# Controlled Language and Machine Translation by GF: a brief introduction

K. V. S. Prasad

Dept of Computer Science

Chalmers University

# Main Sources for talk

- ”Implementing Controlled Languages in GF”
  - Krasimir Angelov and Aarne Ranta
- ”Controlled Language for Everyday Use”
  - Aarne Ranta, Ramona Enache, Gregoire Detrez
- ”The GF Mathematics Library”
  - Jordi Saludes and Sebastian Xambo
- ”Grammatical Framework: Formalizing the Grammars of the World” (slides of talk)
  - Aarne Ranta
- See http://www.grammaticalframework.org/

# Our view of Controlled Language (CL)

- A CL is a natural language (NL)
  - but with restricted (formal) syntax and semantics
    - to reduce ambiguity
    - help human understanding,
    - enable mechanical processing
- Compared with a formal notation, such as maths or programming languages
  - You can understand CL without special training
  - Provided you know the NL the CL came from

# Target application of CL in GF

- Technical documentation
  - from  a formal knowledge base
    - Mute(Loud) -> Soft; Mute(Soft) -> Loud
      - Might describe a toggle button on a phone
    - Loud_b -> Loud; Soft_b -> Soft
      - Separate buttons for loudspeaker on/off
  - mechanically generated CL (Eng., Hin., …)
    - ”When the loudspeaker is off, MUTE turns it on”
    - जब लाउडस्पीकर बंद है, MUTE उसे चलाता है

# Classification of CL's

- Huijsen classifies CL's into
  - Human-Oriented CL
    - To improve readability and comprehensibilty
  - Machine-Oriented CL
    - To improve translatability
- Then a GF CL is an MOCL
  - promising potential application:
    - meaning-preserving high-quality automatic translation.
- O'Brien analyses several CL's for English
  - Only rule in common: encouraging brief sentences
  - GF has no such restriction, only formal grammar

# What is GF?

- GF provides
  - a high-level grammar formalism
  - A programming language for grammars
  - a resource grammar library (RGL)
  - Together, these help write grammars that cover
    - similar fragments in several natural languages
    - at the same time.
  - As the grammars are formal, they are amenable to computer processing. In particular, translation.

# Why ”Framework”?

- Example: to formalise mathematics
  - We can do it all in one formal system, say ZF
  - Or use a logical framework
    - To define a logic for the task at hand
    - Different logics for different bits of mathematics
- Controlled language design in GF
  - You decide what semantics you need
  - Express it in the GF language

# GF abstract and concrete syntax

- The semantic model in GF
  - ”abstract syntax”
    - Give the signature for a many-sorted algebra
    - Elements of this algebra usually shown as trees
  - Not necessarily given any other semantics
    - But can be done, e.g., operational semantics
- Syntactic realization: ”concrete syntax”
  - Interpretation of signature in a world of strings
- Morphisms
  - From semantics to syntax: ”linearisation” (generation)
  - From syntax to semantics: parsing

# Example

- Categories: problem, prop, and exp
  - Functions to construct abstract syntax trees
    - Prove (Even (Sqrt (EInt 36)))
    - Means ”prove that the square root of 36 is even”
- GF trees are statically typed
  - type checker verifies the above tree is well-typed

# An abstract syntax for math problems

```
abstract Math =
        {cat   Problem ; Prop ; Exp ;
         fun
               Prove : Prop → Problem ;
               Compute : Exp → Problem ;
               Even : Exp → Prop ;
               Div : Exp → Exp → Prop ;
               Sqrt : Exp → Exp ;
               EInt : Int → Exp ;   }
```

# Eng. concrete syntax: math problems

```
concrete MathEng of Math =
        {lincat
                Problem , Prop , Exp = Str ;
        lin
                Prove p = ”prove that” + p ;
                Compute e = ”compute” + e ;
                Even x = x + ”is even”;
                Div x y = x + ”is divisible by” + y ;
                Sqrt x = ”the square root of” + x;
                EInt x = x ;}
```

# French syntax?

- A concrete syntax for French
  - Multi-lingual syntax for same abstract sentence
  - Translate between English and French
- But French has
  - Gender, and agreement
  - Indicative and subjunctive moods
- Captured by parameters in the concrete syntax, without affecting abstract syntax

# Fre. concrete syntax: mood, gender

```
concrete MathFre of Math = {
    param
        Mood = Ind | Subj ;        Gender = Masc | Fem ;
    lincat
        Prop = Mood ⇒ Str ;
        Exp = {s : Str ; g : Gender };
    lin
        Even x = table {Ind ⇒ x .s + ”est” + pair;
                        Subj ⇒ x .s + ”soit” + pair}
                 where {pair = case x .g of {Masc ⇒ ”pair”;
                                             Fem ⇒ ”paire”}};}
```

# Resource Grammar Libraries

- But that was a fair amount of French!
  - For a small *mathematics* (application) CL
  - What about the next CL? French again?
- Put the French in a resource grammar library (RGL)
  - RGL based on linguistic concepts
  - independent of domain-specific semantics
  - implemented by linguists, to be used by nonlinguists.
- RGL too has a common abstract syntax.
  - Contains common syntactic structures
  - Example, predication function,
    - fun PredVP : NP → VP → Cl

