A computational grammar and lexicon for Maltese

John J. Camilleri
M.Sc. Computer Science — ALL
Master's thesis defence · September 12, 2013

This thesis

- 60-point thesis
- Language Technology research group
- Part I presented at:
  4th International conference on Maltese Linguistics
  Lyon, France
  June 2013

Introduction

Malta

Maltese

- National language of Malta
- Official EU language since 2004
- 400k–1m speakers
- Semitic with Latin alphabet
- Heavily influenced by Romance, English
- Two kinds of morphology

Qiegħda waħdi nħares 'l isfel. 'Il fuq mis-sħab, 'il fuq mill-ħsibijiet tiegħi nnifsi. Qabadni l-għatx. Kienu għaddejin bil-kafejiet u tħajjart nixtri x'nixrob bl-għali fil-għoli fis-sema.

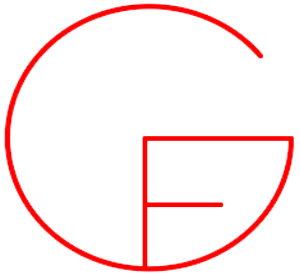
http://mariajdebono.blogspot.de/2013/07/qegdin-burgh-u-ajruplani.html

Computational grammars

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

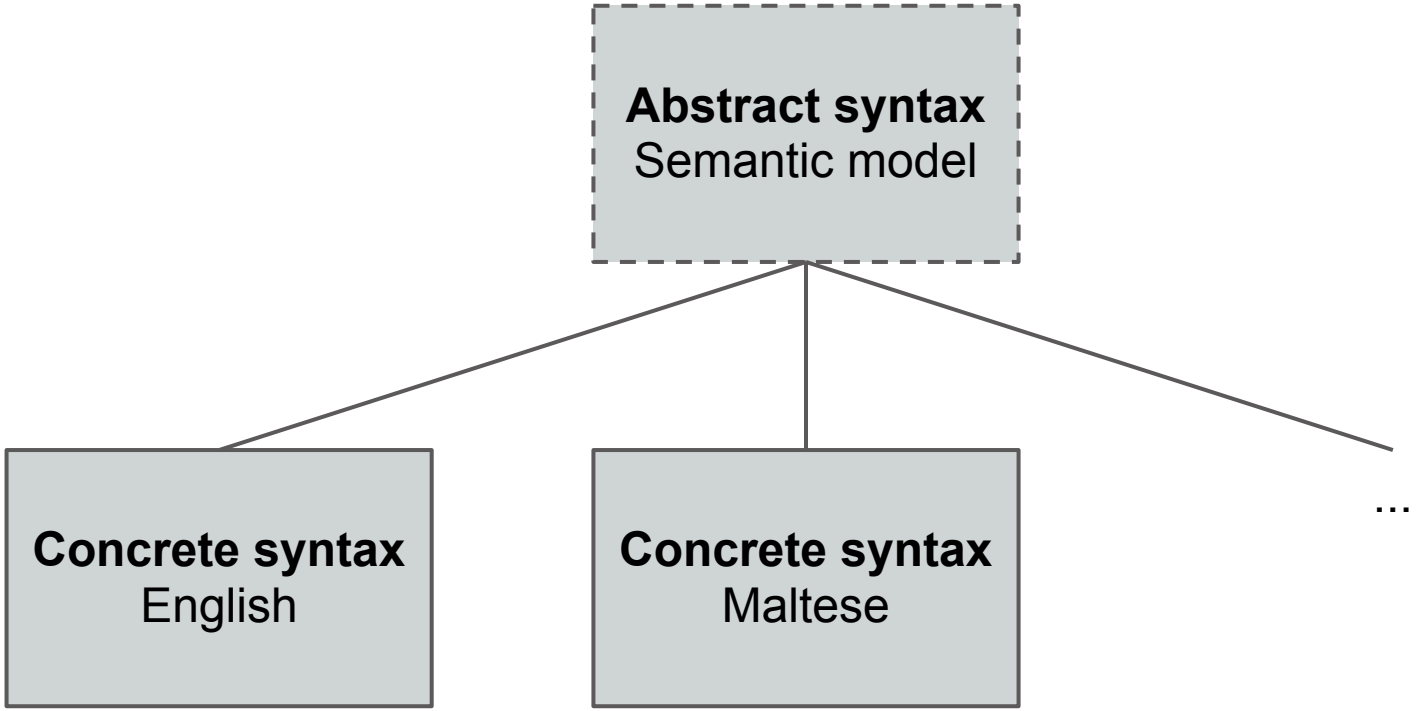
- Convert between surface input and abstract representation (e.g. parse trees)
- → Validate input phrases as in/correct
- ← Produce grammatically-correct phrases

Grammatical Framework



- Functional programming language for multilingual grammars
- Abstract syntax trees as a language-independent interlingua for modelling semantics
- Rule-based translation by combining parsing and generation

