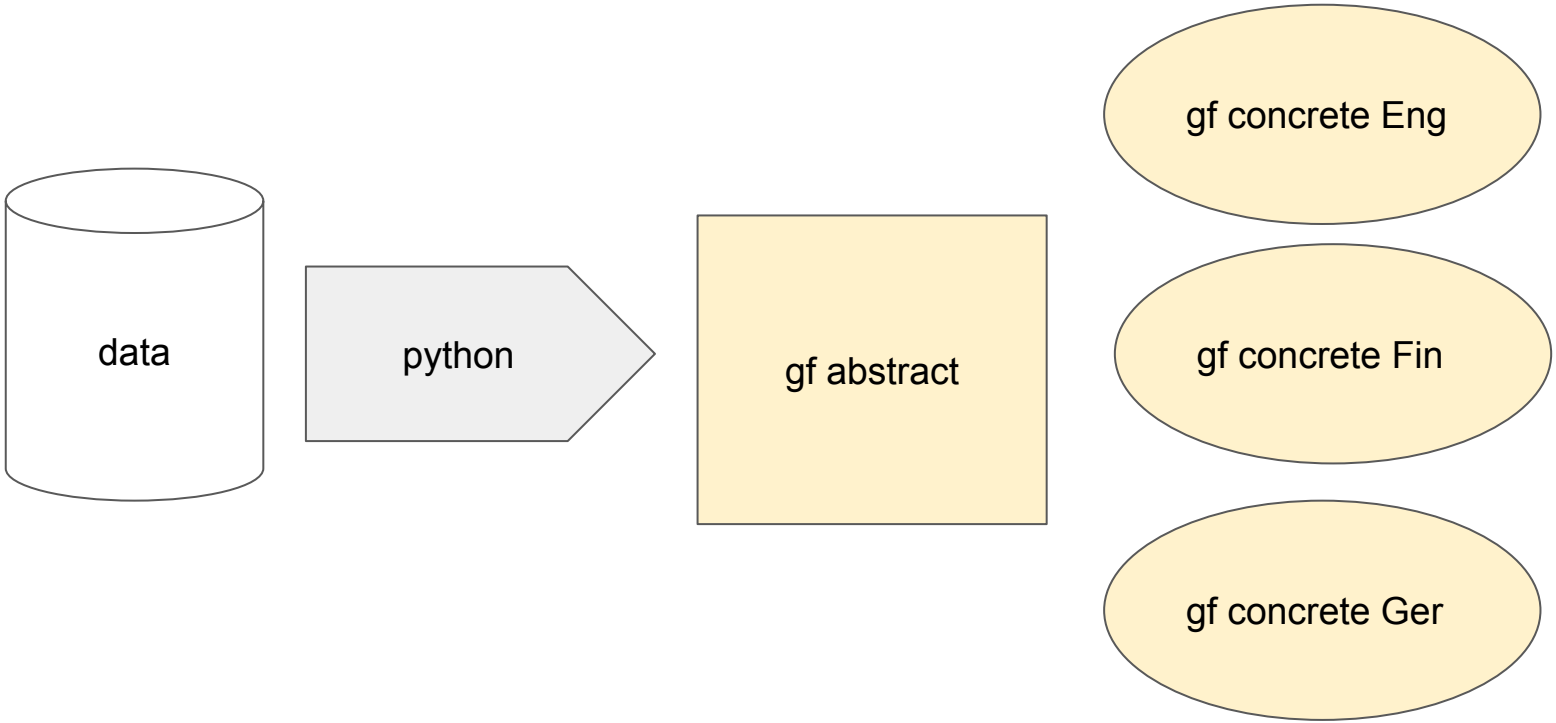
Stage 1: templates for atomic country facts

Stage 2: grammars for atomic facts

Stage 3: text planning

Stage 4: content planning

System architecture



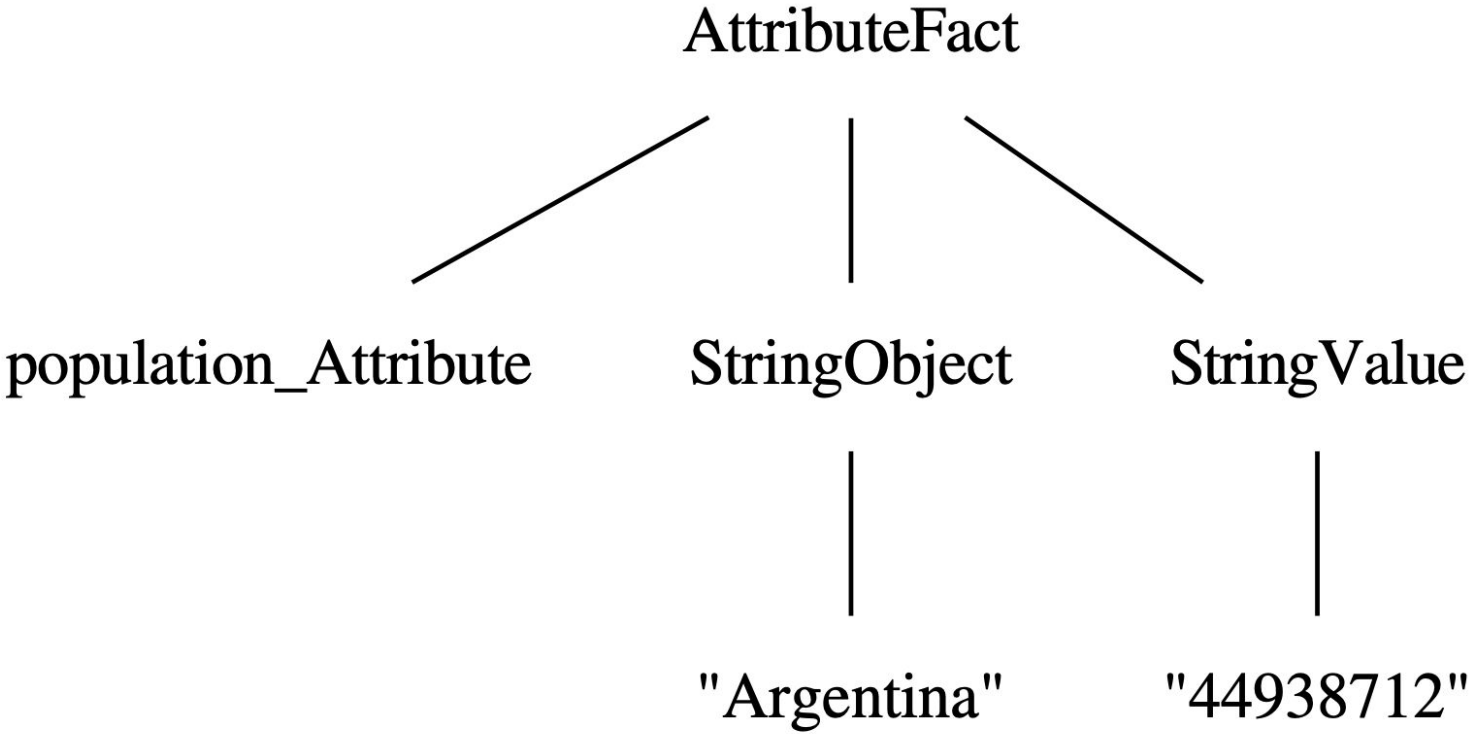
Stage 1:

