Computational Linguistics: Long-Distance Dependencies

Raffaella Bernardi

KRDB, Free University of Bozen-Bolzano

P.zza Domenicani, Room: 2.28, e-mail: bernardi@inf.unibz.it

Contents

1. Summary

In the first lesson after the Winter break, we summarized the first part of the course and we saw that there were still some questions left open.

- Syntax:
  1. Is NL a Context Free Language or do we need a more expressive Formal Grammar?
  2. Can CFG deal with long-distance dependencies?
- How do syntax and semantics relate?

We have answered the last question in the last two lessons.

Next week, we will answer question 1; now we look at long-distance dependencies in more detail and give an answer to 2.

2. Long-Distance Dependencies

We have seen that interdependent constituents need not be juxtaposed, but may form long-distance dependencies, manifested by gaps

- Raynair services Pescara.
- What cities does Ryanair service [. . .]?

The constituent what cities depends on the verb service, but it is at the front of the sentence rather than at the object position.

The displaced element is called the filler and the initial position is known as gap.

2.1. Unbounded Dependencies

These dependencies may extend across arbitrarily many clause boundaries, therefore they are also known as **unbounded dependencies**.

> I wonder who [Sandy loves [. . .]]$_s$
> Kim, [Chris knows Sandy trusts [. . .]]$_s$

Furthermore, in same cases the filler and the gap must **agree on the case** (as above), others in which they don't

> Kim would be easy to bribe [. . .]

Finally, there are also phenomena of double gaps

1. That was the rebel leader who rivals of [. . .] shot [. . .].
2. This is a problem which$_1$ John$_2$ is difficult to talk to [. . .]$_2$ about [. . .]$_1$.

2.2. Kinds of long-distance dependencies

1. Kim_1, Sandy loves $[\ldots]_1$. (topicalization)
2. Who_1 does Sandy loves $[\ldots]_1$? (wh-question)
3. This is the politician [who_1 Sandy loves $[\ldots]_1$] (wh-rel. clause)
4. It's Kim [who_1 Sandy loves $[\ldots]_1$]. (it-cleft)
5. [What_1 Kim loves $[\ldots]_1$] is Sandy. (pseudocleft)
6. I bought it_1 for Sandy to eat $[\ldots]_1$. (purpose infinitive)
7. Sandy_1 is hard to love $[\ldots]_1$ (tough movement)
8. This is the politician_1 [Sandy loves $[\ldots]_1$] (relative clause)

The first five cases are known as **strong unbounded dependencies**, to indicate that they require the filler and the gap to be of the same **syntactic category**.

2.3. Strong unbounded dependencies

1. Kim_1, Dana believes Chris knows Sandy trusts $[\ldots]_1$
2. $[\text{On Kim}]_1$, Dana believes Chris knows Sandy depends $[\ldots]_1$.
3. *$[\text{On kim}]_1$, Dana believes Chris knows Sandy trusts $[\ldots]_1$.

2.4. Multiple extractions

Multiple extractions like below are ungrammatical:

- The witch gave the house-elf to Harry.
- The house-elf which the witch gave [. . .] to Harry . . .
- The student who the witch gave the house-elf to [. . .] . . .
- *The house-elf who the witch gave [. . .] to [. . .] . . .

2.5. Island constraints

Although there is no bound on the distance between fillers and gaps, there are a **number of constraints** about the relative position in which filler and their corresponding gaps may appear. These are known as "island constraints" following Ross 1967. E.g.

- The gap may not be in a **relative clause** if the filler is outside of it

  *Which dog did you criticize the person [who kicked [...]]?

- A gap cannot be in **coordinate conjoined structures not containing its filler**, unless all conjunctions have gaps by the same filler:

  1. *What did they buy [[...] and forget their credit card at the store]?
  2. What did they buy [[...] and forget [...] at the store?]

A great deal of research (in Linguistics!) has gone into these sorts of constraints. Computational Linguistics should take profit of their understanding of these phenomena when formalizing NL.