Abstract & concrete syntaxes

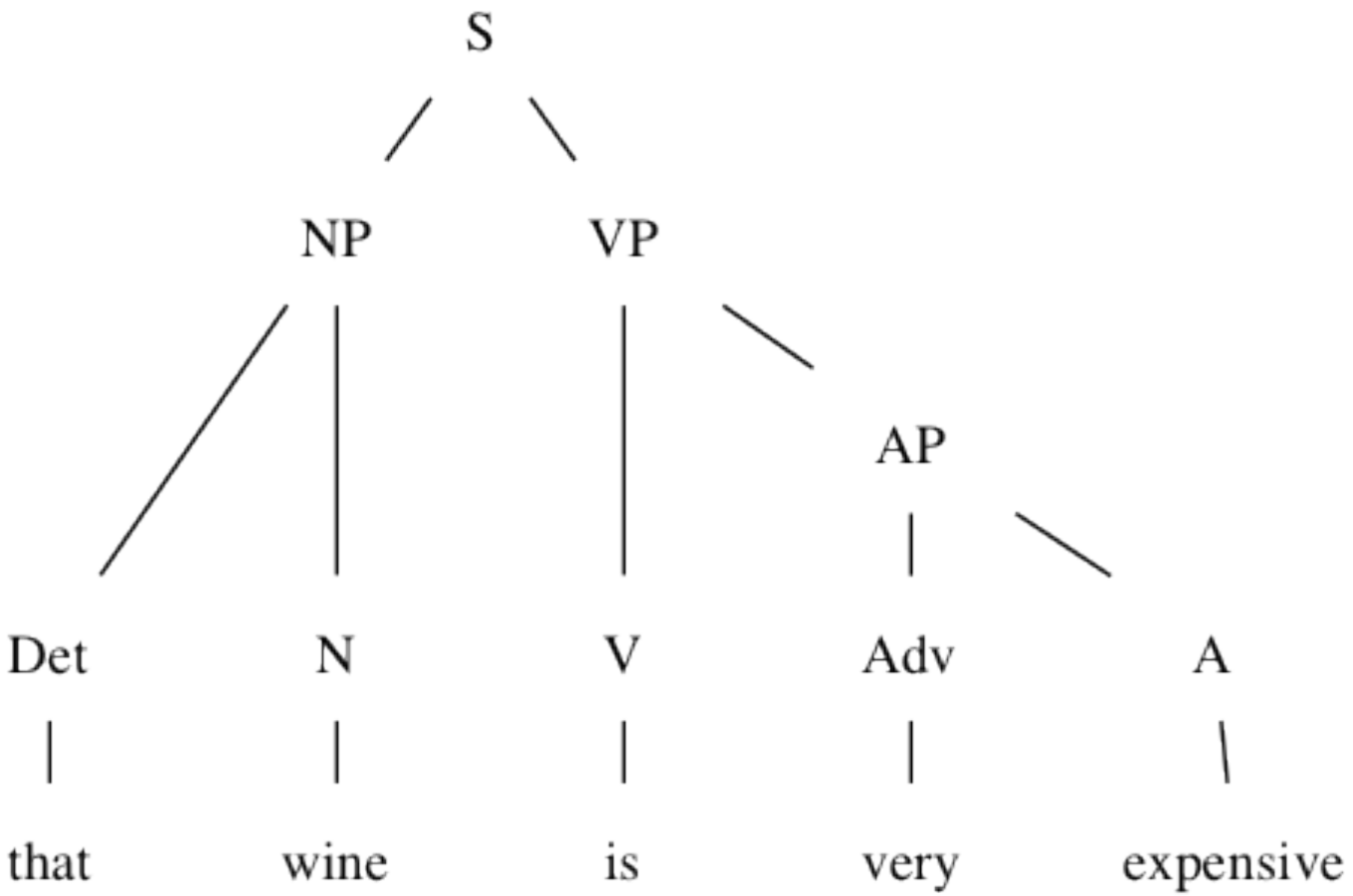


An example

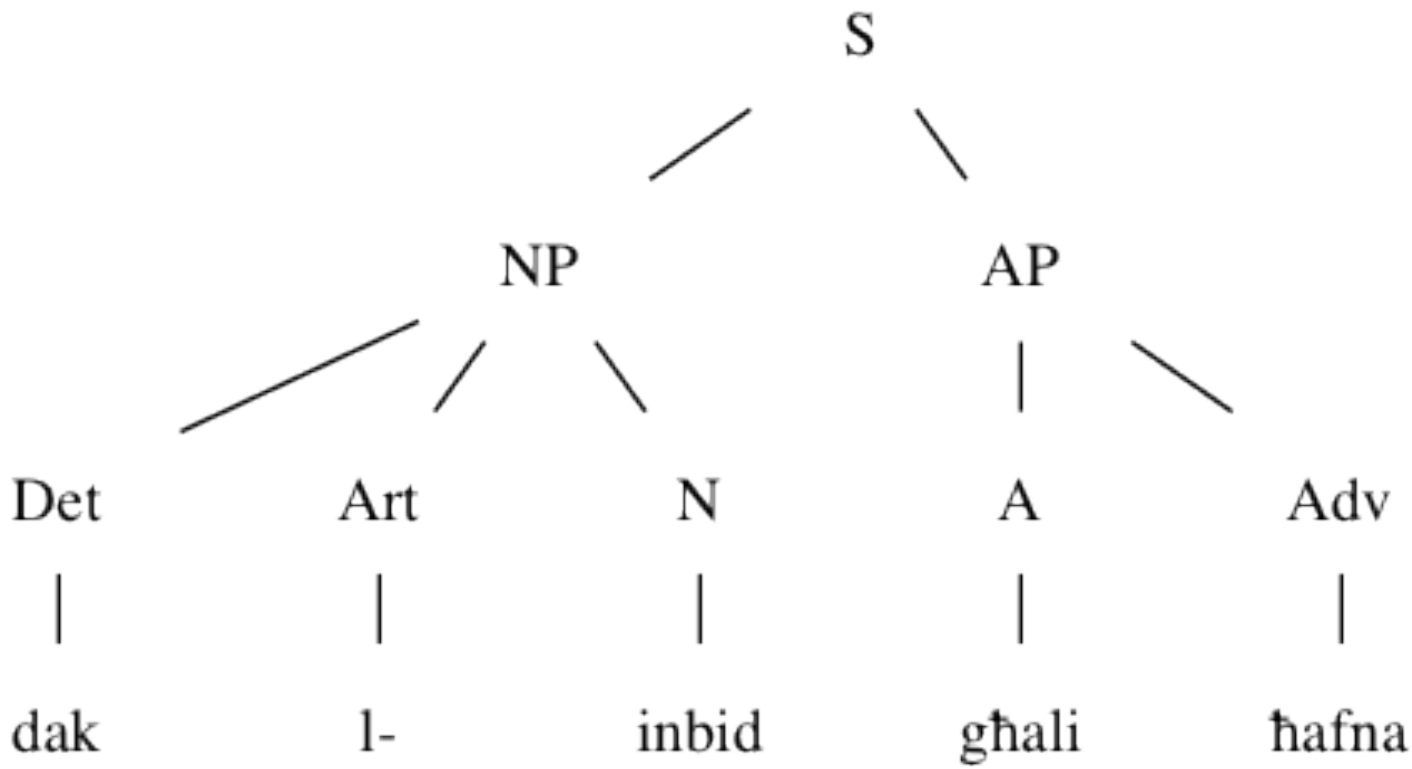
that wine is very expensive

dak l-inbid għali ħafna

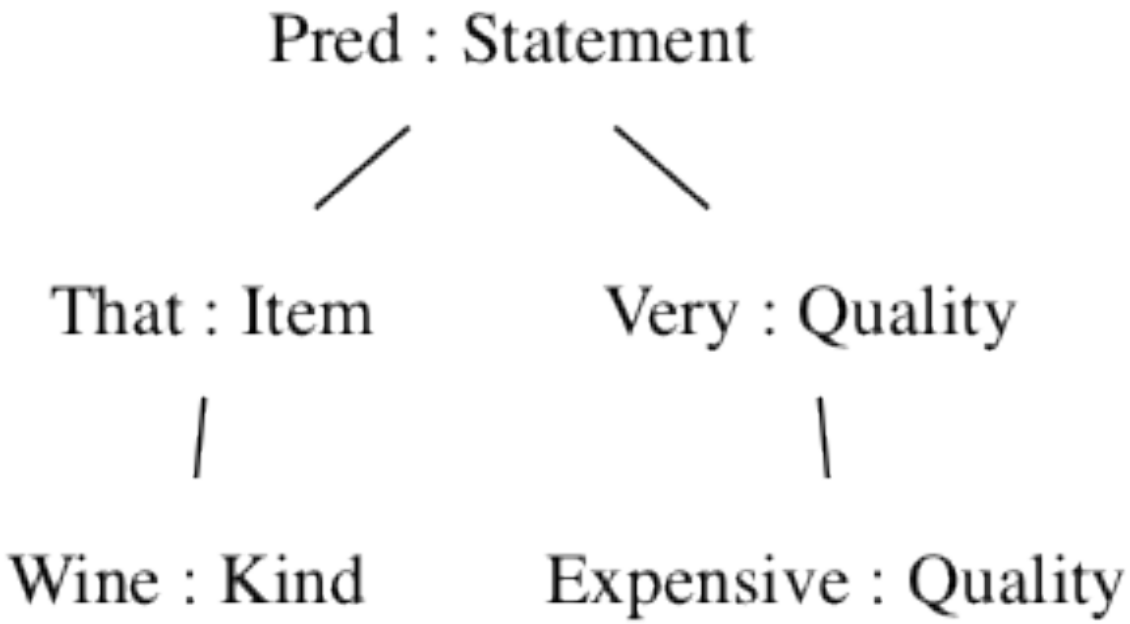
English parse tree



