CS 4744: Final Project Report

Ethan Koenig (etk39)

Overview

My project is a machine translation system which translates between subsets of English and Italian. I selected this project because I have an interest in machine translation, and wanted to see how the ideas, concepts and tools we learned in class could apply to a machine translation system. Additionally, I noticed that many widely-used machine translation systems perform poorly with semi-idiomatic expressions. I wanted to explore ways to build a system that could handle semi-idiomatic expressions more effectively.

My goals for the project were twofold:

1. To build a system that fully and correctly handles a specified set of semi-idiomatic expressions
2. To build a system that could easily scale to support more languages

   Due to limited amounts of time and expertise, I sought to implement support for only English and Italian; however, it was important to me that the system could easily expand to support more languages.

Semi-idiomatic Expressions

I define a semi-idiomatic expression to be an expression whose literal translation differs between two natural languages. For example, the correct Italian translation of the English sentence "the dog is hungry" is "il cane ha fame", which literally means "the dog has hunger". Thus, "be hungry/have hunger" is an semi-idiomatic expression with respect to English and Italian.

As an aside, "semi-idiomatic expression" is not a standard term for this phenomenon; I investigated whether there was an accepted term, could not find one, and chose "semi-idiomatic expression" as the term I would use for this project.

The system handles three semi-idiomatic expressions:

English	**Italian**	**Literal translation**
to be hungry	avere fame	to have hunger
to take (a/the) picture(s)	fare (una/la/le) foto	to do (a/the) picture(s)
X's name is Y	X [chiamarsi] Y	X call(s) "Xself " Y

Technologies

Following Professor Hale's suggestion[1], I decided to use Grammatical Framework[4], a statically-typed functional programming language, for my project. Grammatical Framework, abbreviated as GF, offers several advantages.

1. Support for non-context-free grammars

   The subsets of English and Italian that the system supports can (probably) be represented using context-free grammars. However, expressing them in context-free representations would not be elegant or maintainable. Moreover, this feature allows the system to easily scale to support non-context-free constructions.

2. Native parsing and linearizing functionality

   This saved me from having to worry about the low-level details of a non-context-free parsing algorithm, allowing me to focus more on encoding the English and Italian grammars.

3. An API for interfacing with general-purpose languages

   Since GF is a domain-specific programming language, interfaces with general-purpose languages allow GF to easily fit into a broader application.

4. A robust module system

   In particular, GF's support of functors, or parametrized modules, facilitate the sharing of common functionality across modules.

Approach

I decided to use the interlingua approach to machine translation for this project. The interlingua approach involves constructing a system of language-independent semantic representations, called the interlingua. Then, for each supported language L, there is a module which maps between strings of L and interlingua representations. An interlingua model can then translate by mapping a source sentence s in source language S to its corresponding interlingua representation via the S module. It then converts the interlingua representation to its corresponding sentence t in target language T via the T module[2].

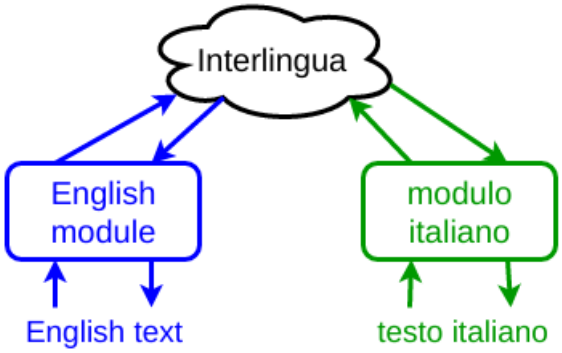

Figure 1: Illustration of the interlingua paradigm for a two-language system.

Compatibility with GF

One advantage of the interlingua approach is that GF is well-suited for interlingua systems. In GF, one defines a single abstract syntax, which describes a set of abstract syntax trees, and several concrete syntaxes, which describe a one-to-many relation between strings and abstract syntax trees. This is nearly identical to the interlingua paradigm, with the abstract syntax playing the role of the interlingua, and the concrete syntaxes playing the role of the language modules.

