CSCI 4150
Final Project
English to Latin Translation

Mark Kilfoil
B00155496

Contents

1 Introduction

The goal of this project is to translate English sentences to Latin. This is done in two steps, first the English sentence is parsed using DCG (definite clause grammar) rules written in prolog. The sentence is then translated using the proper case, number and person for each noun or verb encountered. The grammar used is very basic and translates a limited vocabulary taken from the first three chapters of Wheelock's Latin 6th Edition. The first and second declension of nouns are covered as well as verbs in the present indicative active.

2 The Rules

2.1 DCG parser

In order for the translation to occur, we must first create a grammar for a sentence. We will now take a top down look at how this works. The first grammar rule used to parse a sentence is:

```
s(s(NP,VP)) --> np(Num,Per, NP), vp(Num, Per, VP).
```

This states that a sentence (S) is made up of a noun phrase (NP) and a verb phrase (VP) and that the noun phrase and the verb phrase must match in number and person. Lets take a look at the output generated by this rule:

```
?- s(s(NP,VP),[the, boy, give, money],[]).

No
```

Here the verb 'give' is not in the singular first person, so the sentence does not parse.

```
?- s(s(NP,VP),[the, boy, gives, money],[]).

NP = np(det(the), english_noun(boy))
VP = vp(english_verb(gives), np(english_noun(money)))

Yes
```

Here the proper verb 'gives' is used and the sentence parses properly. Notice the deconstruction of the sentence in to its noun phrase and verb phrase.

For the noun phrase, one of two rules is chosen:

```
np(Num,3,np(D, N)) --> det(Num,D), english_noun(Num,N).
np(Num,3,np(N)) --> english_noun(Num,N).
```

The first rule corresponds to a noun phrase that has a determiner such as 'a' or 'the'. This rule ensures that the number of the determiner matches the number of the noun in the phrase (i.e. 'the girls' not 'a girls'). The second rule corresponds to a noun phrase that is simply just a noun. The '3' argument means that both of these rules are for third person noun phrases.

For the verb phrase, the following rule is used:

```
vp(Num,Per,vp(V,NP)) -->
              english_verb(Num, Per, V),
              np(_,_,NP).
```

As you can see, a verb phrase consists of a verb and a noun phrase. The number and person of the verb is registered in the verb phrase but does not have to match the number and person of the noun phrase. In the sentence "the girls love a rose", the verb phrase 'love a rose' is plural while 'a rose' is singular.

The vocabulary is stored using rules such as:

```
det(a,several).
det(the,the).

english_noun(rose, roses).
english_noun(money, money).

english_verb(frighten, frightens).
english_verb(give, gives).
```

And accessed using rules such as:

```
det(singular, det(D)) --> [D], {det(D,_)}.

english_noun(singular,english_noun(N)) --> [N], {english_noun(N,_)}.

english_verb(singular, 3, english_verb(V)) -->
[V], {english_verb(_,V)}.
```

With this in place, we are ready to translate the sentence.

2.2 Translation

Most of the work involved in translating is already done by the DCG parser. All that is needed is the Latin vocabulary and a few translation rules.

The Latin vocabulary is stored and accessed as follows:

```
latin_verb(amo, amas, amat, amamus, amatis, amant).
latin_verb(video, vides, videt, videmus, videtis, vident).

latin_noun( pecunia, pecuniae, pecuniae, pecuniam, pecunia,
  pecunia, pecuniae, pecuniarum, pecunis, pecunias, pecunis, pecuniae).

latin_verb(plural, 3, BASE, V) :-
latin_verb(BASE,_,_,_,_,V).

latin_noun(plural, acc, BASE, N) :-
latin_noun(BASE,_,_,_,_,_,_,_,_,N,_,_).
```

The second last rule gives the plural, third person form of a Latin verb with a given BASE in the singular first person. The last rule returns the plural accusative form of a Latin noun with a given BASE in the singular nominative form.