Template-based generation of atomic facts

the capital of Argentina is Buenos Aires
the area of Argentina is 2780400
the population of Argentina is 44938712
the continent of Argentina is South America
the currency of Argentina is Argentine peso

```
abstract Facts = {

cat
  Fact ;
  Object ;
  Attribute ;
  Value ;
fun
  AttributeFact : Attribute -> Object -> Value -> Fact ;

  capital_Attribute : Attribute ;
  area_Attribute : Attribute ;
  population_Attribute : Attribute ;
  continent_Attribute : Attribute ;
  currency_Attribute : Attribute ;

  StringObject : String -> Object ;
  StringValue : String -> Value ;
}
```

https://github.com/aarneranta/NLG-examples/blob/main/doc/facts1/Facts.gf


AttributeFact
population_Attribute
StringObject
StringValue
"Argentina"
"44938712"

```
concrete FactsEng of Facts = {

lincat
  Fact = Str ;
  Object = Str ;
  Attribute = Str ;
  Value = Str ;

lin
  AttributeFact attr obj val =
    "the" ++ attr ++ "of" ++ obj ++ "is" ++ val ;

  capital_Attribute = "capital" ;
  area_Attribute = "area" ;
  population_Attribute = "population" ;
  continent_Attribute = "continent" ;
  currency_Attribute = "currency" ;

  StringObject str = str.s ;
  StringValue str = str.s ;
}
```

```
def country_facts(c):
  object = pgf.Expr('StringObject',[string_expr(c.country)])
  return [
    pgf.Expr('AttributeFact',
           [pgf.Expr(attr,[]),object,pgf.Expr('StringValue',[string_expr(val)])])

      for (attr,val) in [
        ('capital_Attribute', c.capital),
        ('area_Attribute', c.area),
        ('population_Attribute', c.population),
        ('continent_Attribute', c.continent),
        ('currency_Attribute', c.currency)
        ]
    ]
```

https://github.com/aarneranta/NLG-examples/blob/main/doc/facts1/facts.py

```
def main():
    gr = pgf.readPGF(pgf_file)
    countries = get_countries(country_file)
    langs = list(gr.languages.values())
    for lang in langs:
        text = []
        for c in countries:
            for t in country_facts(c):
                text.append(lang.linearize(t))
        print('\n'.join(text))
```

```
concrete FactsGer of Facts = {

lincat
  Fact = Str ;
  Object = Str ;
  Attribute = Str ;
  Value = Str ;

lin
  AttributeFact attr obj val =
    attr ++ "von" ++ obj ++ "ist" ++ val ;

  capital_Attribute = "die Hauptstadt" ;
  area_Attribute = "die Fläche" ;
  population_Attribute = "die Einwohnerzahl" ;
  continent_Attribute = "der Kontinent" ;
  currency_Attribute = "die Währung" ;

  StringObject str = str.s ;
  StringValue str = str.s ;
}
```

die Hauptstadt von Argentina ist Buenos Aires
die Fläche von Argentina ist 2780400
die Einwohnerzahl von Argentina ist 44938712
der Kontinent von Argentina ist South America
die Währung von Argentina ist Argentine peso

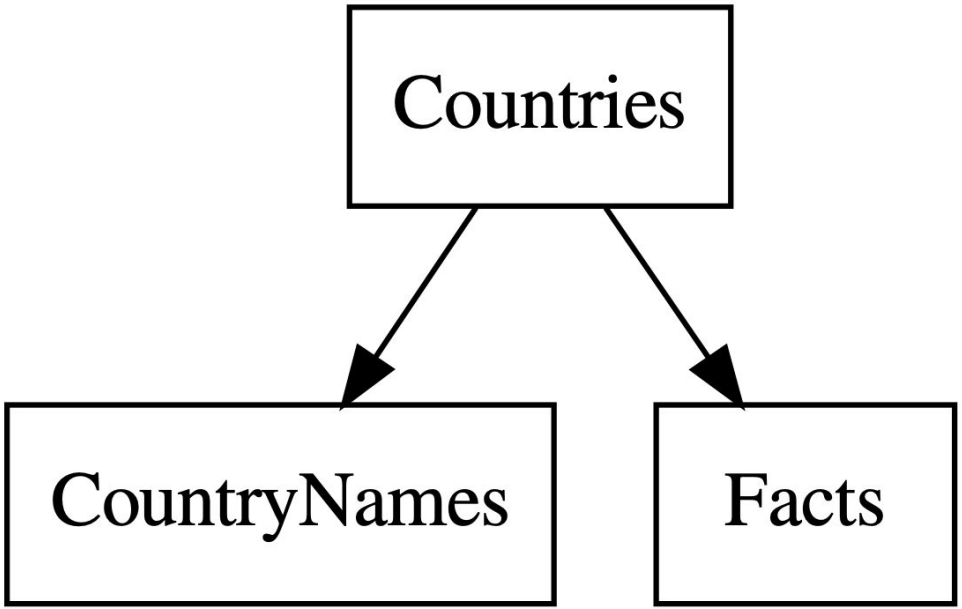
```
concrete FactsFin of Facts = {

lincat
  Fact = Str ;
  Object = Str ;
  Attribute = Str ;
  Value = Str ;

lin
  AttributeFact attr obj val =
    "maan" ++ obj ++ attr ++ "on" ++ val ;

  capital_Attribute = "pääkaupunki" ;
  area_Attribute = "pinta-ala" ;
  population_Attribute = "asukasluku" ;
  continent_Attribute = "maanosa" ;
  currency_Attribute = "valuutta" ;

  StringObject str = str.s ;
  StringValue str = str.s ;
}
```

maan Argentina pääkaupunki on Buenos Aires
maan Argentina pinta-ala on 2780400
maan Argentina asukasluku on 44938712
maan Argentina maanosa on South America
maan Argentina valuutta on Argentine peso

```
aarne$ gf -make Facts???.gf
- compiling Facts.gf...   write file Facts.gfo
- compiling FactsEng.gf...   write file FactsEng.gfo
- compiling FactsFin.gf...   write file FactsFin.gfo
- compiling FactsGer.gf...   write file FactsGer.gfo
linking ... OK
Writing Facts.pgf...

aarne$ python3 facts.py
the capital of Afghanistan is Kabul
the area of Afghanistan is 652230
the population of Afghanistan is 36643815
the continent of Afghanistan is Asia
the currency of Afghanistan is Afghan afghani

the capital of Albania is Tirana
```

Stage 2:

Grammar-based generation of atomic facts


Countries
CountryNames
Facts

```
abstract Facts = {

cat
  Fact ;
  Object ;
  Attribute ;
  Value ;
  Name ;

fun
  AttributeFact : Attribute -> Object -> Value -> Fact ;

  NameObject : Name -> Object ;
  NameValue : Name -> Value ;
  IntValue : Int -> Value ;

}
```

https://github.com/aarneranta/NLG-examples/blob/main/doc/facts2/Facts.gf

```
concrete FactsEng of Facts =
  open SyntaxEng, SymbolicEng in {

lincat
  Fact = Cl ;
  Object = NP ;
  Attribute = CN ;
  Value = NP ;
  Name = NP ;

lin
  AttributeFact attr obj val =
    mkCl (mkNP the_Det (mkCN attr (mkAdv possess_Prep obj))) val ;
  NameObject name = name ;
  NameValue name = name ;
  IntValue int = symb int ;
}
```

the population of United States of America is 331449281

→ (**grammar for names**)

the population of the United States is 331449281

→ (**attribute-specific rendering**)

the United States has 331449281 inhabitants

```
concrete FactsGer of Facts =
  open SyntaxGer, SymbolicGer in {

lincat
  Fact = Cl ;
  Object = NP ;
  Attribute = CN ;
  Value = NP ;
  Name = NP ;

lin
  AttributeFact attr obj val =
    mkCl (mkNP the_Det (mkCN attr (mkAdv possess_Prep obj))) val ;
  NameObject name = name ;
  NameValue name = name ;
  IntValue int = symb int ;
}
```

die Einwohnerzahl von United States of America ist 331449281

→

die Einwohnerzahl von den Vereinigten Staaten ist 331449281

→

die Vereinigten Staaten haben 331449281 Einwohner

```
concrete FactsFin of Facts =
  open SyntaxFin, SymbolicFin, (E=ExtendFin) in {

lincat
  Fact = Cl ;
  Object = NP ;
  Attribute = CN ;
  Value = NP ;
  Name = NP ;

lin
  AttributeFact attr obj val =
    mkCl (mkNP (E.GenNP obj) attr) val ;
  NameObject name = name ;
  NameValue name = name ;
  IntValue int = symb int ;
}
```

maan United States of America asukasluku 331449281

→

Yhdysvaltain asukasluku on 331449281

→

Yhdysvalloissa on 331449281 asukasta

```
select ?country ?countryLabelEn ?countryLabelDe ?countryLabelFi {
  ?country wdt:P31/wdt:P279* wd:Q3624078 .
  ?country rdfs:label ?countryLabelEn .
  ?country rdfs:label ?countryLabelDe .
  ?country rdfs:label ?countryLabelFi .
  filter(lang(?countryLabelEn)='en')
  filter(lang(?countryLabelDe)='de')
  filter(lang(?countryLabelFi)='fi')
}
```

