A computational grammar for Maltese

John J. Camilleri
Chalmers / University of Gothenburg

4th International Conference on Maltese Linguistics 2013 — Lyon, France

About me

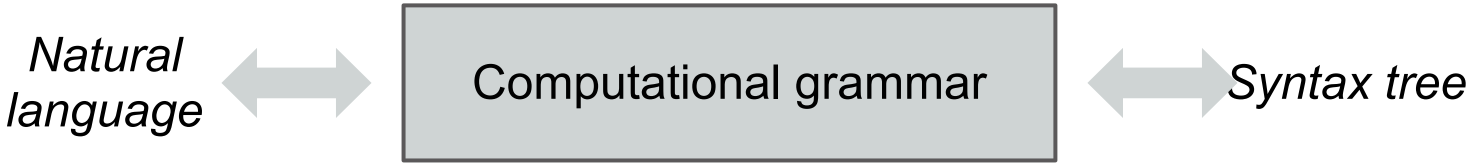
M.Sc. student & research assistant
Language Technology Research Group
Department of Computer Science and Engineering
Chalmers University of Technology /
University of Gothenburg, Sweden

About me

M.Sc. student & research assistant
Language Technology Research Group
Department of Computer Science and Engineering
Chalmers University of Technology /
University of Gothenburg, Sweden

Computational grammars

- Represent the grammar rules of a natural language as software
- Morphology and syntax



- Convert between surface input and abstract representation (e.g. parse trees)
- Validate input phrases as in/correct
- Produce grammatically-correct phrases

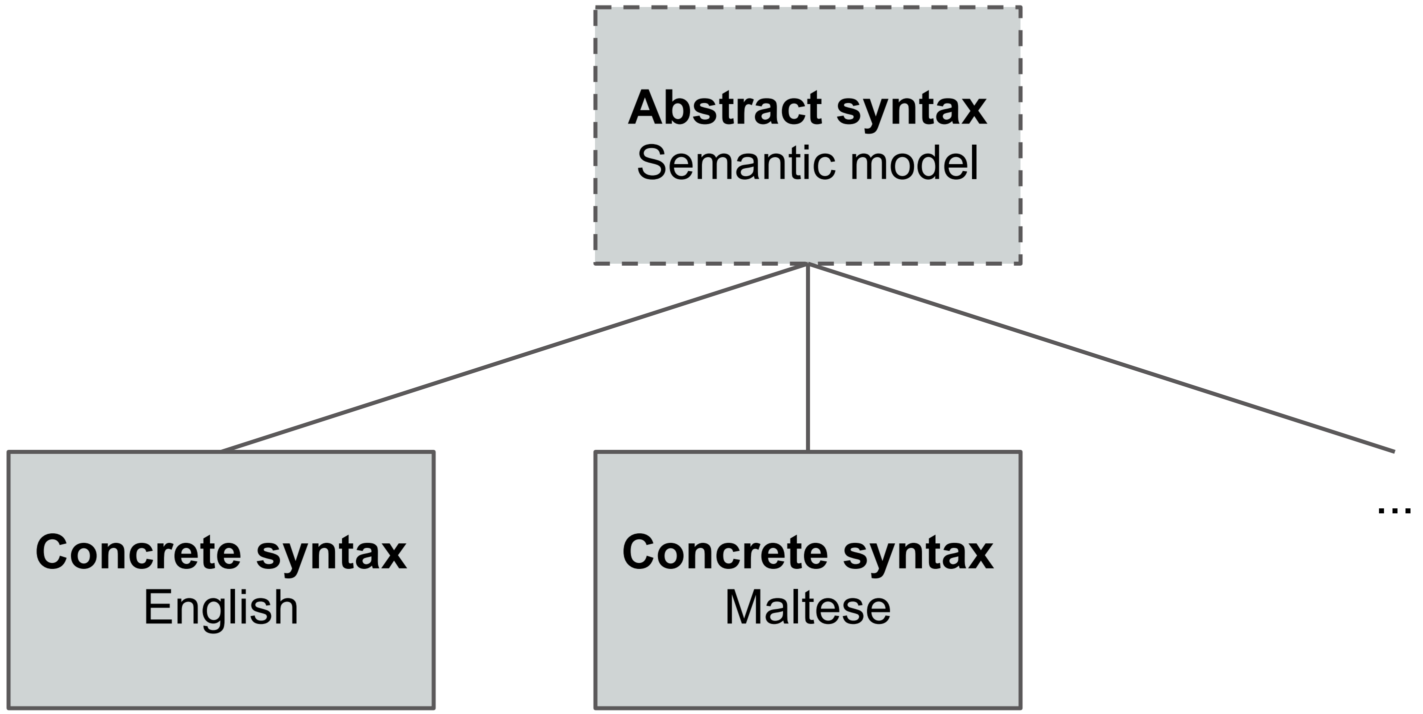
The Grammatical Framework

- A programming language for multilingual grammars
- Language-independent interlingua for modelling semantics
- Tool for rule-based translation