Now only a few rules remain for translation to work:

```
translation(frighten, terreo).
translation(money, pecunia).
```

Here the English base forms of a verb and a noun are matched with the corresponding Latin base forms.

```
/* translate a noun  */
translate_noun(Num, Case, BASE) :-
translation(BASE, LBASE),
latin_noun(Num, Case, LBASE,LN),
write(LN).

/* translate a verb */
translate_verb(Num, Per, BASE) :-
translation(BASE, LBASE),
latin_verb(Num, Per, LBASE, LV),
write(LV).
```

To translate a noun, first the base noun is translated to the Latin base. Second, the Latin noun is declined according to the given number and case. A verb is translated in a similar fashion given the number and person of the verb to be translated.

```
/* translate a noun phrase */
translate_np(Num, Case, np(det(_), english_noun(N))) :-
english_noun(Num, BASE, N),
translate_noun(Num, Case, BASE).

/* translate a noun phrase */
translate_np(Num, Case, np(english_noun(N))) :-
english_noun(Num, BASE, N),
translate_noun(Num, Case, BASE).
```

There are two rules here for translating noun phrases, in each case, the base of the noun is found and then passed to the noun translator. Notice in the first case that the determiner is ignored, this is because determiners are unnecessary for the Latin translation.

```
/* translate a verb phrase */
translate_vp(Num, Per, vp(english_verb(V), NP)) :-
translate_np(_, acc, NP),
tab(1),
english_verb(Num, Per, BASE, V),
translate_verb(Num, Per, BASE).
```

To translate a verb phrase, first the noun phrase portion is translated in the accusative form since it is the object of the verb. Second, the verb is translated in accordance to the given number and person.

```
/* translate english sentence S to latin */
translate_s(S) :-

s(s(NP,VP),S,[]),
nl,
nl,
write(NP),
nl,
write(VP),
nl,
nl,
translate_np(Num, nom, NP),
tab(1),
translate_vp(Num, Per, VP),
nl.
```

This is the final rule. Given a sentence S, the rule parses the sentence using the DCG rules from section 2.1. The English noun and verb phrases are out-putted to the user. The noun phrase is then translated in the nominative form since it is the subject of the verb. The verb phrase is then translated and we are done.

3 Sample Output

Here is some sample output from the program:

- ```
  ?- translate_s([the, girl, loves, a, rose]).


  np(det(the), english_noun(girl))
  vp(english_verb(loves), np(det(a), english_noun(rose)))

  puella rosam amat

  Yes
  ```

- ```
  ?- translate_s([the, girls, love, a, rose]).


  np(det(the), english_noun(girls))
  vp(english_verb(love), np(det(a), english_noun(rose)))

  puellae rosam amant

  Yes
  ```

- ```
  ?- translate_s([the, girls, love, a, roses]).

  No
  ```

- ```
  ?- translate_s([money, frightens, the, boys]).


  np(english_noun(money))
  vp(english_verb(frightens), np(det(the), english_noun(boys)))

  pecunia pueros terret

  Yes
  ```

- ```
  ?- translate_s([the, girl, sees, a, boy]).


  np(det(the), english_noun(girl))
  vp(english_verb(sees), np(det(a), english_noun(boy)))

  puella puerum videt

  Yes
  ```
  ```

- ```
  ?- translate_s([the, boys, give, money]).


  np(det(the), english_noun(boys))
  vp(english_verb(give), np(english_noun(money)))

  pueri pecuniam dant

  Yes
  ```

- ```
  ?- translate_s([the, boy, gives, a, rose]).


  np(det(the), english_noun(boy))
  vp(english_verb(gives), np(det(a), english_noun(rose)))

  puer rosam dat

  Yes
  ```

4 Commented Code

4.1 english.pl

```
/* verbs */
english_verb(love, loves).
english_verb(see, sees).
english_verb(frighten, frightens).
english_verb(give, gives).


/* functions for conjugating verbs */
english_verb(singular, 1, BASE, BASE) :-
english_verb(BASE, _).
english_verb(singular, 2, BASE, BASE) :-
english_verb(BASE, _).
english_verb(singular, 3, BASE, V) :-
english_verb(BASE, V).
english_verb(plural, 1, BASE, BASE) :-
english_verb(BASE, _).
english_verb(plural, 2, BASE, BASE) :-
english_verb(BASE, _).
english_verb(plural, 3, BASE, BASE) :-
english_verb(BASE, _).