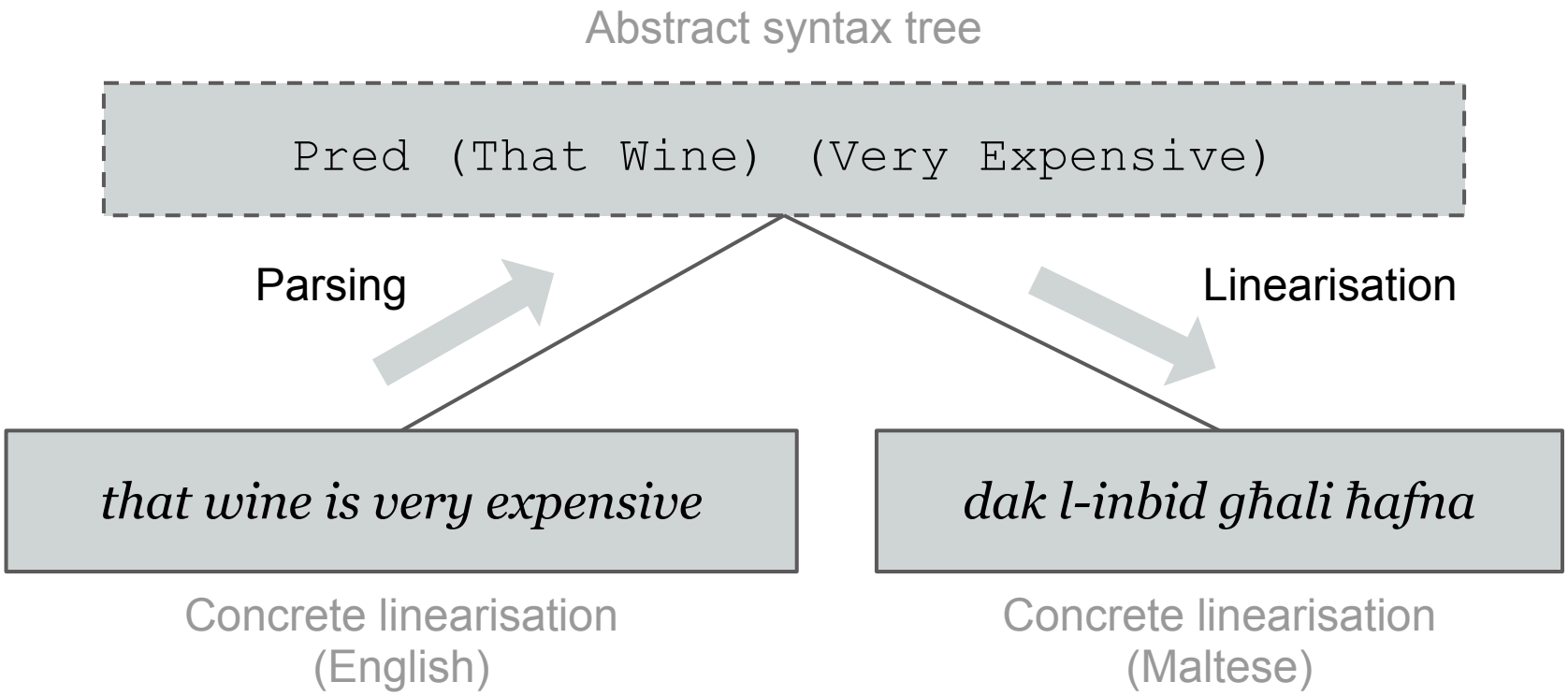
Maltese parse tree



Common abstract syntax tree



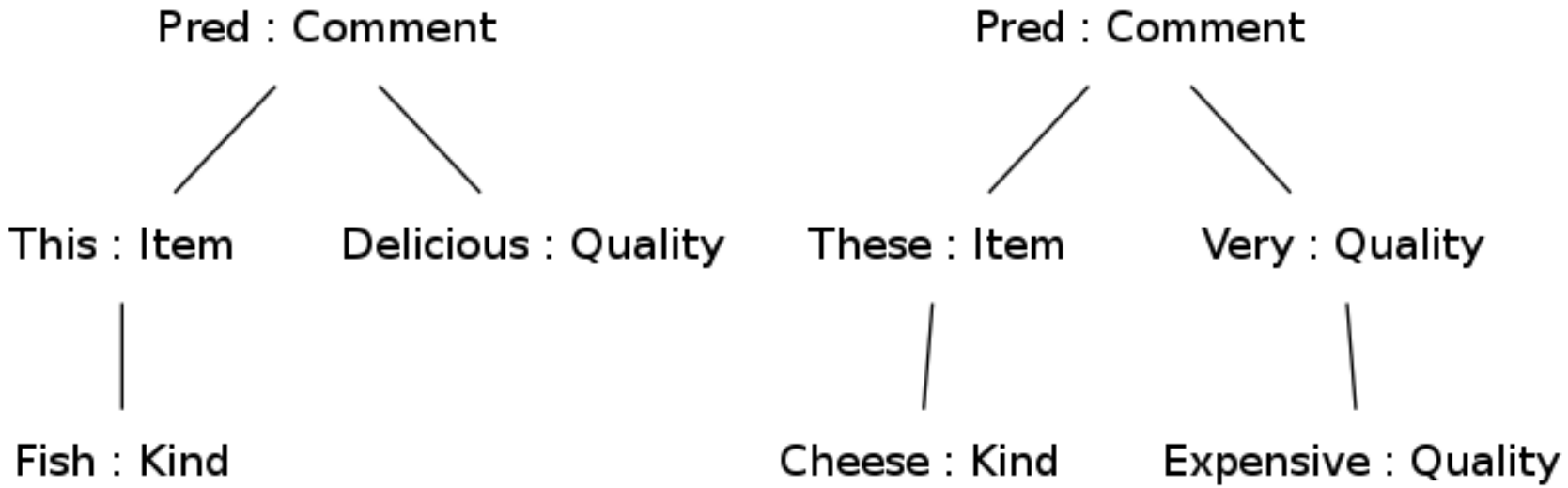
Parsing and linearisation



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