2.6. Parasitic gaps

It can happen that two or more traces in the same sentence are bound by the same filler.

> That was the rebel leader who$_1$ rivals of [. . .]$_1$ shot [. . .]$_1$.

The examples below show that the parasitic gap is the first one, which would cause ungrammaticality if left alone (1), differently from the second gap (2)

1. *That was the rebel leader who rivals of [. . .] shot the British consul.
2. That was the rebel leader who agents of foreign powers shot [. . .].

3. Long Distance Phenomena in CFG

Recall from lecture 2, that Context free rules work **locally**. For example, the rule

$$s \rightarrow np\ vp$$

tells us how an s can be decomposed into two parts, an np and a vp.

But we have seen that certain aspects of natural language seem to work in a non-local, **long-distance way**.

Indeed, for a long time it was thought that such phenomena meant that grammar-based analyses had to be replaced by very powerful new mechanisms (such as the transformations used in transformational grammar).

4. Trasformational Grammar

The traditional explanation basically goes like this. We have the following sentence:

> Harry likes the witch

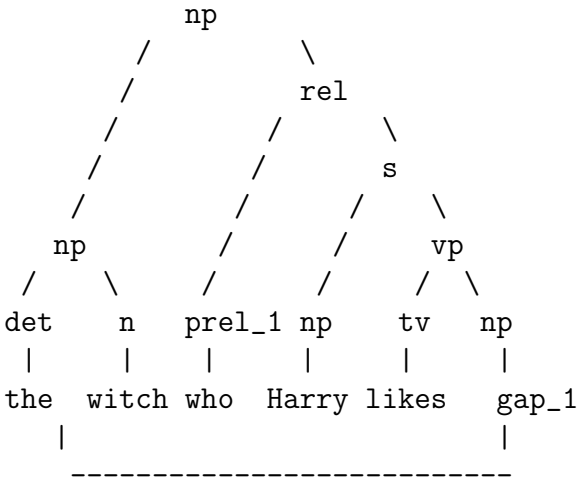
We can think of the np with the object relative clause as follows.

```
      ----------------------
    |                        |
the witch who Harry likes GAP(np)
```

That is, we have

1. extracted the np "the witch" from the object position, leaving behind an np-**gap**,
2. **moved** it to the front, and
3. placed the relative pronoun "who" between it and the gap-containing sentence.

4.1. Trasformational Grammar: Parse Tree

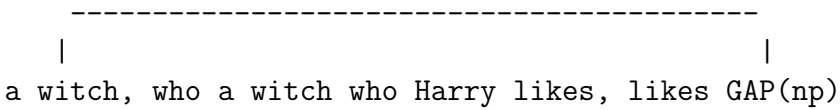


4.2. Trasformational Grammars: Relative Clause

Recall: Relative clauses are an example of **unbounded dependencies**.

The word '**dependency**' indicates that the moved np is linked, or depends on, its original position.

The word '**unbounded**' is there because this "extracting and moving" operation can take place across arbitrary amounts of material. For example, from



This way of thinking is reflected in Trasformational Grammar by means of the **movement** operation that leaves a **trace** in the position from where the move started.

5. Non-Transformational Approaches

CTL and DCG can analyze the phenomena introduced above without making use of the (computational expensive!!!) movement operation.

Projects A) People with strong Logic background (eg. Magdalena) could study how CTL can be used to analyze long-distance phenomena. (Ref. to Moortgat 02)

B) People with knowledge of PROLOG (eg. Mantas and Linas) could implement in PROLOG a DCG able to parse (at least some of) the sentences above. (Ref. to Pereira 00)

Critiques Long-Distance phenomena are interesting also from a Psycolinguistics point of view. Linas is going to read about it.

We will now look at a first solution in DCG.

5.1. A first solution

Gerald Gazdar in '81 introduced an alternative analysis which does not make use of the complex mechanism of transformations.