- Created by Aarne Ranta in 1998

  http://www.grammaticalframework.org/

Abstract & concrete syntaxes

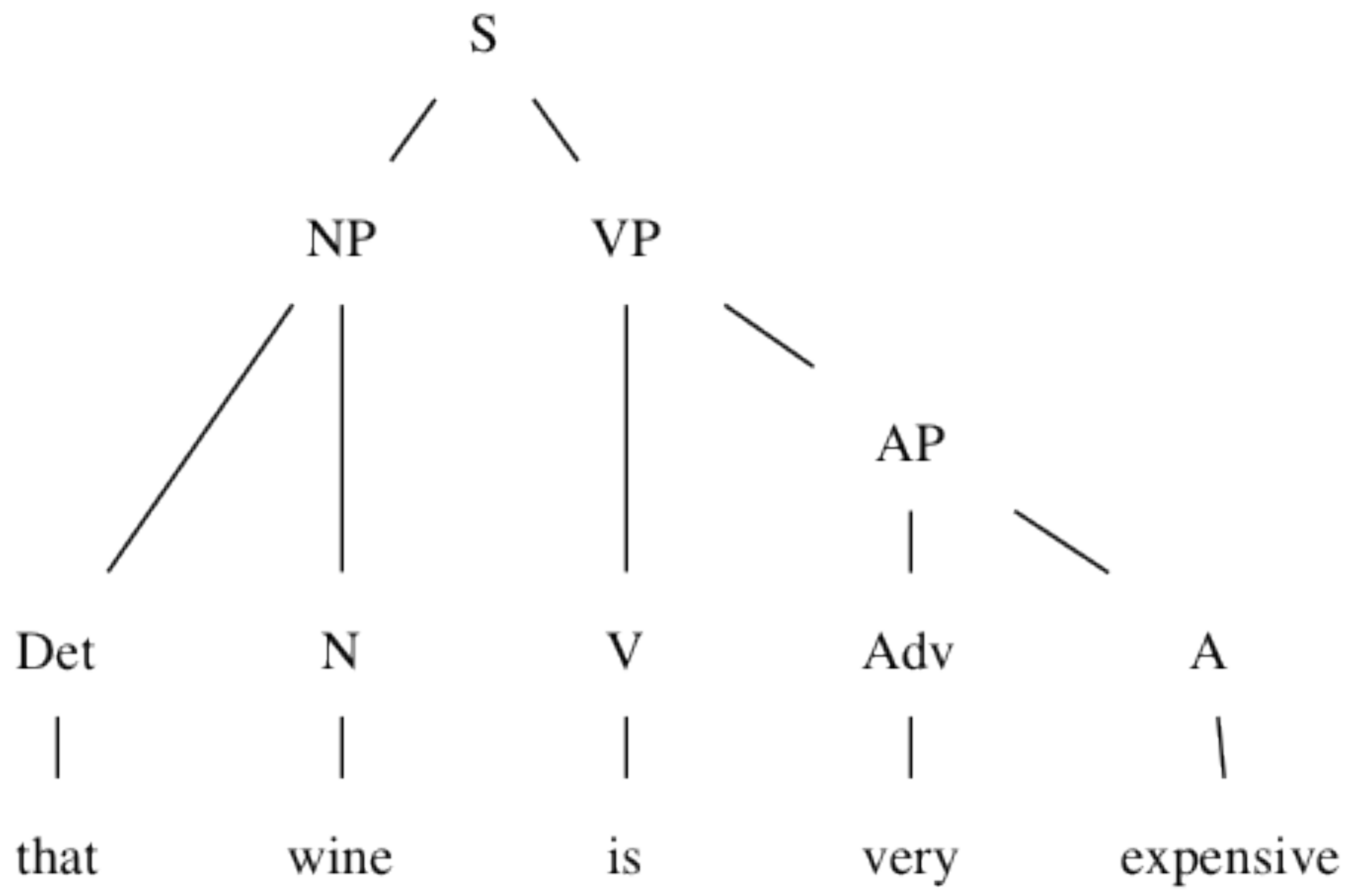


An example

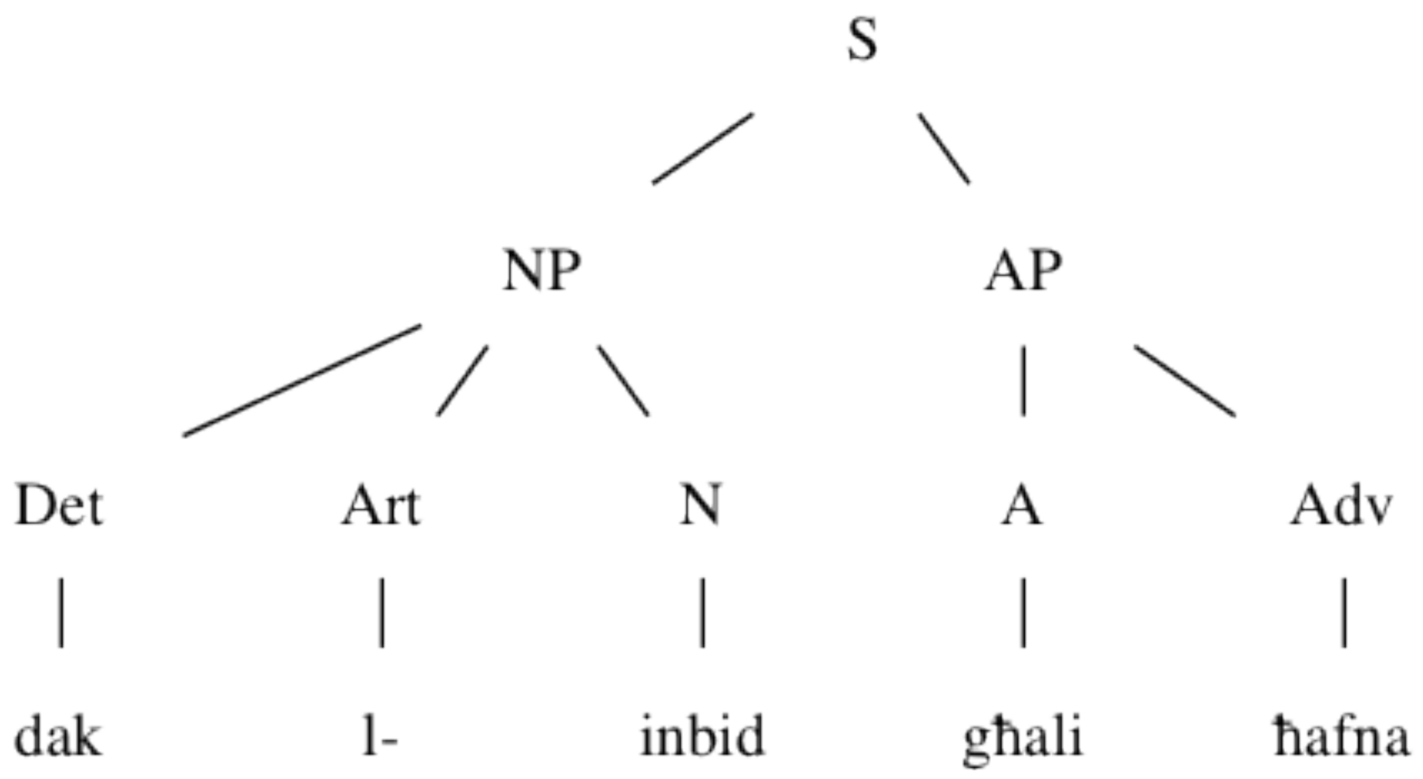
that wine is very expensive

dak l-inbid għali ħafna

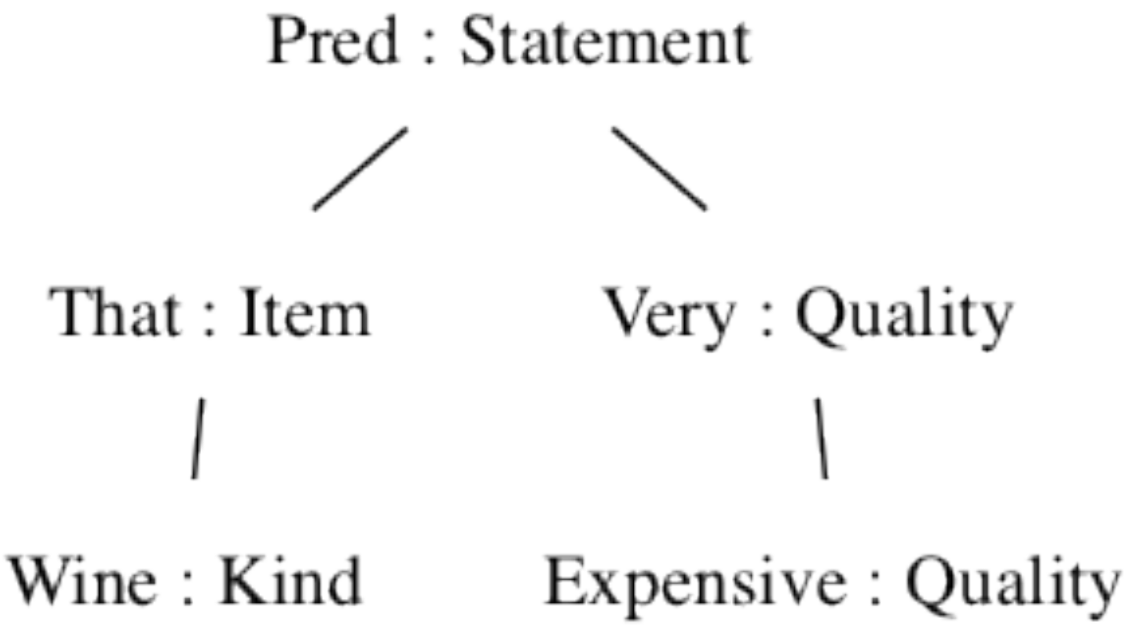
English parse tree



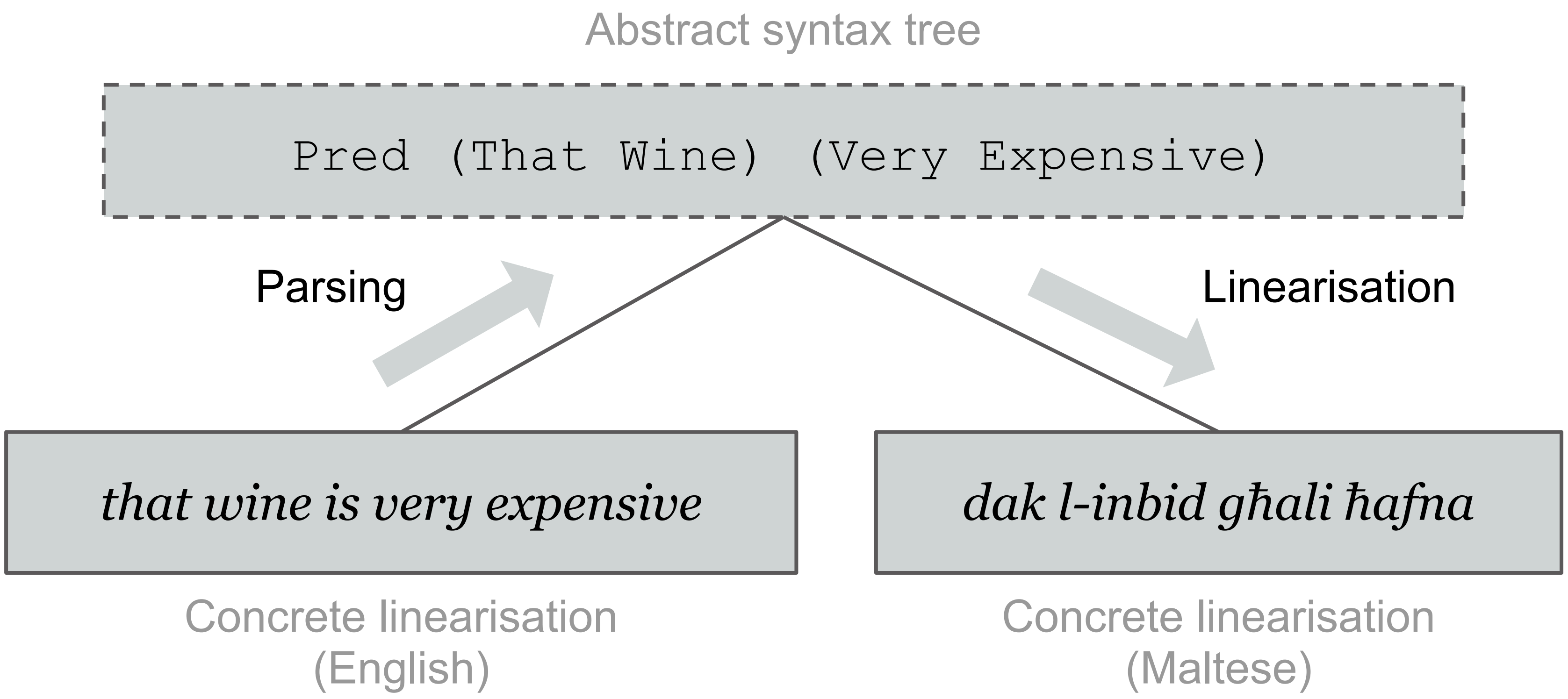
Maltese parse tree