Example grammar: Foods

- Semantically model phrases about food
  - *“this fish is delicious”*
  - *“these cheeses are very expensive”*



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 ;
}
```

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

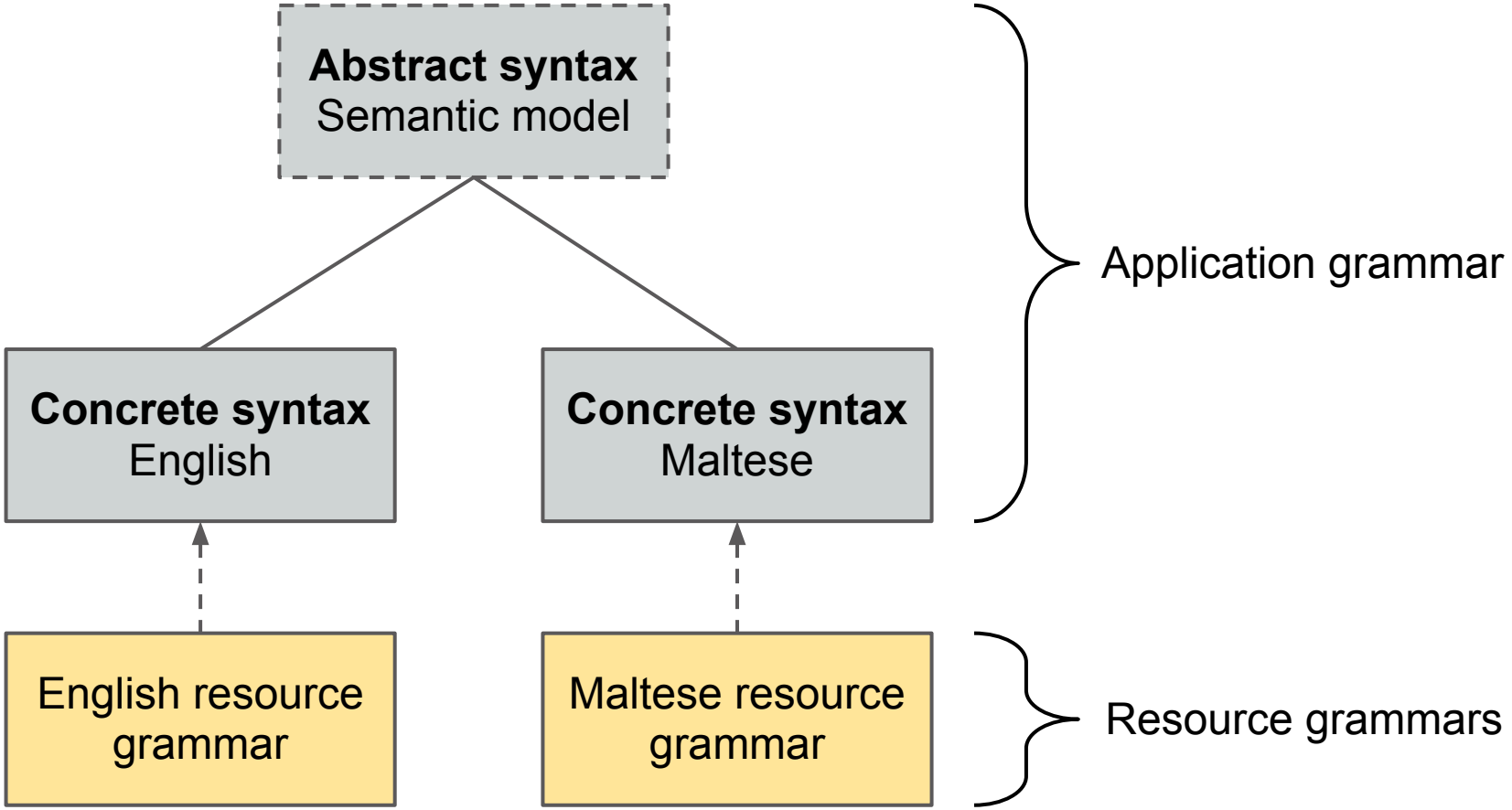
```
concrete FoodsMlt of Foods = {
  lincat Kind   = { s : Number => Str ;  g : Gender } ;
  lin    Cheese = { s = table { Sg => "ġobna"; Pl =>"ġobniet" } ; g = Fem };