/* verb grammar */
english_verb(singular, 1, english_verb(V)) -->
[V], {english_verb(V,_)}.

english_verb(singular, 2, english_verb(V)) -->
[V], {english_verb(V,_)}.

english_verb(singular, 3, english_verb(V)) -->
[V], {english_verb(_,V)}.

english_verb(plural, 1, english_verb(V)) -->
[V], {english_verb(V,_)}.

english_verb(plural, 2, english_verb(V)) -->
[V], {english_verb(V,_)}.

english_verb(plural, 3, english_verb(V)) -->
[V], {english_verb(V,_)}.

/* nouns */
english_noun(boy, boys).
english_noun(girl, girls).
english_noun(rose, roses).
```

```
english_noun(money, money).


english_noun(singular, BASE, BASE) :-
english_noun(BASE,_).

english_noun(plural, BASE, N) :-
english_noun(BASE,N).

/* noun grammar */
english_noun(singular,english_noun(N)) --> [N], {english_noun(N,_)}.
english_noun(plural,english_noun(N)) --> [N], {english_noun(_,N)}.


/* determiners */
det(singular, det(D)) --> [D], {det(D,_)}.
det(plural, det(D)) --> [D], {det(_,D)}.

det(a,several).
det(the,the).

/* noun phrases */
np(Num,3,np(D, N)) --> det(Num,D), english_noun(Num,N).
np(Num,3,np(N)) --> english_noun(Num,N).


/* verb phrases */
vp(Num,Per,vp(V,NP)) -->
              english_verb(Num, Per, V),
              np(_,_,NP).

/* sentences */
s(s(NP,VP)) --> np(Num,Per, NP), vp(Num, Per, VP).

/* functions to print grammar for a given sentence */
print_s(S) :-  s(s(NP,VP),S,[]),
write(NP),
nl,
write(VP),
nl,
print_np(NP),
tab(1),
print_vp(VP),
nl.

print_np(np(det(D), english_noun(N))) :-
write(D),
tab(1),
```

```
write(N).

print_vp(vp(english_verb(V), NP)) :-
write(V),
tab(1),
print_np(NP).
```

4.2 latin.pl

```
/* Latin verbs conjugated by person and number
Present Indicative Active */

/* amare, to love */
latin_verb(amo, amas, amat, amamus, amatis, amant).

/* videre, to see */
latin_verb(video, vides, videt, videmus, videtis, vident).

/* terrere, to frighten */
latin_verb(terreo, terres, terret, terremus, terretis, terrent).

/* dare, to give */
latin_verb(do, das, dat, damus, datis, dant).


/* functions for conjugating verbs */
latin_verb(singular, 1, BASE, BASE) :-
latin_verb(BASE,_,_,_,_,_).
latin_verb(singular, 2, BASE, V) :-
latin_verb(BASE,V,_,_,_,_).
latin_verb(singular, 3, BASE, V) :-
latin_verb(BASE,_,V,_,_,_).
latin_verb(plural, 1, BASE, V) :-
latin_verb(BASE,_,_,V,_,_).
latin_verb(plural, 2, BASE, V) :-
latin_verb(BASE,_,_,_,V,_).
latin_verb(plural, 3, BASE, V) :-
latin_verb(BASE,_,_,_,_,V).


/* latin nouns
First Declension */


/*boy*/
latin_noun( puer,
pueri,
puero,
puerum,
puero,
puer,
pueri,
puerorum,
pueris,
```

```
pueros,
pueris,
pueri).

/*girl*/
latin_noun( puella,
puellae,
puellae,
puellam,
puella,
puella,
puellae,
puellarum,
puellis,
puellas,
puellis,
puellae).

/*rose*/
latin_noun( rosa,
rosae,
rosae,
rosam,
rosa,
rosa,
rosae,
rosarum,
rosis,
rosas,
rosis,
rosae).