Common abstract syntax tree



Parsing and linearisation


Abstract syntax tree
Pred (That Wine) (Very Expensive)
Parsing
Linearisation
that wine is very expensive
dak l-inbid għali ħafna
Concrete linearisation (English)
Concrete linearisation (Maltese)

- Same grammar for both directions
- Only one grammar per language (no pairs)

Demo

Let's try it out!

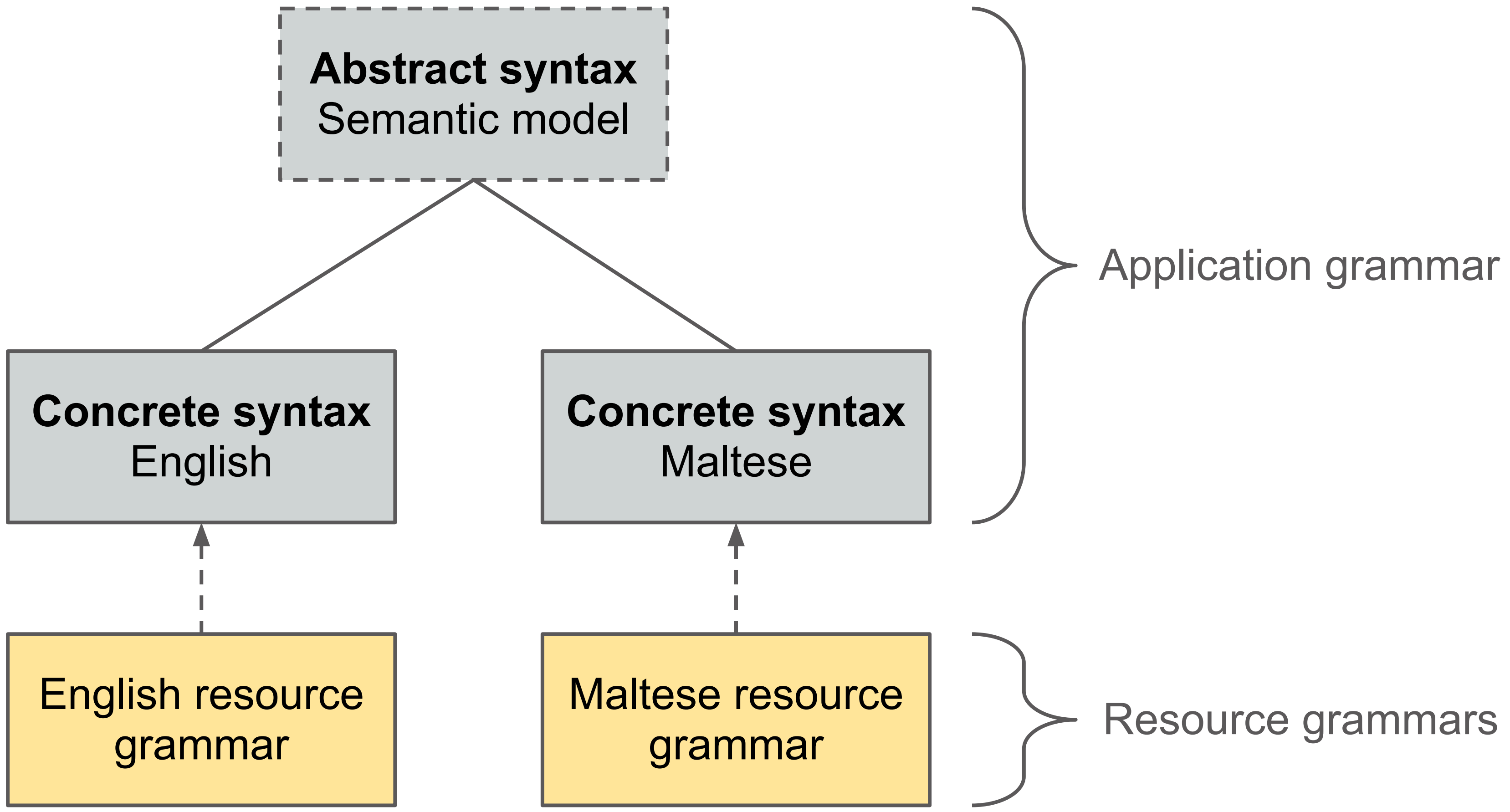
http://cloud.grammaticalframework.org/minibar/minibar.html

Grammars as libraries

- Software applications can use GF to power multilingual interfaces
- The low-level details of a language shouldn't be rewritten each time

- **Application grammars** are specific, focusing on semantic modelling
- **Resource grammars** are reusable, handling linguistic details of a each language