# A library-based French concrete syntax for mathematical problems

```
concrete MathFre of Math = open SyntaxFre, ParadigmsFre in {
    lincat
        Problem = Utt ;   Prop = Cl ;  Exp = NP ;
    lin
        Prove p = mkUtt (mkVP (mkVS ”de´ montrer”) (mkS p));
        Compute e = mkUtt (mkVP (mkV2 ”calculer”) e );
        Even x = mkCl x (mkA ”pair”);
        Div x y = mkCl x (mkA2 ”divisible” ”par”) y ;
        Sqrt x = mkNP defArt (mkCN (mkCN (mkN ”racine”)
                                   (mkA ”carre´ ” (mkAdv de Prep x )));
        EInt x = mkNP x ;}
```

# Syntax preserved as well as semantics?

- How do the English and French syntaxes differ?
  - Main difference is in words:
    - de´ montrer -> to prove
    - pair -> even
  - But the syntactic structure is largely the same
    - the combinations of resource grammar API functions
  - With one exception
    - the main verb
      - French uses the infinitive
      - English, the imperative
    - Prove p = mkUtt (mkImp (mkVP (mkVS ”prove”) (mkS p)));

# Functors

- Translations preserve up to 90% of the syntax
  - GF “Functors” parameterize syntax modules

# Grammars

- Morphology
  - definitions of parts of speech
  - synthesis: produce all forms of words
  - analysis: recognize forms of words
- Syntax
  - definitions of phrase structures
  - synthesis: produce all well-formed phrases
  - analysis: recognize phrase structures

# Availability

- GF is free open-source software
  - linguistic knowledge must accumulate!

# Applications of GF

- Translation
  - interlingual / hybrid
  - automatic / interactive
  - domain-specific / general
- Natural language interaction
  - voice commands / dialogue systems
  - software localization
- Platforms
  - desktop / mobile / cloud-based

# Phrasebook (for Tourists)

- Controlled Language for Everyday Use
  - As opposed to the usual technical domains
  - No existing formalism or domain. So CL
    - Eliminates ambiguity
- What have Berlitz, Lonely Planet, ... not done?
  - They offer a finite list of canned phrases
  - Grammars generate infinitely many sentences

# Abstract Syntax to order a beer

```
Cat
    Phrase ; Item
fun
    GivePlease : Item -> Phrase
    HereWeAre : Phrase
    ThankYou : Phrase
    YouAreWelcome : Phrase
    ABeer : Item
```

# German concrete syntax

```
lin
    GivePlease item = item ++ "bitte”
    HereWeAre = "bitte”
    ThankYou = "Danke”
    YouAreWelcome = "bitte”
    ABeer = "ein Bier"
```

So parsing is ambiguous.  What does “bitte” mean?

Depends on the context.  Disambiguate in the abstract syntax.

# But other languages?

- They make for other ambiguities
- But after design for some known languages
  - Scales up to new languages with little change
- Example:
  - "Are you Swedish?"
  - has the tree PQuestion (QProp (PropAction(ACitizen YouFamMale (CitiNat Swedish))))
  - Other trees possible because "you" is ambiguous

# Politeness and Gender

- YouFamMale, YouFamFemale, YouPolMale, YouPolFemale
- Varying this constant in the above tree gives four French linearizations:
  - YouFamMale: Est-ce que tu es suédois ?
  - YouFamFemale: Est-ce que tu es suédoise ?
  - YouPolMale: Est-ce que vous êtes suédois ?
  - YouPolFemale: Est-ce que vous êtes suédoise ?
- Although German also has gender, it makes no difference in this example.
  - YouFamMale, YouFamFemale: Bist du schwedisch?
  - YouPolMale, YouPolFemale: Sind Sie schwedisch?

# Design Principles for Phrasebook Abstract Syntax

**Convexity**: If I can say *Swedish* and *France*, I can say *Sweden* and *French*

This helps users guess what they can say.

**Orthogonality**: develop using the minimum of concepts to implement

The category Nationality = (language, nationality, country).

These triples can often be formed systematically (e.g. Swedish, Swedish, Sweden). But Belgium has no associated language, whereas Flemish has no associated country.

"Thank you" is a fixed phrase, but others need grammar. Use the RGL.

# Functors again

- The RGL has a common API for the syntax functions
  - If the languages use the same syntactic structures to express the meanings, we can use a functor.
  - But idioms mean exceptions
    - I am fifty years old
      - French: j'ai cinquante ans ("I have fifty years").
    - My name is Bond
      - German: ich heisse Bond ("I have-name Bond")
      - French: je m'appelle Bond ("I call myself Bond").
    - I am hungry
      - French: j'ai faim ("I have hunger")
  - 130 combination rules, 96 (74%) implemented by functor (usually the percentage is close to 100).

# State of coverage in GF

- There are 6000+ languages in the world.
- We have studied ca. 100 of them in GF.
- We *cover 26 languages as of Dec 2012 in the* RGL, plus 6 on-going.