244 results in 1217 ms | Code | Download | Link

Search

country	countryLabelEn	countryLabelDe	countryLabelFi
wd:Q1033	Nigeria	Nigeria	Nigeria
wd:Q16	Canada	Kanada	Kanada
wd:Q691	Papua New Guinea	Papua-Neuguinea	Papua-Uusi-Guinea

https://w.wiki/3mWG

```
aarne$ python3 extract_names.py

fun 'Porto-Novo_CName' : CName ;
lin 'Porto-Novo_CName' = mkCName "Porto-Novo" ;
fun Liberia_CName : CName ;
lin Liberia_CName = mkCName "Liberia" ;
fun Cyprus_CName : CName ;
lin Cyprus_CName = mkCName "Cyprus" ;
fun Gaborone_CName : CName ;
lin Gaborone_CName = mkCName "Gaborone" ;
fun Australian_dollar_CName : CName ;
lin Australian_dollar_CName = mkCName "Australian dollar" ;
```

https://github.com/aarneranta/NLG-examples/blob/main/doc/facts2/extract_names.py

```
abstract CountryNames = {

cat CName ;

-- generated
fun Africa_CName : CName ;
fun Asia_CName : CName ;
fun Central_America_CName : CName ;
fun Europe_CName : CName ;
fun North_America_CName : CName ;
fun South_America_CName : CName ;
fun insular_Oceania_CName : CName ;

fun United_States_of_America_CName : CName ;
```

```
concrete CountryNamesEng of CountryNames =
  open SyntaxEng, ParadigmsEng in {

lincat CName = NP ;

oper mkCName : Str -> NP = \s -> mkNP (mkPN s) ;

lin Africa_CName = mkCName "Africa" ;
lin Asia_CName = mkCName "Asia" ;
lin Central_America_CName = mkCName "Central America" ;
lin Europe_CName = mkCName "Europe" ;
lin North_America_CName = mkCName "North America" ;
lin South_America_CName = mkCName "South America" ;
lin insular_Oceania_CName = mkCName "insular Oceania" ;

lin United_States_of_America_CName = mkCName "the United States" ;
```

```
aarne$ python3 extract_names.py | grep "lin "

lin Maldivian_rufiyaa_CName = mkCName "Rufiyaa" ;
lin Muscat_CName = mkCName "Maskat" ;
lin Brazilian_real_CName = mkCName "brasilianischer Real" ;
lin Russian_ruble_CName = mkCName "russischer Rubel" ;
lin Gaborone_CName = mkCName "Gaborone" ;
lin Malaysian_ringgit_CName = mkCName "Ringgit" ;
lin Chilean_peso_CName = mkCName "chilenischer Peso" ;
lin Canadian_dollar_CName = mkCName "kanadischer Dollar" ;
```

```
source_field = 1  # English
target_field = 4  # German
```

```
concrete CountryNamesGer of CountryNames =
  open SyntaxGer, ParadigmsGer in {

lincat CName = NP ;

oper mkCName = overload {
  mkCName : Str -> NP = \s -> mkNP (mkPN s) ;
  mkCName : NP -> NP = \np -> np ;
  } ;

lin 'Guinea-Bissau_CName' = mkCName "Guinea-Bissau" ;

lin United_States_of_America_CName =
  mkCName (mkNP thePl_Det
    (mkCN (mkA "Vereinigt") (mkN "Staat" "Staaten" masculine))) ;
```

```
concrete CountryNamesFin of CountryNames =
  open SyntaxFin, ParadigmsFin, Prelude in {

lincat CName = LocName ;

oper LocName = {np : NP ; isIn : Bool} ;

oper mkCName = overload {
  mkCName : Str -> LocName = \s -> {np = mkNP (foreignPN s) ; isIn = True} ;
  mkCName : N -> LocName = \n -> {np = mkNP n ; isIn = True} ;
  mkCName : NP -> LocName = \np -> {np = np ; isIn = True} ;
  } ;

 exCName : LocName -> LocName = \name -> name ** {isIn = False} ;
 sgCName : LocName -> LocName = \name -> name ** {np = forceNumberNP singular name.np} ;

lin Finland_CName = mkCName (mkN "Suomi" "Suomia") ;
lin Russia_CName = exCName (mkCName "Venäjä") ;
lin United_States_of_America_CName =
    sgCName (mkCName (mkNP thePl_Det (exceptPlGenN (mkN "Yhdysvalta") "Yhdysvaltain"))) ;
```

```
abstract Countries = Facts, CountryNames ** {
fun
-- using CNames
  cName : CName -> Name ;

-- basic properties
  capital_Attribute : Attribute ;
  area_Attribute : Attribute ;
  population_Attribute : Attribute ;
  continent_Attribute : Attribute ;
  currency_Attribute : Attribute ;

-- specialized expressions for properties
  populationFact : CName -> Int -> Fact ;
  continentFact : CName -> CName -> Fact ;
}
```

```
concrete CountriesEng of Countries = FactsEng, CountryNamesEng **
  open SyntaxEng, ParadigmsEng, SymbolicEng in {
lin
  cName name = name ;

  capital_Attribute = mkAttribute "capital" ;
  area_Attribute = mkAttribute "area" ;
  population_Attribute = mkAttribute "population" ;
  continent_Attribute = mkAttribute "continent" ;
  currency_Attribute = mkAttribute "currency" ;

  populationFact cname int =
    mkCl cname have_V2 (mkNP <symb int : Card> (mkN "inhabitant")) ;
  continentFact cname name = mkCl cname (SyntaxEng.mkAdv in_Prep name) ;

oper
  mkAttribute : Str -> CN = \s -> mkCN (mkN s) ;
}
```

```
concrete CountriesFin of Countries = FactsFin, CountryNamesFin **
  open SyntaxFin, ParadigmsFin, SymbolicFin, Prelude in {
lin
  cName name = name.np ;
  capital_Attribute = mkAttribute "pääkaupunki" ;
  area_Attribute = mkAttribute "pinta-ala" ;
  population_Attribute = mkAttribute "asukasluku" ;
  continent_Attribute = mkAttribute "maanosa" ;
  currency_Attribute = mkAttribute "valuutta" ;

  populationFact cname int =
    mkCl cname.np (mkV2 (caseV (locCase cname) have_V2))
      (mkNP <symb int : Card> (mkN "asukas")) ;
  continentFact cname name =
    mkCl cname.np (SyntaxFin.mkAdv (casePrep (locCase name)) name.np) ;

oper
  mkAttribute : Str -> CN = \s -> mkCN (mkN s) ;
  locCase : LocName -> Case = \name -> case name.isIn of {
    True => inessive ;
    False => adessive
    } ;
}
```

```
class FactSystem:

    def __init__(self,fnames,gr,lang1):
        self.fieldnames = fnames
        self.grammar = gr
        self.language1 = lang1   # the language in which entities are parsed to trees

    def get_data(self,filename):
        data = []
        Data = namedtuple('Data', self.fieldnames)
        file = open(filename)
        for line in file:
            fields = Data(*line.split('\t'))
            data.append(fields)
        return data
```

https://github.com/aarneranta/NLG-examples/blob/main/doc/facts2/data_facts.py

```
class FactSystem:

    ...

    def run(self,datafile,fact_generator):
        gr = self.grammar
        data = sorted(list(self.get_data(datafile)))
        langs = list(gr.languages.values())
        for lang in langs:
            text = []
            for tree in fact_generator(self,data):
                lin = lang.linearize(tree)
                text.append(lin[0].upper() + lin[1:])
            print('\n'.join(text))
```