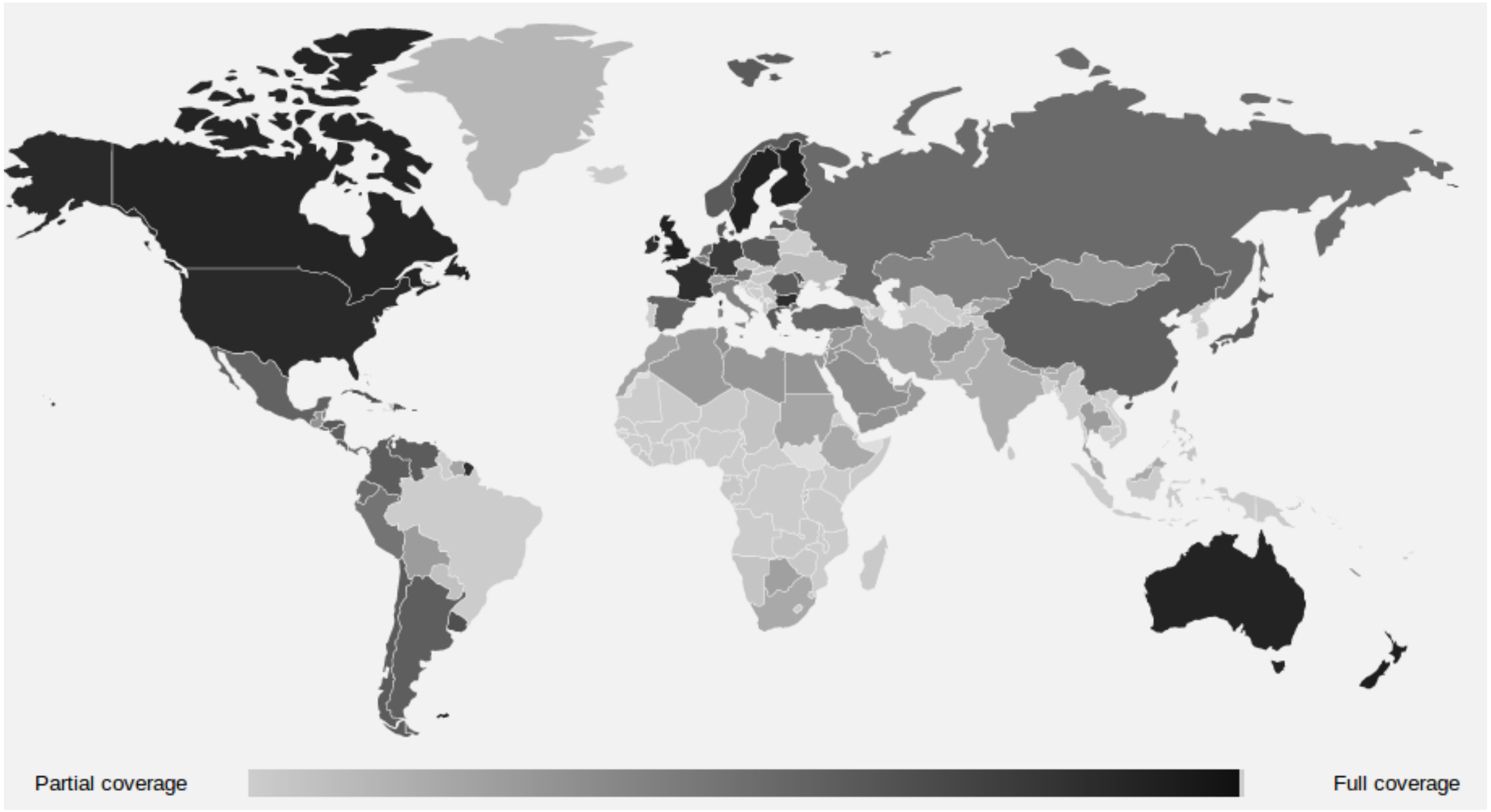
Application & resource grammars



GF Resource Grammar Library

- Implementations for 28 world languages:
  - English, Dutch, German
  - Danish, Swedish, Norwegian bokmål
  - Finnish, Latvian, Polish, Bulgarian, Russian
  - French, Italian, Romanian, Spanish, Catalan
  - Greek, Maltese, Interlingua
  - Chinese, Japanese, Thai
  - Hindi, Nepali, Persian, Punjabi, Sindhi, Urdu
- Single common interface, with optional language-specific extensions
- Open-source (LGPL/BSD licenses)

RGL map



A Maltese resource grammar

- Modules for
  - Morphology
    - Noun, verb, adjective, adverb
    - Structural words (prepositions, pronouns...)
  - Syntax
    - Noun, verb and adjective phrases
    - Numerals
    - Clauses, relative clauses, questions
    - Idiomatic constructions
  - Mini multilingual lexicon (300 entries)
  - Large-scale monolingual dictionary (in progress)

Paradigms

- Paradigm
  - A **function** which builds an inflection table for a lexical entry
- Smart paradigm
  - A paradigm function which requires only a lemmatised form to produce entire table
  - Gradual degradation in smartness until we reach a *worst-case* paradigm

Nouns

Linearisation table

```
fruit_N = {
   s Singulative = "frotta"
   s Collective  = "frott"
   s Dual        = ""
   s Plural      = "frottiet"
   gender    = Fem
   takesPron = False
}
```

Smart paradigm

```
fruit_N = mkN "frotta"
```

Verbs: inflection table

Linearisation table (fragments)

```
sleep_V = {
   s Perf P1 Sg         = "rqadt"
   s Perf P3 Sg Masc    = "raqad"
   s Impf P3 Sg Fem     = "torqod"
   s Impf P3 Pl         = "jorqdu"
   s Imp Sg             = "orqod"
   s PresPart Sg Masc = "rieqed"
   form     = FormI
   class    = Strong
   root     = "r-q-d"
   pattern = "a-a"
}
```

Verbs: paradigms

Smart paradigm (ideal case)

```
sleep_V = mkV "raqad"
```

Verbs: paradigms

Smart paradigm (ideal case)

```
sleep_V = mkV "raqad"
```

Other paradigms

```
mkV "dar" (mkRoot "d-w-r")
```

Verbs: paradigms

Smart paradigm (ideal case)

```
sleep_V = mkV "raqad"
```

Other paradigms

```
mkV "dar" (mkRoot "d-w-r")
mkV "ħareġ" "oħroġ" (mkRoot "ħ-r-ġ")
```

