CS 4744: Final Project Proposal

Ethan Koenig (etk39)

Overview

I plan to implement a machine translation system that translates between subsets of English and Italian. I plan to use the Grammatical Framework (GF) functional programming language as my primary computational tool. My goal is for the system to correctly handle idiomatic expressions such as "chiamarsi" and "avere fame".

Significance

Machine translation was one of the earliest computational lingiustic problems, and to this day remains one of the most challenging. Researchers in the late 1940s, inspired by computer's ability to break cryptographic codes, began investigating how machines could be used to produce translations between natural languages. However, the task proved much more difficult than expected, and researchers made only modest progress.

In the last two decades, statistical machine translation approaches have become increasingly popular. Today, many, if not all, of the best performing machine translation platforms employ statistical models to some degree.

However, many modern systems make errors that even non-fluent speakers would not make. I am proficient, but arguably not fluent, in Italian, and it is relatively easy for me to find flaws in the translations that systems such as Google Translate make on relatively simple sentences.

For example, when translating from English to Italian, Google Translate often mistranslates "to be hungry" constructions as "essere affamato" (which literally means "to be hungry"), whereas the correct translation would be "avere fame" (which literally means "to have hunger"). This is a mistake that even a beginner-level Italian student would not make. Results such as this have caused me to wonder if there is still a place for rules in modern machine translation systems. I am curious as to whether a small set of idiosyncratic rules for each language, such as a rule that Italian expresses hunger with "avere fame", would make these types of simple errors be less common.

Therefore, I am planning to implement a rule-based translation system that correctly handles a selection of English and Italian idiomatic expressions. If I were designing a full-fledged translation system, I would certainly incorporate statistical techniques. However, given the time constraints of the project, my more modest goal is to illustrate that rule-based techniques can avoid some of the errors common in modern translation systems.

Goals

I specifically hope to correctly handle two cases.

First, in Italian the sequence "X's name is Y" should be translated using the verb "chiamarsi", which literally means "to call oneself". For example, the correct translation of "My name is Ethan" is "Mi chiamo Ethan", whose literal translation is "I call myself Ethan". Many translation systems do not fully handle this case. (Try entering "What is the tall boy's name?" into Google Translate.)

Second, in Italian, "to be hungry" should be translated as "avere fame", which literally means "to have hunger". For example, the correct translation of "They are hungry" is "Hanno fame", whose literal translation is "They have hunger". Similarly, many translation systems do not fully handle this case. (Try entering "The boys who will be hungry left" into Google Translate.)

Third, in Italian, "to take a bath" should be translated as "fare un bagno", which literally means "to do a bath". Likewise, many translation systems do not fully handle this case. (Try entering "The boys who took a bath left." into Google Translate.)

From a high-level, I plan to handle these cases by making each language "responsible" for its own idiosyncracies. That is, in my abstract syntax, each of the above phrases will be represented in one particular way. Languages whose representation of the phrase do not match the abstract syntax will need to account for the difference it their concrete syntaxes.

For example, "to be hungry" will be represented in the abstract syntax by **be hungry** (where bold text represents the abstract concepts "be" and "hungry", instead of the concrete English words). The Italian concrete syntax will then have to include a corner case to map "avere fame" to **be hungry** instead of **have hunger**. Similarly, the phrases "to take a bath" will be represented as **do a bath**, and the English concrete syntax will need a corner case to map "take a bath" to **do a bath** instead of **take a bath**.

Grammar

I have not fully decided what my grammar will be, but I do have an idea of what it will be like.

First, it will be relatively narrow in scope. For this project, I am more focused on accuracy than breadth, so I do not plan to have an extremely large lexicon that covers a large portion of English and Italian, but instead one that covers a small, focused subset of each language.

However, I plan to choose the lexicon so that each languages has both regular and irregular words,and in the case of Italian, both genders. Further, I plan for the grammar to support multiple tenses, singularity and plurality, and negation. I will also include whatever words and constructions are necessary to support the above idiomatic expressions.

I originally planned to utilize the GF Resource Grammar Libraries for my grammars. This would have saved me from writing a lot of mechanical, boilerplate code. However, after experimenting with the Resource Grammar Library and reading its documentation more closely, I realized that it is not particularly well-suited for my project.

The Resource Grammar Library is intended for "resource grammars", or grammars that describe grammatically correct combinations of words, but do not include enough semantic information necessary for translations. For example, the abstract syntax defined by the library does not include information about plurality, polarity or tense, so all this information is lost during parsing. See the "Intermediate Results" section for a more in-depth example.

While there are some parts of the Resource Library, such as inflectional paradigms for English and Italian, which undoubtedly would be helpful for my project, I cannot "cherry-pick" the useful components of the library. Every type in the library is encapsulated, in the sense that its underlying type is hidden from the user. Without knowing the underlying structure of the types in the library, it is not possible to make additional (non-trivial) operations on these types.