Modularity

Another reason I opted for the interlingua paradigm is its ability to easily support many languages. Most other translation paradigms, including paradigms lower on the Vauquois triangle and most statistical approaches, require $n(n-1)$ translation models in order to support n languages. However, the interlingua model requires only $2n$ translation models to support n languages. This allows interlingua systems to scale with much less human effort than other systems (with the possible exception of fully statistical models, which may only require annotated training data).

In addition to requiring asymptotically fewer models than other paradigms, the interlingua paradigm allows languages to be isolated from one another. This is valuable, because one can add support for a new language to an interlingua system neither knowing the currently supported languages nor worrying about their implementations. Since one of my goals was to build a system that could easily scale to more languages, the interlingua approach seemed advantageous.

Semi-idiomatic expression handling

The interlingua approach also provided a sound way of handling semi-idiomatic expressions. For each supported semi-idiomatic expression, I selected a single literal translation of that expression for the interlingua. For example, the interlingua represents the "do/take a picture" semi-idiomatic expression as "do a picture". Each language module then needs to account for the semi-idiomatic expressions whose literal translations differ from those of the interlingua, and make conversions accordingly.

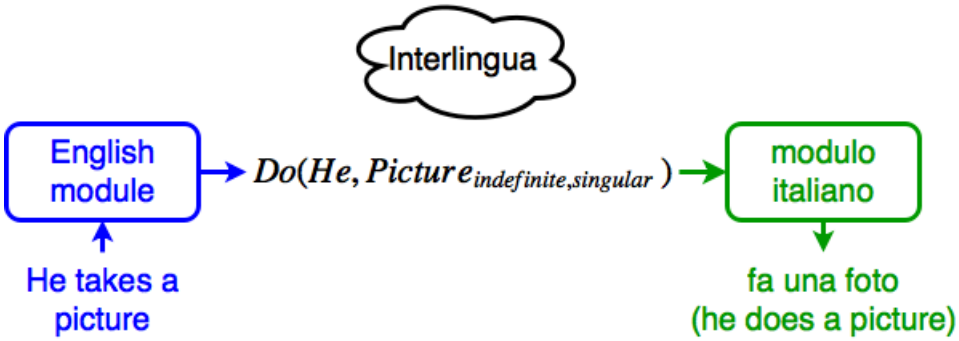

Figure 2: Interlingua handling of a semi-idiomatic expression. Note that since the interlingua's representation of "do/take a picture" is "do a picture", the English module must convert from "take a picture" to "do a picture", since it differs from the interlingua for this semi-idiomatic expression. On the other hand, the Italian module does not have to do any conversion, since it agrees with the interlingua for this semi-idiomatic expression.

Progression

While I had decided from the beginning to use GF and the interlingua approach, I made most of the implementation decisions via trial and error.

First Prototype

After becoming acquainted with GF, I tried constructing a proof-of-concept system using the GF Resource Library. Below are some of the results of that 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 could correctly parse various tenses and polarities, but could not distinguish between different tenses and polarities during linearization. This led me to conclude 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.

Even though this approach ultimately did not work, building this preliminary system was still a valuable experience. While exploring the GF Resource Library, I discovered several design paradigms that I would employ in later versions of the system. For example, the GF Resource Library relies heavily on functors (i.e. parametrized modules) to reduce redundancy, which I also do in my final version. Further, my

division of concrete syntaxes into syntactic, morphological, and lexical components closely mirrors the layout of the GF Resource Library.

Second Prototype

After deciding that building from scratch was preferable to using the GF Resource Library, I tried handling semi-idiomatic expressions directly in the GF concrete syntaxes.

This approach required defining a collection of algebraic data types that mirrored the concepts found in the abstract syntax. This is necessary, because each constituent in the concrete syntax must know what it's head is, because the transformations for handling semi-idiomatic expressions are head-dependent. For example, in the Italian concrete syntax, we want to transform "avere fame" (to have hunger), but do not want to transform "avere un cane" (to have a dog). Unfortunately, there is no way to reuse the concepts in the abstract syntax for these algebraic data types, or vice versa. Therefore, this approach required two separate modules that needed to be kept in sync with each other.