Verbs: paradigms

Smart paradigm (ideal case)

```
sleep_V = mkV "raqad"
```

Other paradigms

```
mkV "dar" (mkRoot "d-w-r")
mkV "ħareġ" "oħroġ" (mkRoot "ħ-r-ġ")
mkV form1 (mkRoot "ġ-j-'") (mkPatt "ie" [])
   "ġejt" "ġejt" "ġie" "ġiet" "ġejna" ...
   "niġi" "tiġi" "jiġi" "tiġi" "niġu" ...
   "ejja" "ejjew"
   "ġej"  "ġejja" "ġejjin"
```

Clauses

- Produces linearisation as a function of:
  - Tense (present, past, future, conditional)
  - Anteriority (simultaneous, anterior)
  - Polarity (positive, negative)

```
PredVP (UsePron (we_Pron)) (AdvVP (UseV (live_V)) (here_Adv))
```

```
s Pres Simul Pos = "ngħixu hawn"
s Pres Simul Neg = "ma ngħixux hawn"
s Past Simul Pos = "għexna hawn"
s Past Simul Neg = "m'għexniex hawn"
s Past Anter Pos = "konna għexna hawn"
s Past Anter Neg = "ma konniex għexna hawn"
s Fut  Simul Pos = "se ngħixu hawn"
s Fut  Simul Neg = "m'aħniex se ngħixu hawn"
s Fut  Anter Pos = "se nkunu għexna hawn"
s Fut  Anter Neg = "m'aħniex se nkunu għexna hawn"
s Cond Simul Pos = "konna ngħixu hawn"
s Cond Simul Neg = "ma konniex ngħixu hawn"
```

Limitations with the grammar

- In general, paradigms are not very smart
- Verb stem allomorphy is not perfect
- Pattern changes must often be explicit
- Participles must be added explicitly
- Free word order not handled
- Coverage unknown

Limitations with GF

- Restricted definition of word boundaries
  - articles, euphonic *i*, enclitic pronouns
- Unable to handle out-of-lexicon words, despite containing morphological rules
- In general cannot parse open text
- Computational limitations (next slide)

Computational limitations

- Ultimately the grammar must be tractable
- Size and compile-time considerations
    - ~3.5GB memory to import entire resource grammar
- Refactoring to please the compiler
    - Choosing less-natural representations
    - Throwing away information
    - Enclitic pronouns not treated as part of inflection table, harder to choose correct stem
    - Non-existent forms not efficiently supported
    - Avoiding exponential explosions in space and time

What's next?

- Use in application grammars
- Test morphological paradigms against corpus
- Monolingual lexicon
    - Semi-automatic extraction
- Use grammar to generate full-form lexicon

Access and use

- Released under the LGPL license
    - Can be used for any purpose, including commercial

- Stable release (part of GF):

http://www.grammaticalframework.org/download/

- Bleeding-edge source code and project page:

https://github.com/johnjcamilleri/Maltese-GF-Resource-Grammar

References, acknowledgements

- **Aarne Ranta**, *Grammatical Framework: Programming with Multilingual Grammars*, CSLI Publications, Stanford, 2011.
- **Aarne Ranta**, *The GF Resource Grammar Library*, Linguistic Issues in Language Technology, 2 (2), 2009.
- **John J. Camilleri**, *A Computational Grammar and Lexicon for Maltese*, M. Sc. thesis, Chalmers University of Technology, Gothenburg, 2013 (forthcoming).

Partly supported by the MOLTO project
From the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. FP7-ICT-247914
http://www.molto-project.eu/