https://github.com/aarneranta/NLG-examples/blob/main/doc/facts2/data_facts.py

```
def example_run():

    factsys = FactSystem(
        'country capital area population continent currency',
        pgf.readPGF('Countries.pgf'),
        'CountriesEng'
        )

    factsys.run('../data/countries.tsv',simple_facts)
```

https://github.com/aarneranta/NLG-examples/blob/main/doc/facts2/data_facts.py

```
def simple_facts(factsys,data):

    "for each tuple in data, generate an attribute fact for each field"

    fields = factsys.fieldnames.split()
    facts = []
    for tuple in data:
        object = factsys.str2exp("Object",tuple[0])
        for (attr,val) in [(fields[i],tuple[i]) for i in range(1,len(fields))]:
            fact = pgf.Expr("AttributeFact", [
                    factsys.str2exp("Attribute",attr),
                    object,
                    factsys.str2exp("Value",val)])
            facts.append(fact)
    return facts
```

https://github.com/aarneranta/NLG-examples/blob/main/doc/facts2/data_facts.py

```
class FactSystem:

    ...

    def str2exp(self,cat,s):
        eng = self.grammar.languages[self.language1]
        try:
            pp = eng.parse(s,cat=pgf.readType(cat))
            _,e = pp.__next__()
            return e
        except:
            print("WARNING:","no", cat, "from", s,"with",self.language1)
            return pgf.Expr(s,[])
```

https://github.com/aarneranta/NLG-examples/blob/main/doc/facts2/data_facts.py

```
aarne$ python3 data_facts.py
...
The capital of the United States is Washington, D.C.
The area of the United States is 9826675
The population of the United States is 331449281
The continent of the United States is North America
The currency of the United States is United States dollar
...
Yhdysvaltain pääkaupunki on Washington
Yhdysvaltain pinta-ala on 9826675
Yhdysvaltain asukasluku on 331449281
Yhdysvaltain maanosa on Pohjois-Amerikka
Yhdysvaltain valuutta on Yhdysvaltain dollari
...
Die Hauptstadt von den Vereinigten Staaten ist Washington, D.C.
Die Fläche von den Vereinigten Staaten ist 9826675
Die Einwohnerzahl von den Vereinigten Staaten ist 331449281
Der Kontinent von den Vereinigten Staaten ist Nordamerika
Die Währung von den Vereinigten Staaten ist US-Dollar
```

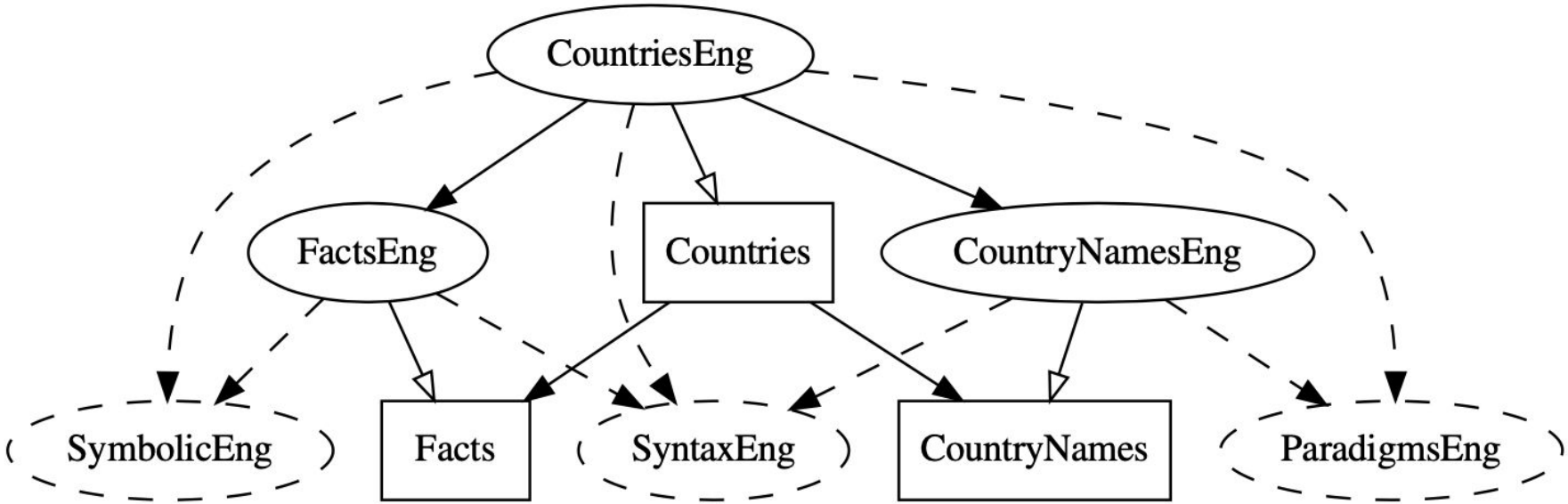
```
aarne$ python3 country_facts.py
...
The capital of the United States is Washington, D.C.
The area of the United States is 9826675
The United States has 331449281 inhabitants
The United States is in North America
The currency of the United States is United States dollar
...
Yhdysvaltain pääkaupunki on Washington
Yhdysvaltain pinta-ala on 9826675
Yhdysvalloissa on 331449281 asukasta
Yhdysvallat on Pohjois-Amerikassa
Yhdysvaltain valuutta on Yhdysvaltain dollari
...
Die Hauptstadt von den Vereinigten Staaten ist Washington, D.C.
Die Fläche von den Vereinigten Staaten ist 9826675
Die Vereinigten Staaten haben 331449281 Einwohner
Die Vereinigten Staaten liegen in Nordamerika
Die Währung von den Vereinigten Staaten ist US-Dollar
```

```
def country_facts_embedded(factsys,tuple):
    countr = factsys.str2exp("CName",tuple[0])
    cap    = factsys.str2exp('Name',tuple.capital)
    cont   = factsys.str2exp('CName',tuple.continent)
    curr   = factsys.str2exp('Name',tuple.currency)
    pop    = mkInt(tuple.population)
    are    = mkInt(tuple.area)

    factsys.grammar.embed("G")
    import G
    object = G.NameObject(G.cName(countr))

    return [
      G.AttributeFact(G.capital_Attribute, object, G.NameValue(cap)),
      G.AttributeFact(G.area_Attribute, object, G.IntValue(are)),
      G.populationFact(countr, pop),
      G.continentFact(countr, cont),
      G.AttributeFact(G.currency_Attribute, object, G.NameValue(curr))
      ]
```

https://github.com/aarneranta/NLG-examples/blob/main/doc/facts2/country_facts.py

```
def country_facts_parsed(factsys,tuple):
    countr = factsys.data2lin("CName",tuple[0])
    cap    = factsys.data2lin('Name',tuple.capital)
    cont   = factsys.data2lin('CName',tuple.continent)
    curr   = factsys.data2lin('Name',tuple.currency)
    pop    = mkInt(tuple.population)
    are    = mkInt(tuple.area)

    return [ factsys.str2exp('Fact',s) for s in
        [
        "the capital of {} is {}".format(countr,cap),   # templates are back!
        "the area of {} is {}".format(countr,are),
        "{} has {} inhabitants".format(countr,pop),
        "{} is in {}".format(countr, cont),
        "the currency of {} is {}".format(countr,are)
        ]
      ]
```