To make matters worse, due to the underlying implementation of the GF parser, this approach caused GF to replace constituents with metavariables while applying semi-idiomatic transformations. This led to the following sort of behavior

```
> p -lang=Ita "il ragazzo ha fame" | l -lang=Eng
the boy is ?
```

At this point, I decided that GF was very well-suited for parsing and linearizing between concrete and abstract syntaxes, but not well-suited for other tasks, such as semantic transformations.

Final Version

In my third prototype, which ultimately became my final version, I decided to utilize GF's Haskell interface, and perform semi-idiomatic transformations in Haskell. This approach saved me from having to keep two separate modules in sync with each other, and even better, avoided the metavariable scenarios that haunted the previous approach.

Furthermore, I noticed that there are some constructions that are much easier for GF to handle if preprocessed. For instance, when certain Italian prepositions (e.g. "di") immediately precede a definite article (e.g. "il"), the preposition and article must combine to form a single contraction ("del"). However, it is much easier to GF to handle the uncontracted versions ("di il"), so I added a text processing module for each language which maps between raw ("del") and tokenized ("di il") strings via simple, rule-based transformations.

Thus, the modules for each language L then have three components. The first component, which I refer to as the "text-and-text" component, maps between raw

and tokenized strings of L via Haskell. The second component, which I refer to as the "text-and-tree" component, maps between tokenized strings of L and language-specific abstract syntax trees via GF. The third component, which I refer to as the "tree-and-tree" component, maps between language-specific abstract syntax trees and interlingua-compliant abstract syntax trees via Haskell. Composing these three components, the module provides functionality for mapping between raw strings of L and interlingua-compliant abstract syntax trees.

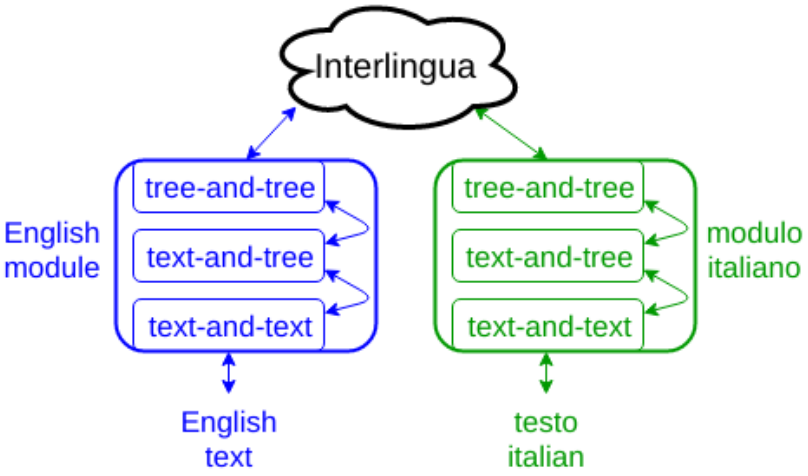

Figure 3: Module composition and interaction

Each component of a language module corresponds to a "step" up or down the Vauquois Triangle[2].