Third GF Summer School 2013 *Scaling up Grammatical Resources*

18–30th August 2013

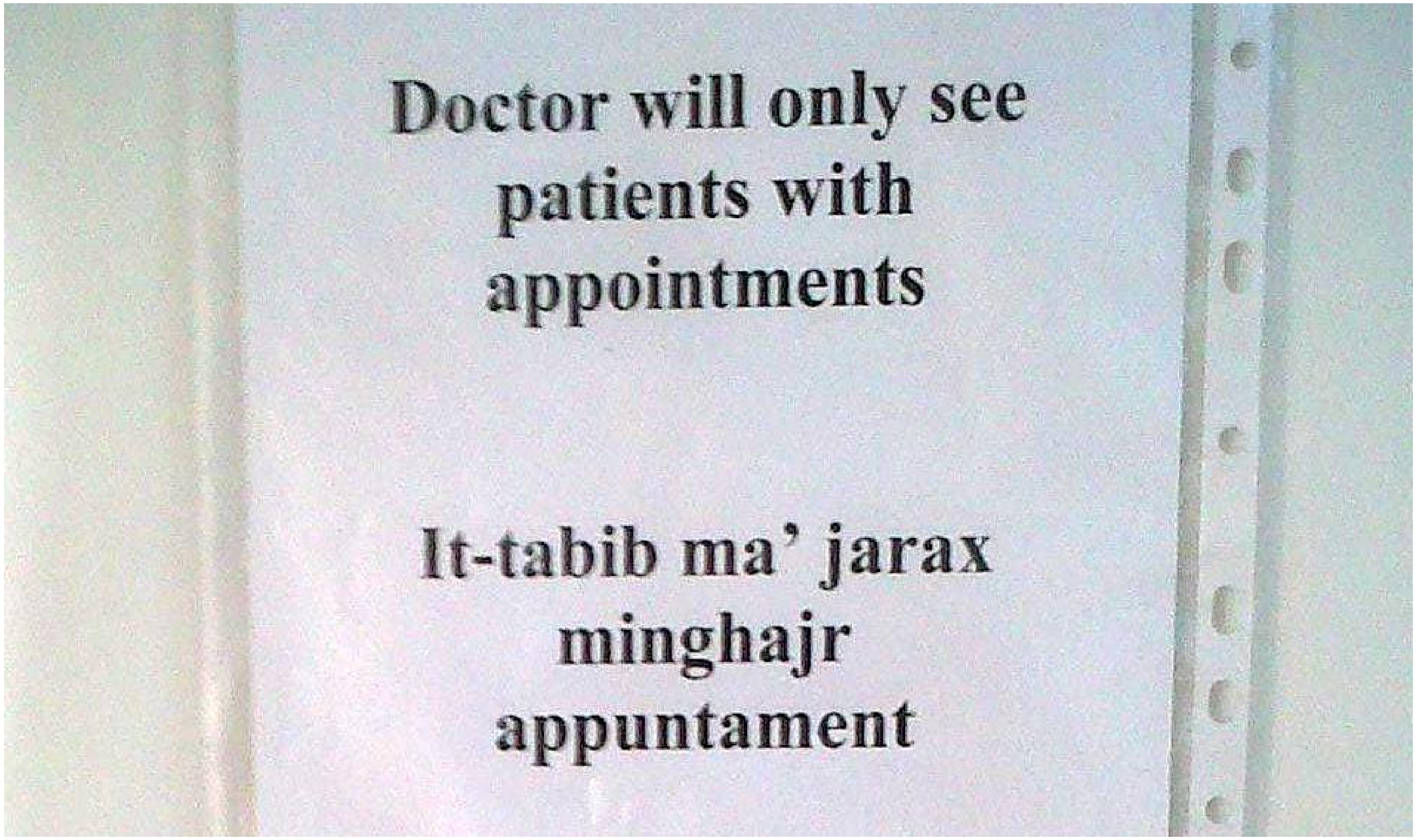
Frauenchiemsee Island, Bavaria

Week 1: Introduction to GF and multilingual grammar programming

Week 2: Advanced work in specialized tracks

http://school.grammaticalframework.org/

Thanks!


Doctor will only see patients with appointments

It-tabib ma' jarax minghajr appuntament

Extra slides...

RGL tense system

- Tense, anteriority, polarity (16 combinations)
- Mapped onto Maltese tenses as follows:

Temporal order	**Anteriority**	**Maltese equivalent**	**Example**
Present	Simultaneous	Imperfective	*jorqod*
Past	Simultaneous	Perfective	*raqad*
Future	Simultaneous	Prospective	*se jorqod*
Conditional	Simultaneous	Past Imperfective	*[kieku] kien jorqod*
Present	Anterior	Perfective	*raqad*
Past	Anterior	Past Perfect	*kien raqad*
Future	Anterior	Future Perfect	*se jkun raqad*
Conditional	Anterior	Past Prospective	*kien jorqod*

Example grammar: Foods

- Semantically model phrases about food
  - *“this fish is delicious”*
  - *“these cheeses are very expensive”*
- Linearise into multiple languages
- Parse multiple languages
- Single grammar for both directions!

Abstract syntax: Nouns

```
abstract Foods = {
  flags startcat = Comment ;
  cat
    Comment ; Item ; Kind ; Quality ;
  fun
    Pred : Item → Quality → Comment ;
    This, These : Kind → Item ;
    Cheese, Fish : Kind ;
    Very : Quality → Quality ;
    Expensive, Delicious : Quality ;
}
```

Abstract syntax: Quantifiers

```
abstract Foods = {
  flags startcat = Comment ;
  cat
    Comment ; Item ; Kind ; Quality ;
  fun
    Pred : Item → Quality → Comment ;
    This, These : Kind → Item ;
    Cheese, Fish : Kind ;
    Very : Quality → Quality ;
    Expensive, Delicious : Quality ;
}
```

Abstract syntax: Adjectives

```
abstract Foods = {
  flags startcat = Comment ;
  cat
    Comment ; Item ; Kind ; Quality ;
  fun
    Pred : Item → Quality → Comment ;
    This, These : Kind → Item ;
    Cheese, Fish : Kind ;
    Very : Quality → Quality ;
    Expensive, Delicious : Quality ;
}
```

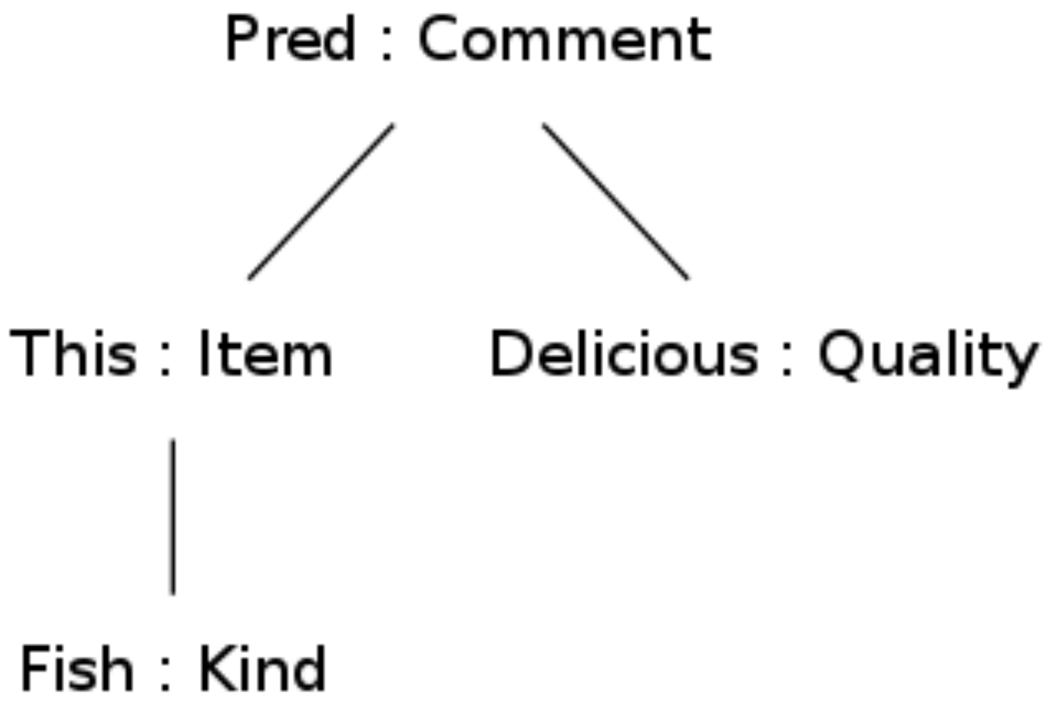
Abstract syntax: Very

```
abstract Foods = {
  flags startcat = Comment ;
  cat
    Comment ; Item ; Kind ; Quality ;
  fun
    Pred : Item → Quality → Comment ;
    This, These : Kind → Item ;
    Cheese, Fish : Kind ;
    Very : Quality → Quality ;
    Expensive, Delicious : Quality ;
}
```