https://github.com/aarneranta/NLG-examples/blob/main/doc/facts2/country_facts.py

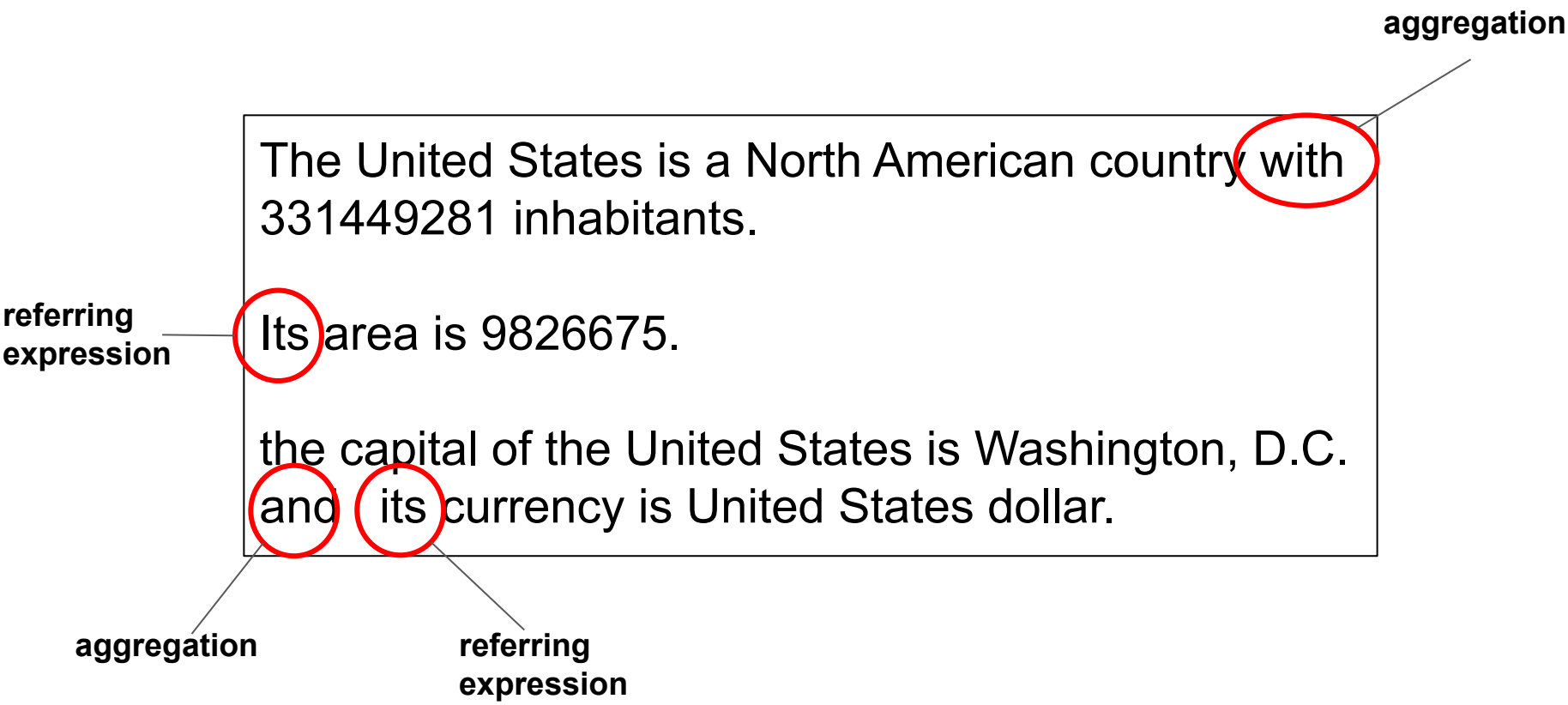

CountriesEng
FactsEng
Countries
CountryNamesEng
SymbolicEng
Facts
SyntaxEng
CountryNames
ParadigmsEng

Stage 3:

Building a fluent text

```
aarne$ cd ../facts3
aarne$ python3 country_facts.py
...
The United States is a North American country with 331449281
inhabitants. its area is 9826675. the capital of the United States
is Washington, D.C. and its currency is United States dollar.
...
Yhdysvallat on pohjoisamerikkalainen maa, jossa on 331449281
asukasta. sen pinta-ala on 9826675. Yhdysvaltain pääkaupunki on
Washington ja sen valuutta on Yhdysvaltain dollari.
...
Die Vereinigten Staaten sind ein Nordamerikanisches Land mit
331449281 Einwohnern. ihre Fläche ist 9826675. die Hauptstadt von
den Vereinigten Staaten ist Washington, D.C. und ihre Währung ist
US-Dollar.
```

https://github.com/aarneranta/NLG-examples/blob/main/doc/facts3/country_facts.py


aggregation
The United States is a North American country with 331449281 inhabitants.
referring expression
Its area is 9826675.
the capital of the United States is Washington, D.C.
and its currency is United States dollar.
aggregation
referring expression

```
abstract Facts = {

cat
  Doc ;       -- complete document
  Sentence ;  -- sentence with determinate tense and polarity
  Fact ;      -- predicative clause whose tense and polarity can vary
  Object ;    -- argument in predication, either constant or pronoun

fun
  OneSentenceDoc : Sentence -> Doc ;           -- S.
  AddSentenceDoc : Doc -> Sentence -> Doc ;    -- D. S.

  ConjSentence : Sentence -> Sentence -> Sentence ; -- S and S
  FactSentence : Fact -> Sentence ;                 -- F

  NameObject : Name -> Object ;  -- N
  PronObject : Name -> Object ;  -- it
```

https://github.com/aarneranta/NLG-examples/blob/main/doc/facts3/Facts.gf

```
def country_texts_embedded(factsys,data):
...
        doc = G.OneSentenceDoc(
                G.FactSentence(
                    G.KindFact(G.NameObject(countr),
                        G.ModifierKind(G.PropertyKind(G.cdProperty(cont),G.country_Kind),
                            G.NumericKindModifier(G.IntNumeric(pop),G.inhabitant_Kind)))))
        doc = G.AddSentenceDoc(doc,
                   G.FactSentence(G.AttributeFact(G.area_Attribute, G.PronObject(countr),
                                                  G.NumericValue(G.IntNumeric(are)))))
        doc = G.AddSentenceDoc(doc,
                    G.ConjSentence(
                      G.FactSentence(G.AttributeFact(G.capital_Attribute, G.NameObject(countr), G.NameValue(cap))),
                      G.FactSentence(G.AttributeFact(G.currency_Attribute, G.PronObject(countr), G.NameValue(curr)))))
...
```

https://github.com/aarneranta/NLG-examples/blob/main/doc/facts3/country_facts.py

```
def country_texts_parsed(factsys,data):
  ...
  doc = factsys.str2exp("Doc",
          ("{} is a {} country with {} inhabitants. "
           "its area is {} . "
           "the capital of {} is {} and its currency is {}.").
                       format(countr,cont,pop,are,countr,cap,curr))
  ...
```

https://github.com/aarneranta/NLG-examples/blob/main/doc/facts3/country_facts.py

Functor-based concrete syntax

```
abstract Facts = {
cat
  Doc ;       -- complete document
  Sentence ;  -- sentence with determinate tense and polarity
  Fact ;      -- predicative clause whose tense and polarity can vary
  Object ;    -- argument in predication, either constant or pronoun
  Property ;  -- modifying adjectival phrase, e.g. "European"
  Attribute ; -- single property of an object, e.g. "population"
  Modifier ;  -- post-modifier, e.g. adverbial phrase or relative clause
  Kind ;      -- type of objects, e.g. "European country"
  Value ;     -- value of an attribute, such as entity name or numeric
  Name ;      -- name of an entity, e.g. "Honduras", "South America"
  Numeric ;   -- cardinal number, e.g. "23", "100 million", "over a billion"

fun
  OneSentenceDoc : Sentence -> Doc ;           -- S.
  AddSentenceDoc : Doc -> Sentence -> Doc ;    -- D. S.

  ConjSentence : Sentence -> Sentence -> Sentence ; -- S and S
  FactSentence : Fact -> Sentence ;                 -- F

  KindFact : Object -> Kind -> Fact ;               -- O is a K
  PropertyFact : Object -> Property -> Fact ;       -- O is P
  AttributeFact : Attribute -> Object -> Value -> Fact ; -- the A of O is V
  PropertyKind : Property -> Kind -> Kind ;  -- P K
  ModifierKind : Kind -> Modifier -> Kind ;  -- K M
  NumericKindModifier : Numeric -> Kind -> Modifier ; -- with N K

  NameObject : Name -> Object ;  -- N
  PronObject : Name -> Object ;  -- it

  NumericKindValue : Numeric -> Kind -> Value ; -- N K
  NameValue : Name -> Value ;                   -- N
  NumericValue : Numeric -> Value ;             -- N V
  IntNumeric : Int -> Numeric ;          -- I
  IntMillionNumeric : Int -> Numeric ;   -- I million
  IntBillionNumeric : Int -> Numeric ;   -- I billion
  IntTrillionNumeric : Int -> Numeric ;  -- I billion
  a_billion_Numeric : Numeric ;          -- a billion
  AboutNumeric : Numeric -> Numeric ;    -- about N
  OverNumeric : Numeric -> Numeric ;     -- over N
  UnderNumeric : Numeric -> Numeric ;    -- over N

--------------------
-- data aggregation
cat
  [Object] {2} ;

fun
  ConjObject : [Object] -> Object ;  -- O, O and O

  NumericKindFact : Numeric -> Kind -> Fact ;                    -- there are N K
  NumericKindModifierFact : Numeric -> Kind -> Modifier -> Fact ; -- there are N K M
  MaxObjectAttributeFact : Object -> Attribute -> Fact ;         -- O has the largest A
  MinObjectAttributeFact : Object -> Attribute -> Fact ;         -- O has the smallest A
  SumAttributeFact : Attribute -> Object -> Numeric -> Fact ;    -- the total A of O is N

  UniqueInKindFact : Object -> Kind -> Fact ;  -- O is the only K
}
```