/*money*/
latin_noun( pecunia,
pecuniae,
pecuniae,
pecuniam,
pecunia,
pecunia,
pecuniae,
pecuniarum,
pecunis,
pecunias,
pecunis,
pecuniae).

/* functions for declining nouns */
latin_noun(singular, nom, BASE, BASE) :-
```

```
latin_noun(BASE,_,_,_,_,_,_,_,_,_,_,_,_).

latin_noun(singular, gen, BASE, N) :-
latin_noun(BASE,N,_,_,_,_,_,_,_,_,_,_,_).

latin_noun(singular, dat, BASE, N) :-
latin_noun(BASE,_,N,_,_,_,_,_,_,_,_,_,_).

latin_noun(singular, acc, BASE, N) :-
latin_noun(BASE,_,_,N,_,_,_,_,_,_,_,_,_).

latin_noun(singular, abl, BASE, N) :-
latin_noun(BASE,_,_,_,N,_,_,_,_,_,_,_,_).

latin_noun(singular, voc, BASE, N) :-
latin_noun(BASE,_,_,_,_,N,_,_,_,_,_,_,_).

latin_noun(plural, nom, BASE, N) :-
latin_noun(BASE,_,_,_,_,_,N,_,_,_,_,_,_).

latin_noun(plural, gen, BASE, N) :-
latin_noun(BASE,_,_,_,_,_,_,N,_,_,_,_,_).

latin_noun(plural, dat, BASE, N) :-
latin_noun(BASE,_,_,_,_,_,_,_,N,_,_,_,_).

latin_noun(plural, acc, BASE, N) :-
latin_noun(BASE,_,_,_,_,_,_,_,_,N,_,_,_).

latin_noun(plural, abl, BASE, N) :-
latin_noun(BASE,_,_,_,_,_,_,_,_,_,N,_,_).

latin_noun(plural, voc, BASE, N) :-
latin_noun(BASE,_,_,_,_,_,_,_,_,_,_,N).
```

4.3 translate.pl

```
/*verbs*/
translation(love, amo).
translation(see, video).
translation(frighten, terreo).
translation(give, do).

/*nouns*/
translation(boy, puer).
translation(girl, puella).
translation(rose, rosa).
translation(money, pecunia).

/* translate a noun  */
translate_noun(Num, Case, BASE) :-
translation(BASE, LBASE),
latin_noun(Num, Case, LBASE,LN),
write(LN).

/* translate a verb */
translate_verb(Num, Per, BASE) :-
translation(BASE, LBASE),
latin_verb(Num, Per, LBASE, LV),
write(LV).

/* translate a noun phrase */
translate_np(Num, Case, np(det(_), english_noun(N))) :-
english_noun(Num, BASE, N),
translate_noun(Num, Case, BASE).

/* translate a noun phrase */
translate_np(Num, Case, np(english_noun(N))) :-
english_noun(Num, BASE, N),
translate_noun(Num, Case, BASE).

/* translate a verb phrase */
translate_vp(Num, Per, vp(english_verb(V), NP)) :-
translate_np(_, acc, NP),
tab(1),
english_verb(Num, Per, BASE, V),
translate_verb(Num, Per, BASE).

/* translate english sentence S to latin */
translate_s(S) :-

s(s(NP,VP),S,[]),
nl,
```

```
nl,
write(NP),
nl,
write(VP),
nl,
nl,
translate_np(Num, nom, NP),
tab(1),
translate_vp(Num, 3, VP),
nl.
```

5 Future work

This project has many aspects that can be extended and enhanced. I will list a few here:

- The vocabulary is very small but can very easily be added to.
- Transitive verbs are not handled, the indirect object of which would take the dative form.
- The translation only works one way, it would be nice to work in both directions.
- Verbs and nouns are conjugated and declined in the code, it would be better to just give the root and have the rules do the declination based on common endings.

6 References

- Wheelock, Frederick M. (2000) Wheelock's Latin 6th Edition.
  HarperCollins, New York.
- http://www.csupomona.edu/ jrfisher/www/prolog_tutorial/7_1.html