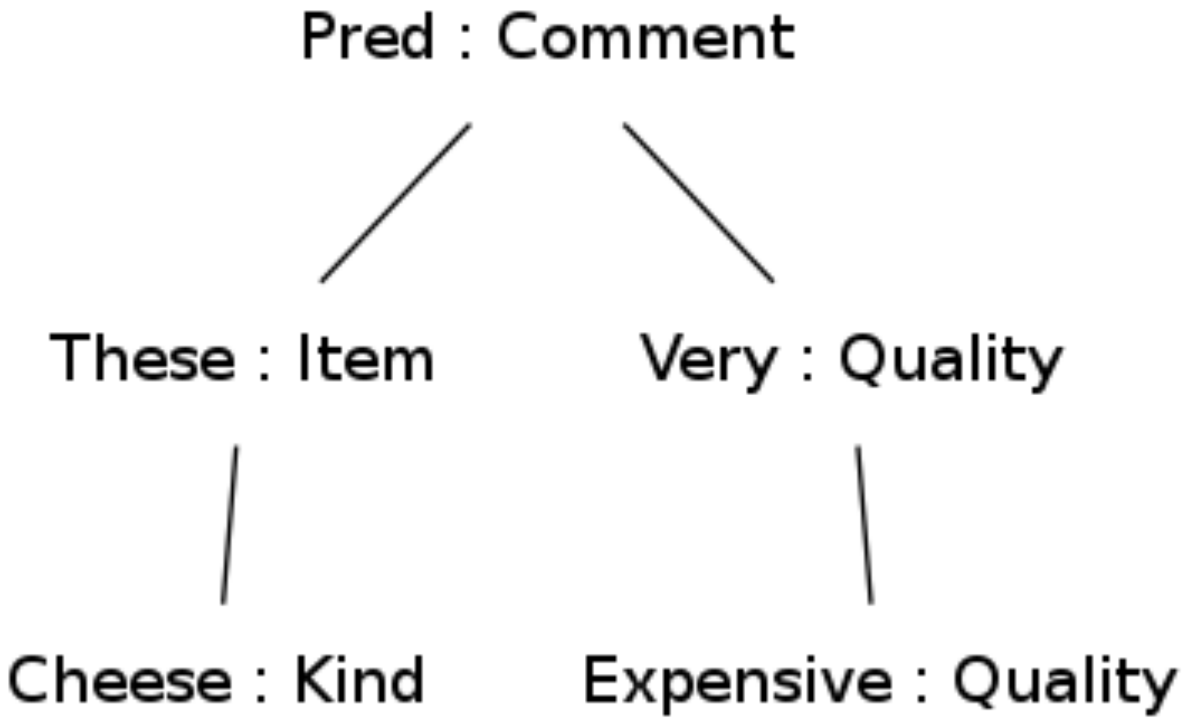
Abstract syntax: Predication

```
abstract Foods = {
  flags startcat = Comment ;
  cat
    Comment ; Item ; Kind ; Quality ;
  fun
    Pred : Item → Quality → Comment ;
    This, These : Kind → Item ;
    Cheese, Fish : Kind ;
    Very : Quality → Quality ;
    Expensive, Delicious : Quality ;
}
```

Abstract syntax tree (1)



Abstract syntax tree (2)



Concrete syntax: English

```
concrete FoodsEng of Foods = {
  lincat Kind   = { s : Number => Str } ;
  lin    Cheese = { s = table { Sg =>  "cheese" ; Pl => "cheeses" }};
         Fish   = { s = table { _  =>  "fish" }} ;

  lincat Quality   = { s : Str } ;
  lin    Expensive = { s = "expensive" } ;
         Delicious = { s = "delicious" } ;

  lincat Item    = { s : Str ; n : Number } ;
  lin    This  _ = { s = "this"  ; n = Sg } ;
         These _ = { s = "these" ; n = Pl } ;
  lin
    Pred item quality =
      {s = item.s ++ copula ! item.n ++ quality.s} ;
}
```

Concrete syntax: Maltese

```
concrete FoodsMlt of Foods = {
  lincat Kind    = { s : Number => Str ;  g : Gender } ;
  lin    Cheese = { s = table { Sg => "ġobna"; Pl =>"ġobniet" } ; g = Fem
};

  lincat Quality = { s :  Number => Gender => Str  } ;
  lin  Expensive = { s = table {
      Sg => table { Masc =>  "għali" ; Fem => "għalja" } ;
      Pl => table { _     =>  "għaljin" } } } ;

  lincat Item       = { s : Str ; n : Number ;  g : Gender } ;
  lin    This kind = { s = case kind.g of {Masc =>  "dan il-" ;
                                           Fem =>  "din il-" } ;
                    n = Sg ;  g = kind.g } ;
    Pred item quality =
      {s = item.s ++ copula ! item.n  ! item.g
                 ++ quality.s  ! item.n ! item.g} ;
}
```