https://github.com/aarneranta/NLG-examples/blob/main/doc/facts3/Facts.gf

```
concrete FactsEng of Facts = open
  SyntaxEng,
  SymbolicEng,
  GrammarEng,
  Prelude
in {
lincat
  Doc = Text ;
  Sentence = S ;
  Fact = Cl ;
  Object = {np : NP ; pron : Pron ; isPron : Bool} ;
  Property = AP ;
  Attribute = CN ;
  Modifier = {adv : Adv ; rs : RS ; isAdv : Bool} ;
  Kind = CN ;
  Value = NP ;
  Name = NP ;
  Numeric = Card ;
lin
  OneSentenceDoc sent = mkText sent ;
  AddSentenceDoc doc sent = mkText doc (mkText sent) ;
  ConjSentence a b = mkS and_Conj a b ;
  FactSentence fact = mkS presentTense positivePol fact ;
  KindFact obj kind = mkCl obj.np (mkNP a_Det kind) ; --- sind ein Land
  PropertyFact obj prop = mkCl obj.np prop ;
  AttributeFact attr obj val = case obj.isPron of {
    True => mkCl (mkNP (mkDet obj.pron) attr) val ;
    _ => mkCl (mkNP the_Det (mkCN attr (mkAdv possess_Prep obj.np))) val
    } ;
  PropertyKind prop kind = mkCN prop kind ;
  ModifierKind kind mod = case mod.isAdv of {
    False => mkCN kind mod.rs ;
    True => mkCN kind mod.adv
    } ;
  NumericKindModifier num kind = mkModifier (mkAdv with_Prep (mkNP num kind)) ;
```

```
concrete FactsGer of Facts = open
  SyntaxGer,
  SymbolicGer,
  GrammarGer,
  Prelude
in {
lincat
  Doc = Text ;
  Sentence = S ;
  Fact = Cl ;
  Object = {np : NP ; pron : Pron ; isPron : Bool} ;
  Property = AP ;
  Attribute = CN ;
  Modifier = {adv : Adv ; rs : RS ; isAdv : Bool} ;
  Kind = CN ;
  Value = NP ;
  Name = NP ;
  Numeric = Card ;
lin
  OneSentenceDoc sent = mkText sent ;
  AddSentenceDoc doc sent = mkText doc (mkText sent) ;
  ConjSentence a b = mkS and_Conj a b ;
  FactSentence fact = mkS presentTense positivePol fact ;
  KindFact obj kind = mkCl obj.np (mkNP a_Det kind) ; --- sind ein Land
  PropertyFact obj prop = mkCl obj.np prop ;
  AttributeFact attr obj val = case obj.isPron of {
    True => mkCl (mkNP (mkDet obj.pron) attr) val ;
    _ => mkCl (mkNP the_Det (mkCN attr (mkAdv possess_Prep obj.np))) val
    } ;
  PropertyKind prop kind = mkCN prop kind ;
  ModifierKind kind mod = case mod.isAdv of {
    False => mkCN kind mod.rs ;
    True => mkCN kind mod.adv
    } ;
  NumericKindModifier num kind = mkModifier (mkAdv with_Prep (mkNP num kind)) ;
```

```
concrete FactsEng of Facts = open
  SyntaxEng,
  SymbolicEng,
  GrammarEng,
  Prelude
in {
lincat
  Doc = Text ;
  Sentence = S ;
  Fact = Cl ;
  Object = {np : NP ; pron : Pron ; isPron : Bool} ;
  Property = AP ;
  Attribute = CN ;
  Modifier = {adv : Adv ; rs : RS ; isAdv : Bool} ;
  Kind = CN ;
  Value = NP ;
  Name = NP ;
  Numeric = Card ;
lin
  OneSentenceDoc sent = mkText sent ;
  AddSentenceDoc doc sent = mkText doc (mkText sent) ;
  ConjSentence a b = mkS and_Conj a b ;
  FactSentence fact = mkS presentTense positivePol fact ;
  KindFact obj kind = mkCl obj.np (mkNP a_Det kind) ; --- sind ein Land
  PropertyFact obj prop = mkCl obj.np prop ;
  AttributeFact attr obj val = case obj.isPron of {
    True => mkCl (mkNP (mkDet obj.pron) attr) val ;
    _ => mkCl (mkNP the_Det (mkCN attr (mkAdv possess_Prep obj.np))) val
    } ;
  PropertyKind prop kind = mkCN prop kind ;
  ModifierKind kind mod = case mod.isAdv of {
    False => mkCN kind mod.rs ;
    True => mkCN kind mod.adv
    } ;
  NumericKindModifier num kind = mkModifier (mkAdv with_Prep (mkNP num kind)) ;
```

```
concrete FactsGer of Facts = open
  SyntaxGer,
  SymbolicGer,
  GrammarGer,
  Prelude
in {
lincat
  Doc = Text ;
  Sentence = S ;
  Fact = Cl ;
  Object = {np : NP ; pron : Pron ; isPron : Bool} ;
  Property = AP ;
  Attribute = CN ;
  Modifier = {adv : Adv ; rs : RS ; isAdv : Bool} ;
  Kind = CN ;
  Value = NP ;
  Name = NP ;
  Numeric = Card ;
lin
  OneSentenceDoc sent = mkText sent ;
  AddSentenceDoc doc sent = mkText doc (mkText sent) ;
  ConjSentence a b = mkS and_Conj a b ;
  FactSentence fact = mkS presentTense positivePol fact ;
  KindFact obj kind = mkCl obj.np (mkNP a_Det kind) ; --- sind ein Land
  PropertyFact obj prop = mkCl obj.np prop ;
  AttributeFact attr obj val = case obj.isPron of {
    True => mkCl (mkNP (mkDet obj.pron) attr) val ;
    _ => mkCl (mkNP the_Det (mkCN attr (mkAdv possess_Prep obj.np))) val
    } ;
  PropertyKind prop kind = mkCN prop kind ;
  ModifierKind kind mod = case mod.isAdv of {
    False => mkCN kind mod.rs ;
    True => mkCN kind mod.adv
    } ;
  NumericKindModifier num kind = mkModifier (mkAdv with_Prep (mkNP num kind)) ;
```