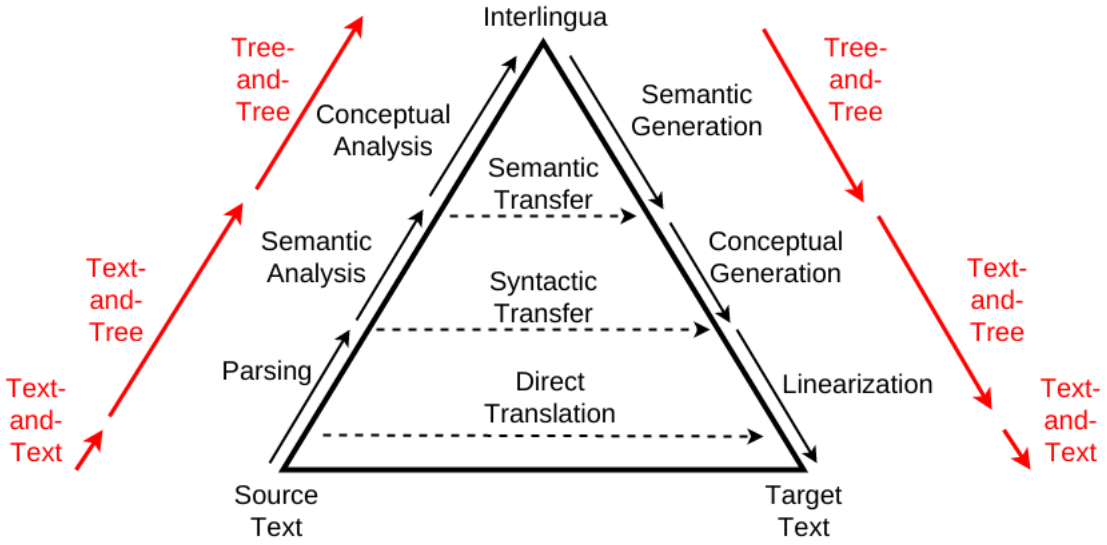

Figure 4: How the system relates to the Vauquois Triangle. The black corresponds to the Vauquois Triangle, and the red corresponds to components of the system's language modules

Evaluation

To evaluate the system, I manually created a "gold-standard" parallel corpus. It consists of a list of English sentences, each associated with a set of possible Italian translations, and a list of Italian sentences, each associated with a set of possible English translations. Since manually producing test cases is time-consuming, the corpus is relatively small; in each "direction" (i.e. English to Italian, Italian to English) there are 23 sentences without semi-idiomatic expressions and 12 sentences with semi-idiomatic expressions. I restrict the vocabulary of each source sentence to the vocabulary supported by the system.

I compare the system's translations of the sources sentences with the "gold-standard" translations using the n-gram BLEU metric[2]. However, since the system can provide several translations of a given source sentence, I compute the metric in two ways. In the first way (denoted as "Best" below), I select the best system-produced translation for each source sentence, as determined by the BLEU metric. In the second way (denoted as "Random" below), I uniformly at random select a system-produced translation for each source sentence.

I also performed the same evaluations on a baseline system, Google Translate, to provide a benchmark. Below are the results.

English to Italian

Test Cases with Semi-Idiomatic Expressions

n	System Best	Baseline Best	System Random	Baseline Random
1	1.0	0.5315	1.0	0.5315
2	1.0	0.1487	1.0	0.1487
3	1.0	0.0303	1.0	0.0303

Test Cases without Semi-Idiomatic Expressions

n	System Best	Baseline Best	System Random	Baseline Random
1	1.0	0.8078	1.0	0.8078
2	1.0	0.6525	1.0	0.6525
3	1.0	0.2931	1.0	0.2931

All Test Cases

n	System Best	Baseline Best	System Random	Baseline Random
1	1.0	0.6944	1.0	0.6944
2	1.0	0.3826	1.0	0.3826
3	1.0	0.1292	1.0	0.1292

Italian to English

Test Cases with Semi-Idiomatic Expressions

n	System Best	Baseline Best	System Random	Baseline Random
1	1.0	0.7241	0.8879	0.7241
2	1.0	0.4230	0.5975	0.4230
3	1.0	0.0455	0.2567	0.0455

Test Cases without Semi-Idiomatic Expressions

n	System Best	Baseline Best	System Random	Baseline Random
1	1.0	0.7039	0.9863	0.7039
2	1.0	0.2761	0.7784	0.2761
3	1.0	0.0343	0.5697	0.0343

All Test Cases

n	System Best	Baseline Best	System Random	Baseline Random
1	1.0	0.7107	0.9469	0.7107
2	1.0	0.3219	0.6768	0.3219
3	1.0	0.0379	0.4378	0.0379

The system generally produces more translations when translating from Italian to English than when translating from English to Italian, as evidenced by the difference in the system's random evaluation scores. See Ambiguity for a more detailed explanation.

Limitations

Despite outperforming the baseline during evaluation, the system still has several limitations.

Overgeneration

The system mildly overgenerates. If it were the case that all ungrammatical sentences in a source language always translated to sentences with no grammatical readings in a target language, this would not be a large problem.