Technologies

I plan to implement the system using the Grammatical Framework (GF) functional programming languages. GF offers several advantages, including

1. A robust module system, which allows for shareable, scalable and maintainable programs
2. A static type system, which can detect a wide variety of bugs at compile-time
3. Built-in parsing and linearization functionality, which greatly reduces the amount of "boilerplate" code needed
4. Native support for multilingual applications, such as the separation of abstract and concrete grammars
5. Features such as algebraic types, functors, and dependent types, which allow for elegant, symbolic, and expressive code

In addition to GF being well-suited for translation systems, I also believe that after having used the general-purpose OCaml language for the entirety of the semester, gaining exposure to a technology designed specifically for computational linguistics will be beneficial. It will both give me a taste of the selection of available computational linguistic tools, and hopefully encourage me to approach the problems I encounter in a different way than OCaml would.

Intermediate Results

I have been reading "Grammatical Frameworks: Programming with Multilingual Grammars", by Aarne Ranta, to become acquainted with the GF language. So far, I have learned about

- The structure of multilingual grammars
- GF data types, including parameters, tables and records
- Modules, inheritance and hierarchies
- Parametrized data types and record extension
- The GF Resource Grammar Library
- Module functors, and compile-time transfer
- Dependent types and selectional restrictions

Additionally, I have made some preliminary grammars to test what I have learned. Below is the abstract syntax for a simple grammar I have made as an exercise.

```
abstract Trans = {
  -- Abs stands for Abstract
  flags startcat = AbsS;
  cat
    AbsS; AbsNP; AbsVP; AbsD; AbsN; AbsNb; AbsV; AbsAdj;
  fun
    {- GRAMMATICAL RULES -}
    MakeS : AbsNP -> AbsVP -> AbsS;
    MakeNP : AbsD -> AbsNb -> AbsNP;
    AdjoinNb : AbsNb -> AbsAdj -> AbsNb;
    MakeNb : AbsN -> AbsNb;
    MakeVP : AbsV -> AbsNP -> AbsVP;

    {- LEXICAL RULES -}
      {- Adjectives -}
```

```
    Fast, Tall : AbsAdj;

  {- Determiners -}
    The, A : AbsD;

  {- Nouns -}
    Boy, Name, Student, Woman : AbsN;

  {- Verbs -}
    See, Meet : AbsV;
}
```

I implemented both English and Italian concrete syntaxes, using the GF Resource Grammar Library. Below are some results of the preliminary system.

```
> p -lang=Eng "the tall woman sees a fast boy" | l -lang=Ita
la donna alta vede un ragazzo veloce
> p -lang=Eng "the tall woman saw a fast boy" | l -lang=Ita
la donna alta vede un ragazzo veloce
> p -lang=Eng "the tall woman does not see a fast boy"
  | l -lang=Ita
la donna alta vede un ragazzo veloce
```

The system can correctly parse various tenses and polarities, thanks to the `alltense` resource libraries, but currently distinguish between tenses and polarity during linearization.

This system illustrates that the GF Resource Library is not well-suited for translation systems, since important semantic features of a sentence such as tense and polarity are lost during parsing.

It would be possible to add rules in the abstract syntax to preserve these features. For example, one could imagine adding a rule `MakePositivePresentS : AbsNP -> AbsVP -> AbsS`, which both creates a sentence using the given `AbsNP` and `AbsVP`, and signifies that the sentence is positive in polarity and present in tense. In fact, it would be possible to implement rules such as these, since the `mkS` function in the Resource Grammar Library can take arguments of type `Tense` and `Pol`.

However, there are several flaws with this approach. It is not very elegant, as it requires having many `Make...S` functions which do very similar things. Further, it become difficult to add more features, as the number of S-constructors increases exponentially with the number of features. Even worse, this approach still cannot capture the plurality of noun phrases. There is no `mkNP` function which takes an argument specifying polarity, so it would not be possible to make feature-signifying NP-constructors like `MakePositivePresentS` above.

Next Steps

As next steps, I plan to

- Finish reading "Grammatical Framework: Programming with Multilingual Grammars"
- Refactor my preliminary grammar above to not be dependent on the Resource Grammar Library
- Extend the above grammar to include more constructions that I want my final grammar to have
- Brainstorm ways to correctly handle the idiomatic expressions listed above

  This will be easier after having finished "Grammatical Framework: Programming with Multilingual Grammars", and thereby having a more comprehensive view of what GF can do

Possible Extensions

Time-permitting, there are a few more cases I would like to handle.

1. Relative Clauses:

   I would need to add support for "missing" subjects/objects, as in "The boy that ____ sneezed".

2. Questions:

   For English, I would need to add support for T-to-C movement (Subject-Aux inversion) and do-support.

Evaluation

Given the time constraints of the project, the translator will only be able to handle relatively small subsets of English and Italian. Therefore, using a third-party parallel corpus for evaluation is not practical.

Instead, I plan to randomly generate a small number English sentences and Italian sentences within my grammars using built-in GF functionality. I will then translate these sentences by hand, to collect a small gold-standard corpus. I will then use translation-quality metrics, such as the BLEU metric, to compare my system's translations with those of Google Translate.

References

Hutchins, J. (2005, November 1). The history of machine translation in a nutshell. Retrieved November 11, 2015, from http://www.hutchinsweb.me.uk/Nutshell-2005.pdf