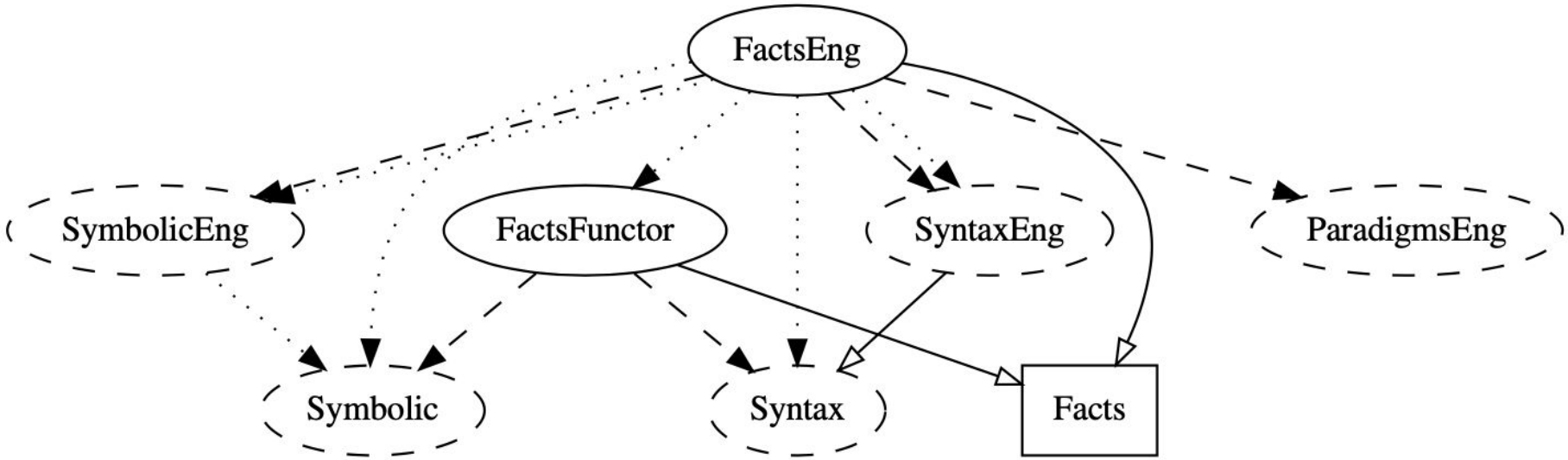
```
incomplete concrete FactsFunctor of Facts = open
  Syntax,
  Symbolic,
  Grammar,
  Prelude
in {
lincat
  Doc = Text ;
  Sentence = S ;
  Fact = Cl ;
  Object = {np : NP ; pron : Pron ; isPron : Bool} ;
  Property = AP ;
  Attribute = CN ;
  Modifier = {adv : Adv ; rs : RS ; isAdv : Bool} ;
  Kind = CN ;
  Value = NP ;
  Name = NP ;
  Numeric = Card ;
lin
  OneSentenceDoc sent = mkText sent ;
  AddSentenceDoc doc sent = mkText doc (mkText sent) ;
  ConjSentence a b = mkS and_Conj a b ;
  FactSentence fact = mkS presentTense positivePol fact ;
  KindFact obj kind = mkCl obj.np (mkNP a_Det kind) ; --- sind ein Land
  PropertyFact obj prop = mkCl obj.np prop ;
  AttributeFact attr obj val = case obj.isPron of {
    True => mkCl (mkNP (mkDet obj.pron) attr) val ;
    _ => mkCl (mkNP the_Det (mkCN attr (mkAdv possess_Prep obj.np))) val
    } ;
  PropertyKind prop kind = mkCN prop kind ;
  ModifierKind kind mod = case mod.isAdv of {
    False => mkCN kind mod.rs ;
    True => mkCN kind mod.adv
    } ;
  NumericKindModifier num kind = mkModifier (mkAdv with_Prep (mkNP num kind)) ;
```

A **functor** opens **interfaces** instead of complete resources.

```
concrete FactsEng of Facts = FactsFunctor with
  (Syntax = SyntaxEng),
  (Symbolic = SymbolicEng),
  (Grammar = GrammarEng)
```

Functor instantiation.

```
concrete FactsGer of Facts = FactsFunctor with
  (Syntax = SyntaxGer),
  (Symbolic = SymbolicGer),
  (Grammar = GrammarGer)
```


FactsEng
SymbolicEng
FactsFunctor
SyntaxEng
ParadigmsEng
Symbolic
Syntax
Facts

```
concrete FactsFin of Facts = FactsFunctor with
  (Syntax = SyntaxFin),
  (Symbolic = SymbolicFin),
  (Grammar = GrammarFin)
```

restricted inheritance with an **exclude list**

```
concrete FactsFin of Facts = FactsFunctor - [AttributeFact]
with
  (Syntax = SyntaxFin),
  (Symbolic = SymbolicFin),
  (Grammar = GrammarFin)

** open (E=ExtendFin) in {

lin
  AttributeFact attr obj val = mkCl (mkNP (E.GenNP obj.np) attr) val ;
}
```

```
concrete FactsEng of Facts = FactsFunctor with
  (Syntax = SyntaxEng),
  (Symbolic = SymbolicEng),
  (Grammar = GrammarEng)
  ** open ParadigmsEng, (E=ExtendEng) in {

-- functor parameters
oper
  largest_AP : AP = GrammarEng.AdjOrd (mkOrd (mkA "large")) ;
  smallest_AP : AP = GrammarEng.AdjOrd (mkOrd (mkA "small")) ;
  total_AP : AP = mkAP (mkA "total") ;
  only_AP : AP = mkAP (mkA "only") ;

  npNum : NP -> Num = \np ->
    case ifPluralNP np of {False => singularNum ; True => pluralNum} ;

-- functions left to instantiation
lin
  IntMillionNumeric int =
    E.CardCNCard <symb int : Card> (mkN "million" "million") ;
```

*N.B. This is a quick and dirty way: a clean way would be to write a separate **interface module**.*

Stage 4:

Selecting content

```
aarne$ python3 world_facts.py

There are 194 countries in the world.
The total population of the world is 7552 million.
People's Republic of China has the largest population and Russia has
the largest area.
India and People's Republic of China are the only countries with
over a billion inhabitants.

There are 54 countries in Africa.
The total population of Africa is 1253 million.
Nigeria has the largest population and Algeria has the largest area.
...
```

https://github.com/aarneranta/NLG-examples/blob/main/doc/facts3/world_facts.py

data aggregation

```
def continent_text(factsys,data,cont):
    cont_data = [d for d in data if cont in [d.continent,the_world]]

    ncountries = len(cont_data)
    largestpop = max(cont_data, key=lambda c: int(c.population)).country
    largestarea = max(cont_data, key=lambda c: int(c.area)).country
    totalpop = sum([int(c.population) for c in cont_data])//1000000

    doc = factsys.str2exp("Doc",
      ("there are {} countries in {}.").format(ncountries,cont))

    doc = G.AddSentenceDoc(doc, factsys.str2exp("Sentence",
      ("the total population of {} is {} million").format(cont,totalpop)))

    doc = G.AddSentenceDoc(doc, factsys.str2exp("Sentence",
      ("{} has the largest population and {} has the largest area".
                                        format(largestpop,largestarea)))
```

ellipsis

```
abstract Facts = {
...
-- data aggregation
cat
  [Object] {2} ;

fun
  ConjObject : [Object] -> Object ;  -- O, O and O

  NumericKindFact : Numeric -> Kind -> Fact ;                    -- there are N K
  NumericKindModifierFact : Numeric -> Kind -> Modifier -> Fact ; -- there are N K M
  MaxObjectAttributeFact : Object -> Attribute -> Fact ;         -- O has the largest A
  MinObjectAttributeFact : Object -> Attribute -> Fact ;         -- O has the smallest A
  SumAttributeFact : Attribute -> Object -> Numeric -> Fact ;    -- the total A of O is N

  UniqueInKindFact : Object -> Kind -> Fact ;  -- O is the only K
```

https://github.com/aarneranta/NLG-examples/blob/main/doc/facts3/Facts.gf

'X is the only K that P' is **correct by construction**

```
billions = [c.country for c in cont_data if int(c.population) > 1000000000]

if len(billions) == 1:
    object = billions[0] + ' is the only country '
elif len(billions) > 1:
    object = ', '.join(billions[:-1]) + ' and ' + billions[-1] +
             ' are the only countries '

if billions:
    doc = G.AddSentenceDoc(doc, factsys.str2exp('Sentence',
                object + ' with over a billion inhabitants'))
```

Who selects the content?