Unfortunately, this is not the case. For example, the system accepts the ungrammatical English sentence "we see us", whose corresponding, ungrammatical Italian translation is "ci vediamo". However, "ci vediamo" also means "we see ourselves", since "ci" is both a direct-object pronoun and a reflexive pronoun in Italian. Therefore, the system correctly translates "ci vediamo" as "we see ourselves", but also incorrectly translates it as "we see us". This phenomenon contributes to the difference between the system's best and random scores in Italian to English evaluation.

Deterministic linearization

Another limitation of the system is that GF only supports deterministic linearization. That is, every abstract syntax tree yields exactly one string. This in turn means that two distinct strings cannot be parsed into the same abstract syntax tree.

This is a disadvantage, because there are many sentences with equivalent, or near equivalent, meanings that a system should treat as such. For example, the English sentences "a judge should be unbiased" and "a judge should be unprejudiced" are for (almost) all purposes semantically equivalent, and the difference between "unbiased"

and "unprejudiced" is so nuanced that we probably would not want to distinguish the two in our abstract syntax. However, using GF we must differentiate the two if we want to support both adjectives.

As another example, Italian nominative pronouns are optional; they are usually omitted, but are sometimes included for emphasis or disambiguation. Two Italian sentences that differ only in the presence of a nominative pronoun should be mapped to the same abstract syntax tree, but cannot due to the constraints of GF. I decided to have the system support omitted nominative pronouns, at the expense of supporting present nominative pronouns.

Deterministic linearization severely limits the scalability of the system. For example, Italian distinguishes between male students ("studente") and female students ("studentessa"), whereas English does not. The only way to support both "studente" and "studentessa" is for the abstract syntax to distinguish between male and female students. As more languages are added to the system, we will potentially have to account for more and more distinctions between students, which both exponentially increase the size of the abstract syntax, and require refactoring of all previously implemented language modules.

Ideally, the best way to handle this would be to decide in advance what distinctions the abstract syntax will and will not make. For example, if the architect decided not to differentiate male and female students in the abstract syntax, then both "studente" and "studentessa" would correspond to the same abstract syntax concept. In translating an English sentence containing "student" to Italian, two translations (one for "studente", another for "studentessa") would be generated, and it would be left to either the user or a probabilistic model to decide between the two. Unfortunately, this schema is not possible, as it requires one-to-many linearization from abstract syntax to concrete syntax. For more on the potential role of probabilistic models in the system, see Future steps

Performance

The system is not tremendously fast; the verification test cases, which in total parse 62 sentences, take roughly 80 seconds to run on my computer. I attribute this to the difficultly of parsing in non-context-free grammars; the fastest known parsing algorithms for mildly context-sensitive grammars have a worst-case time complexity of $O(n^6)$, where n is the length of the sentence to parse[4].

Ambiguity

The system often gives several interpretations (i.e. several abstract syntax representations) for a given source sentence, but has no mechanism for preferring one over another. This is one factor behind the difference between the system's best and random evaluation scores.

Lexical misalignment

One source of ambiguity is lexical misalignment between languages. For example, the Italian verb "fare" can either mean "to do" or "to make". To a human translator, it is usually clear from context which translation of "fare" is more appropriate. However, the system has no mechanism for preferring one translation over another, so it always outputs both.

Optional Italian pronouns

The optionality of Italian nominative pronouns is another source of ambiguity. For example, consider the Italian sentence "dorme".

	dorme
lei (omitted)	sleeps
she (omitted)	sleeps

If the subject "lei" were included, then the sentence would be unambiguous. However, without "lei", the sentence could also mean "He/It sleeps" or "Does he/she/it sleep?". To humans, the subject of the sentence could be determined via context, and whether the sentence is declarative or interrogative could be determined from punctuation or tone. However, since the system does not account for context or punctuation, it cannot rule out any of the above cases.

The ambiguity caused by the optionality of Italian nominative pronouns is one of the reasons why the system receives lower random evaluation scores when translating from Italian to English than when translating from English to Italian.

Reflections

This project was a valuable experience, and through it I gained a better understanding and appreciation for many ideas, theories, and phenomena. Below are some of my largest takeaways.