  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} ;
}
```

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

Application & resource grammars



Part I
A computational grammar for Maltese

GF Resource Grammar Library

- Implementations for 28 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)

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)

Paradigms

- Paradigm
  - The inflection pattern which a word follows
  - A **function** which builds an inflection table for a lexical entry
- Smart paradigm
  - A paradigm function which only requires a small number of forms to produce entire table
  - Gradual degradation in smartness until we reach a *worst-case* paradigm

Verbs: lincat

Linearisation type (simplified)

```
Verb : Type = {
  s : VForm => Str ;
  i : VerbInfo ;
  hasPresPart : Bool ;
  hasPastPart : Bool ;
} ;

VForm =
    VPerf VAgr | VImpf VAgr | VImp Number
  | VPresPart GenNum
  | VPastPart GenNum ;
```

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"
   i form    = FormI
   i class   = Strong
   i root    = { c1="r" ; c2="q" ; c3="d" }
   i vseq    = { v1="a" ; v2="a" }
}
```

Verbs: paradigms

Smart paradigm (ideal case)

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

Verbs: paradigms

Smart paradigm (ideal case)

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

Graceful degradation

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

Verbs: paradigms

Smart paradigm (ideal case)

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

Graceful degradation

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

Verbs: paradigms

Smart paradigm (ideal case)

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

Graceful degradation

```
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 I

Linearisation is a function of:

- Tense (present, past, future, conditional)
- Anteriority (simultaneous, anterior)
- Polarity (positive, negative)

```
Clause : Type = {
  s : Tense => Anteriority => Polarity => Str
} ;
```

Clauses II

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

Limitations

- Bugs with enclitic pronouns and Sandhi (stem/affix changes)
- Free word order not handled
- Word boundary phenomena
- Refactoring to please the compiler
  - 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
- Cannot parse without a separate lexer

Part II
Towards a computational lexicon for Maltese

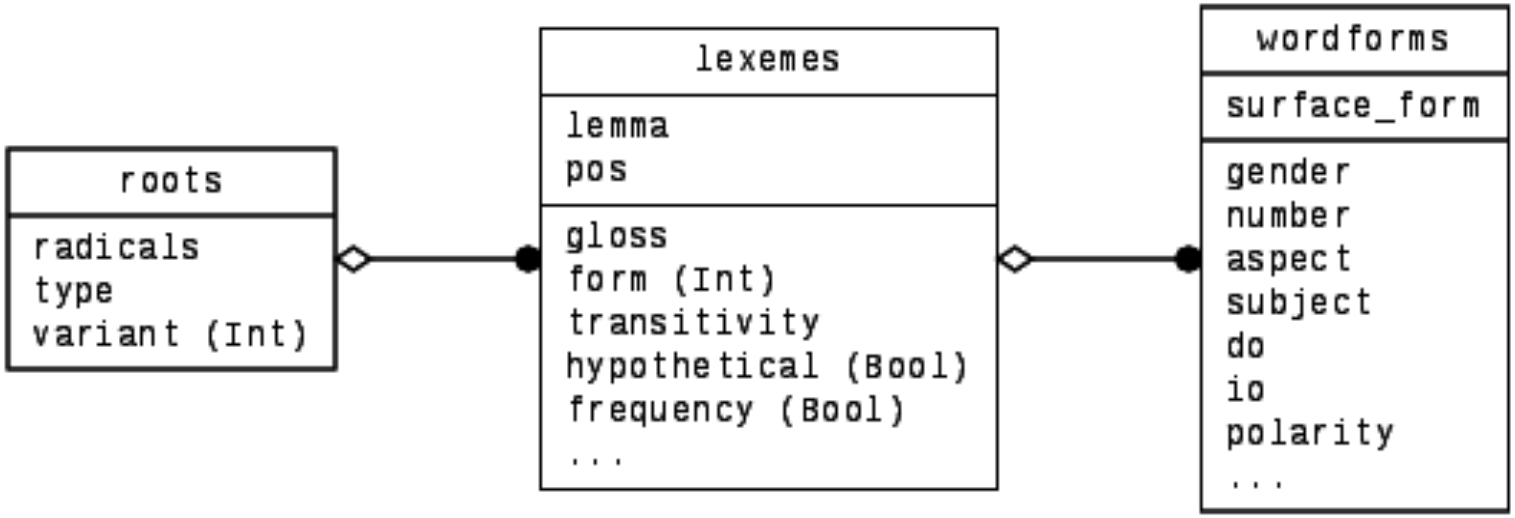
Various heterogeneous resources

1. Verbal roots and patterns
   - 1923 roots
   - 4,142 root-and-pattern verbs
   - MySQL database
2. Corpus of broken plurals
   - 654 plurals in TSV
3. List of verbal nouns
   - Over 2000 entries in a Microsoft Word table
4. Basic English-Maltese dictionary
   - 5,454 English entries in XML

Collect them all together in a single database

A flexible database schema

- MongoDB
  - No SQL
  - JSON-style documents with flexible schemas
  - No joins!
- Importation Haskell script for each source



Example from `lexemes` collection