Stages 1 to 3:

- the SPARQL query selects the facts
- all facts are verbalized

Stage 4:

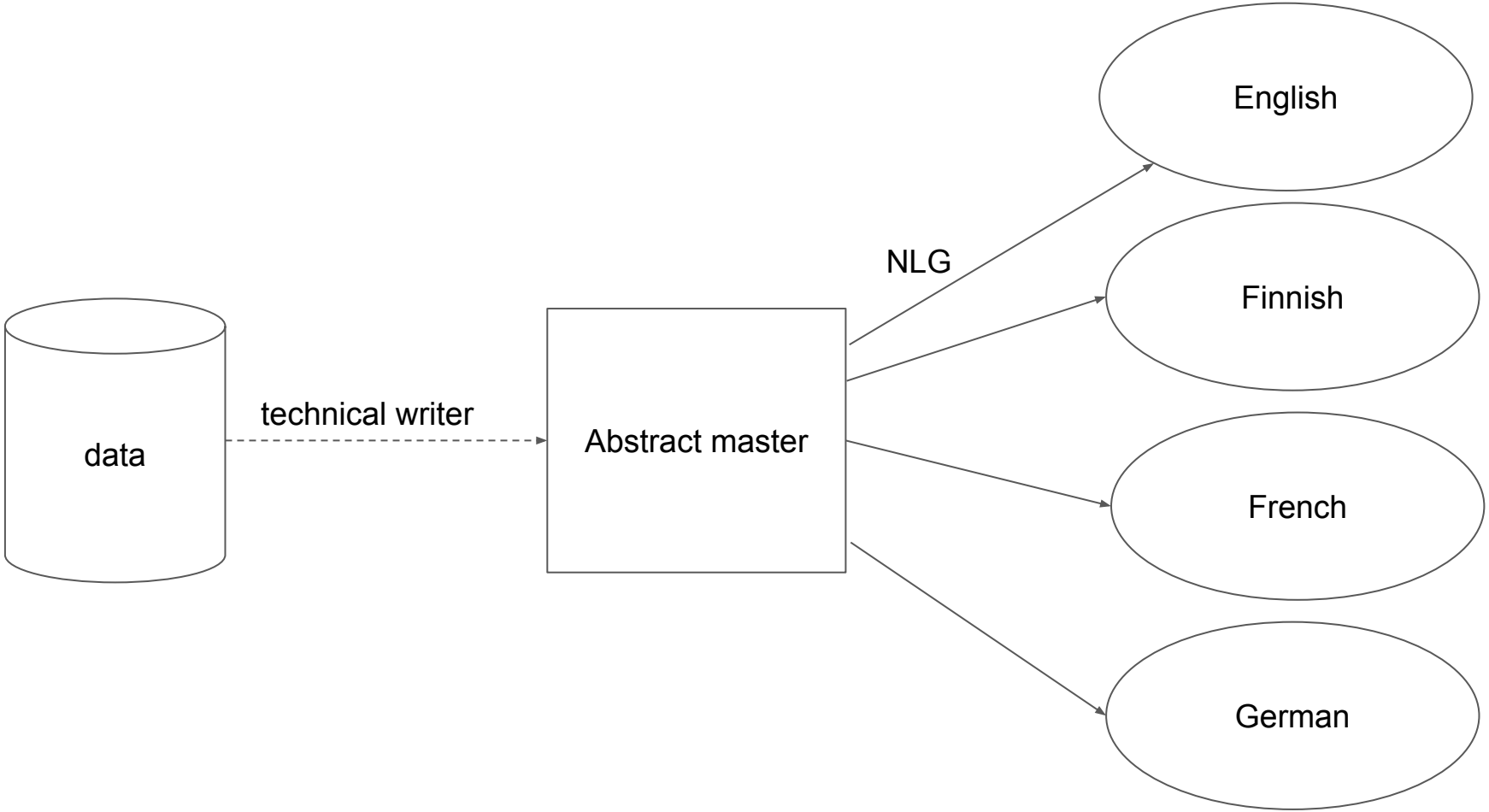
- function **world_facts.continent_text()** selects the data for each continent and for the world

Interactive selection

No algorithm can always select the most interesting data.

In the Abstract Wikipedia, **authors** of the documents must be able to select it.

Technically, this means that they **produce the master documents in abstract syntax**.


English
NLG
Finnish
data
technical writer
Abstract master
French
German

How to produce abstract master documents

Write abstract syntax trees

Write programs that produce abstract syntax trees, e.g. **Wikifunctions**

Write natural language and let GF parse it

- the language can be - but does not need to be - one of the NLG output languages (**FactSystem.language1**)
- it is in any case a **Controlled Natural Language (CNL)**, a language defined - and thereby controlled - by a GF grammar

Summary:

The NLG stages of Reiter & Dale


Reiter & Dale
content determination
discourse planning
sentence aggregation
lexicalization
referring expression generation
linguistic realization

Reiter & Dale	Stage 1	Stage 2	Stage 3	Stage 4
content determination	all facts in data			semantic aggregation
discourse planning	fact by fact		syntactic aggregation	
sentence aggregation			syntactic aggregation	
lexicalization	data strings	data labels		
referring expression generation	constants		constants and pronouns	ellipsis
linguistic realization	template	RGL	RGL functor	

Reiter & Dale	Stage 1	Stage 2	Stage 3	Stage 4	to do
content determination	all facts in data			semantic aggregation	interactive authoring
discourse planning	fact by fact		syntactic aggregation		collect text patterns
sentence aggregation			syntactic aggregation		aggregation in all categories
lexicalization	data strings	data labels			WordNet, concept alignment
referring expression generation	constants		constants and pronouns	ellipsis	definite descriptions
linguistic realization	template	RGL	RGL functor		language model optimization

Exercises

add	Stage 1	Stage 2	Stage 3	Stage 4
languages	GF	GF + Wikidata		
atomic fact expressions		GF + Python		
syntactic aggregations			GF + Python	
semantic aggregations				Python + optionally GF
domains of data	GF	GF + optionally Python		

repository: https://github.com/aarneranta/NLG-examples
relevant code: https://github.com/aarneranta/NLG-examples/tree/main/doc
full text of tutorial: https://github.com/aarneranta/NLG-examples/blob/main/doc/gf-nlg.pdf

No resource grammar yet?

No resource grammar yet?

needed at Stage 2

```
AdverbEng.PrepNP
ConstructorsEng.ComplV2
ConstructorsEng.DetArtCard
ConstructorsEng.the_Det
NounEng.AdvCN
NounEng.DetCN
NounEng.IndefArt
NounEng.UseN
NounEng.UsePN
ParadigmsEng.regN
ParadigmsEng.regPN
SentenceEng.PredVP
StructuralEng.have_V2
StructuralEng.in_Prep
StructuralEng.possess_Prep
SymbolEng.IntPN
SymbolEng.SymbNum
SymbolicEng.mkSymb
VerbEng.CompAdv
VerbEng.CompNP
VerbEng.UseComp
```

additionally needed at Stages 3 and 4

```
AdjectiveEng.AdjOrd
AdjectiveEng.PositA
ConjunctionEng.BaseNP
ConjunctionEng.BaseS
ConjunctionEng.ConjNP
ConjunctionEng.ConjS
ConjunctionEng.ConsNP
ConjunctionEng.ListNP,
ExtendEng.CardCNCard
IdiomEng.ExistNP
IdiomEng.ExistNPAdv
NounEng.AdNum
NounEng.AdjCN
NounEng.DetNP
NounEng.DetQuant
NounEng.NumSg
NounEng.OrdSuperl
NounEng.PossPron
NounEng.RelCN
NounEng.UsePron
ParadigmsEng.mkAdN
```

```
ParadigmsEng.mkAdv
ParadigmsEng.mkOrd
ParadigmsEng.regA
PhraseEng.NoPConj
PhraseEng.NoVoc
PhraseEng.PhrUtt
PhraseEng.UttS
RelativeEng.RelVP
StructuralEng.and_Conj
StructuralEng.it_Pron
StructuralEng.somewhere_Adv;
StructuralEng.they_Pron
StructuralEng.with_Prep
VerbEng.CompAP
```

Thanks!