Unification Grammars

The implementation of the English and Italian concrete syntaxes closely mirrors the unification grammars we studied in class. The representations of constituents in both the English and Italian concrete syntaxes are essentially feature structures; each constituent has features such as gender, number, and person, and two constituents can combine/unify only if their features match.

Additionally, the system's underlying grammar is in some ways similar to lexical functional grammars (LFGs). In particular, both the system's abstract syntax trees and the F-structures of LFGs are abstract representations of a sentence which encode features such as plurality and tense independent of syntax. However, one key

difference between the two is that the system's abstract syntax trees are intended to be language-independent, whereas F-structures are not. One manifestation of this difference is that some of the features that are explicitly listed in a F-structure, such as case, are only implicitly encoded in an abstract syntax tree, since they may vary from language to language.

By building a system that was partially inspired by unification grammars, and then comparing the system with the grammar formalisms we covered in class, I gained a better understanding of both the grammar formalisms we covered, and how my system stands in relations to other linguistic frameworks. Since we did not have an assignment on unification grammars, I am glad that I was able to indirectly gain more exposure to them through this project.

Grammar Description

This project gave me a greater appreciation of the difficulty of comprehensively describing a grammar. Even though the system only supports small subsets of English and Italian, I still had to put a lot of thought into how to

- enforce agreement and subcategorization constraints
- enable support for auxiliary verbs, tenses, polarities, and yes-no questions
- handle language-specific irregularities, such as do-support (English), and past participle-direct object pronoun agreement (Italian)

Further, the grammars of each language model were not quite as elegant as I had originally hoped they would be; as I expanded the grammars to support more constructions, they became more and more complex. I am confident that further extension of the system would require introducing even more complexity to the grammars.

The Human Brain

One reason why I had previously underestimated the difficulty of encoding a grammar is that humans use language so naturally. However, as I realized how difficult it is to systematically describe a grammar, I also gained a greater admiration for the language-processing abilities of the human brain. I put a lot of time and effort into building a system that performs reasonably well on limited subsets of natural language, but a human brain can handle the constructions and nuances of an entire grammar seemingly effortlessly. I do not claim to know how a human mind can parse and understand a sentence, but this project has led me to realize that the mechanism(s) by which it does must be quite amazing.

Future steps

Given more time, the two future directions I would like to pursue are nondeterministic linearization and probabilistic disambiguation.

Nondeterministic linearization

As noted in Limitations, GF's inability to support deterministic linearization limits the scalability of the system. I am confident that Aarne Ranta and the other founding developers of GF had a good reason for opting to not support nondeterministic linearizations, and I would like to investigate what trade-offs would need to be made in order to support it.

Probabilistic disambiguation

As mentioned in Limitations, one of the system's biggest limitations is its inability to prefer one interpretation/parse of an input sentence over another. One way to handle this would be to have probabilistic models that rank possible interpretations of a sentence.

To train these models, I would use a parallel dataset. For each sentence in the dataset, I would input each translation of that sentence into the (non-probabilistic) system, getting a set of abstract syntax trees for each translation. I would compute the intersection of each of these sets, hopefully eliminating some of the incorrect interpretations. Each translation would then be associated with this refined set of abstract syntax trees. Using these new associations, I would then train a probabilistic model for each supported language that estimates the probability of an abstract syntax tree given a source sentence. I would need to do more research of statistical translation to decide which models would be best to experiment with.

Code

A repository for this project is available to the public on my Github page. There you will find all of the source code, evaluation results, documentation, and instructions for using the system. I encourage you to check it out!

References

[1] Hale, John T. "CS 4744: Computational Linguistics." Cornell University. Fall 2015. Lecture.

[2] Jurafsky, Dan, and James H. Martin. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. 2nd ed. Upper Saddle River, N.J.: Pearson Prentice Hall, 2009. Print.

[3] Lipovaca, Miran. "Learn You a Haskell for Great Good!" Learn You a Haskell for Great Good! N.p., n.d. Web. Nov. 2015.

[4] Ranta, Aarne. Grammatical Framework: Programming with Multilingual Grammars. Stanford, Calif.: CSLI Publications, Center for the Study of Language and Information, 2011. Print.