```
{
    "_id"     : ObjectId("5200a366e36f2379750007b6"),
    "lemma"   : "tbarważ",
    "pos"     : "V",
    "root"    : {
        "radicals" : "b-r-w-ż"
    },
    "form"    : 2,
    "source"  : "Spagnol2011"
},
{
    "_id"     : ObjectId("5200a368e36f237988000006"),
    "lemma"   : "skarpan",
    "pos"     : "N",
    "gloss"   : "shoemaker",
    "gender"  : "m",
    "source"  : "Mayer2013"
}
```

Web application

Ġabra: an opportunistic collection of Maltese linguistics resources

- Written using CakePHP framework
- Browsing & search interface to DB
- Lemmas and full inflectional forms (generated or otherwise)
- User feedback: mark forms as incorrect
- Web service

Ġabra: an opportunistic collection of Maltese linguistics resources

English Malti

Roots Lexemes

^ħ-.-ġ$ Search gloss Search Clear

Radicals ħ – – ġ – none Class Root search

‹ previous next › Page 1 of 1, showing records 1 to 11 out of 11 total.

Radicals	Type	I	II	III	V	VI	VII	VIII	IX	X
ħ-j-ġ (ħ-w-ġ)	Tri. weak-medial						*nħtieġ*	*ħtieġ*		
ħ-l-ġ	Tri. strong	*ħaleġ*	*ħalleġ*				*nħaleġ*	*ħtileġ*		
ħ-m-ġ	Tri. strong		*ħammeġ*		*tħammeġ*					
ħ-r-ġ	Tri. strong	*ħareġ*	*ħarreġ*		*tħarreġ*		*nħareġ*			*stħarreġ* (*staħreġ*)
ħ-ġ-ġ [1]	Tri. geminated	*ħaġġ*							*ħġaġ*	
ħ-ġ-ġ [2]	Tri. geminated		*ħaġġeġ*							
ħ-ġ-ġ [3]	Tri. geminated		*ħeġġeġ* (*ħaġġeġ*)		*tħeġġeġ* (*tħaġġeġ*)					

‹ previous next › Page 1 of 1, showing records 1 to 11 out of 11 total.

Random entry

kiteb

POS	V
Gloss	1. write 2. recruit 3. register 4. feed a computer
Root	**k-t-b**
Features	Form 1 common trans.
Source	Spagnol2011

Word forms

Aspect	Subject	Direct object	Indirect object	Polarity	Surface form
Perf	P3 Sg Masc			Pos	*kiteb*
Perf	P3 Sg Masc			Neg	*kitebx*
Perf	P3 Sg Masc		P1 Sg	Pos	*kitebli*
Perf	P3 Sg Masc		P1 Sg	Neg	*kiteblix*
Perf	P3 Sg Masc		P2 Sg	Pos	*kiteblek*
Perf	P3 Sg Masc		P2 Sg	Neg	*kiteblekx*
Perf	P3 Sg Masc		P3 Sg Masc	Pos	*kiteblu*
Perf	P3 Sg Masc		P3 Sg Masc	Neg	*kiteblux*
Perf	P3 Sg Masc		P3 Sg Fem	Pos	*kitebilha*

Full-forms

- Dictionaries only give partial information

ħtieġa *n.f.s., pl. -t, -ijiet* bżonn; neċessità; siwi;

- What are the plural forms?
  - Wrong: *ħtieġat*, *ħtieġaijiet*
  - Wrong: *ħtieġiet*, *ħtieġijiet*
  - Correct: *ħtiġiet*, *ħtiġijiet*
  - Multiple implicit rules at play
- Storing full forms makes things explicit
- Required for lookup, e.g. spell-checking
- Easy to handle exceptions

Generating full-forms: step 1

- Lemma list → monolingual GF dictionary module `DictMlt`
- For each, generate GF identifier and use smart paradigm with available info

```
abstract DictMltAbs = Cat ** {
    ...
    fun rikeb_RKB_1_V : V ;
    ...
}
concrete DictMlt of DictMltAbs = CatMlt **
  open ParadigmsMlt in {
    ...
    lin rikeb_RKB_1_V = mkV "rikeb" (mkRoot "r-k-b") ;
    ...
}
```

Generating full-forms: step 2

- Use GF's linearise command

Adjective	`<A>`
Noun	`DetCN (DetQuant (PossPron <DO>) NumSg) <N>`

Verb	`UseCl`
	`  <Tense>`
	`  <Pol>`
	`  (PredVP`
	`  (UsePron <Subj>)`
	`  ComplSlash (SlashVa <V>) (UsePron <DO>)`
	`  )`

- Store back in DB collection `wordforms`

Numbers

Roots	1928
Lexemes	13,783
Wordforms	4774186
Sources	5

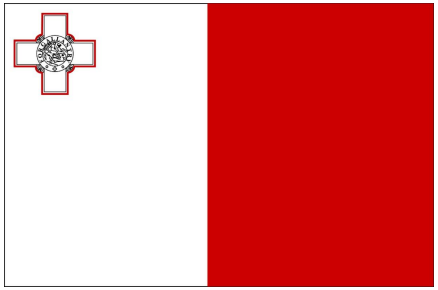
- Comparison
  - Serracino-Inglott (2003): ~26,000 entries
  - Aquilina dictionary (1987-1990): ~80,000 entries
- Soon: access to digitised versions of the above

Finally,

Access and use

- Resource grammar
  - LGPL license
  - Stable release (part of GF): http://www.grammaticalframework.org/download/
  - Bleeding-edge source code: https://github.com/johnjcamilleri/Maltese-GF-Resource-Grammar
- Lexicon
  - CC-BY license (contents)
  - Web app: http://mlrs.research.um.edu.mt/resources/gabra/

Acknowledgements



The research work disclosed in this publication is funded by the Strategic Educational Pathways Scholarship (Malta). The scholarship is part-financed by the European Union — European Social Fund (ESF) under Operational Programme II — Cohesion Policy 2007-2013, “Empowering People for More Jobs and a Better Quality of Life”.

MOLTO

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/


Don’t guess if you know.

Overflow...

Nouns: lincat

Linearisation type

```
Noun : Type = {
   s : Noun_Number => Str ;
   g : Gender ;
   hasColl : Bool ;
   hasDual : Bool ;
   takesPron : Bool ;
   } ;
Noun_Number =
     Singulative
   | Collective
   | Dual
   | Plural ;
```

Nouns: lin

Linearisation table

```
ear_N = {
   s Singulative = "widna"
   s Collective  = ""
   s Dual        = "widnejn"
   s Plural      = "widniet"
   g = Fem
   hasColl = False
   hasDual = True
   takesPron = False
}
```

Smart paradigm

```
ear_N = mkNDual "widna"
```

Enclitic pronouns I

Perf. P1 Pl. *fetaħ* ‘he opened’

Direct Object	Indirect Object	Positive	Negative
-	-	*ftaħna*	*ftaħniex*
P3 Sg Masc	-	*ftaħnieh*	*ftaħnihx*
-	P3 Pl	*ftaħnilhom*	*ftaħnilhomx*
P3 Sg Masc	P3 Pl	*ftaħnihulom*	*ftaħnihulhomx*

- 952 combinations! But only 3 stems:
  - *ftaħna*
  - *ftaħnie*
  - *ftaħni*

Enclitic pronouns II

Store only stems in inflection table

```
Verb : Type = {
   s : VForm => VerbStems ;
   i : VerbInfo ;
   hasPresPart : Bool ;
   hasPastPart : Bool ;
} ;

VerbStems : Type = {s1, s2, s3 : Str} ;
```

Enclitic pronouns III

Join enclitic pronouns at syntax level

```
dirobj : Verb -> Agr -> Pronoun -> Str
dirobj v agr pron =
            (v.s ! Perf agr).s2
            ++ BIND
            ++ pron.s ! Suffixed
```

Resulting token list

```
["ftaħnie", "&+", "h"]
```

After unlexing

```
ftaħnieh
```

Word boundaries

- English
  - a house, an airplane
- Maltese pre-change
  - *il-knisja* (‘the church’)
  - *id-dar* (‘the house’)
  - *l-iskola* (‘the school’)
- Maltese post-change
  - *hu jmur* (‘he goes’)
  - *kien imur* (‘he used to go’)
  - This is impossible in GF!

Treebank results

Treebank	Passed	Total	Percentage
articles	5	5	100.0
exx-resource	111	186	59.7
n-clitics	35	49	71.4
numerals-np	32	32	100.0
numerals-simple	52	63	82.5
phrases	19	22	86.4
prep	24	24	100.0
v-clitics-past	336	392	85.7
v-clitics-pres	368	392	93.9
vp	120	128	93.8