He proposes an approach that doesn't talk about moving bits of sentences round. He just talks about **missing information**, and says which kind of information needs to be **shared** with the other category.

The link with the movement story will be clear: but the new information-based story is simpler, clearer and more precise.

5.2. Gap as features: NP rules

Let's first look at the NP rules:

```
i.      np(nogap) --> det n.
ii.     np(nogap) --> det n rel.
iii.    np(nogap) --> pn.
iv.     np(nogap) --> pn rel.
v.      np(gap(np)) --> [].
```

The first four rules say that an English NP can consist of:

i. a determiner and a noun (for example: "a witch"), or
ii. a determiner and a noun followed by a relative clause (for example: "a witch who likes Harry"), or
iii. a proper name (for example: "Harry"), or
iv. a proper name followed by a relative clause (for example: "Harry, who likes a house-elf").

All these NPs are 'complete' or 'ordinary' NPs. **Nothing is missing** from them. That is why the extra argument on NP contains the value **nogap**.

5.3. Gap as features: Empty String

What about the fifth rule?

```
v.  np(gap(np)) --> [].
```

This tells us that an NP can also be realized as an **empty string** — that is, as nothing at all. Obviously, this is a special rule: it's the one that lets us **introduce gaps**. It says:

> we are free to use 'empty' NPs, but such NPs have to be marked by a feature which says that they are special.

Hence in this rule, the value of the extra argument is **gap(np)**. This tells us that we are dealing with a special NP — one in which the usual NP information is absent.

5.4. Gap as Feature: S rule

Now for the S rule.

```
s(Gap) --> np(nogap) vp(Gap).
```

This rule says that an S consists of an NP and a VP.

- **No empty subject** Note that the NP must have the feature nogap. This simply records the fact that in **English the NP in subject position cannot be realized by the empty string** (in some languages, for example Italian, this is possible in some circumstances).
- **Feature Passing** Moreover, note that the value of the **Gap variable** carried by the VP (which will be either nogap or gap(np), depending on whether the VP contains empty NPs) is **unified** with the value of the Gap variable on the S. That is, we have here an example of **feature passing**: the record of the missing information in the verb phrase (if any) is passed up to the sentential level, so that we have a record of exactly which information is missing in the sentence.

5.5. Gap as Feature: VP rule

The VP rule is

```
vp(Gap) --> tv  np(Gap).
```

This rule says that a VP can consist of an ordinary transitive verb together with an NP.

Note that this rule also performs **feature passing**: it passes the value of Gap variable up from the NP to the VP. So the VP will know whether the NP carries the value nogap or the value gap(np).

5.6. Gap as Feature: Rel rules

Now for the relativization rules:

```
1.    rel --> prorel s(gap(np)).
2.    rel --> prorel vp(nogap).
```

1. The first rule deals with **relativization in object position** — for example, the clause "who Harry likes" in "The witch who Harry likes".

   The clause "who Harry likes" is made up of the relative pronoun "who" (that is, a prorel) followed by "Harry likes". What is "Harry likes"? It's a sentence that is missing its object NP — that is, it is a s(gap(np)), which is precisely what the first relativization rule demands.

2. The second rule deals with **relativization in subject position** — for example, the clause "who likes the witch" in "Harry, who likes the witch".

   The clause "who likes the witch" is made up of the relative pronoun "who" (that is, a prorel) followed by "likes the witch". What is "likes the witch"? Just an ordinary VP — that is to say, a vp(nogap) just as the second relativization rule demands.

5.7. Lexicon

And that's basically it. We only have to add a few lexical rules and we're ready to go. Here's some lexicon:

```
n --> [house-elf].
n --> [witch].
pn --> [harry].
det -->[a].
det -->[the].
tv --> [likes].
tv --> [watches].
prorel --> [who].
```

5.8. Gap as features: Extension

Let's try to extend the tiny fragment with verbs such as "give" which subcategorize for an NP followed by a PP.

Consider the following sentence:

"The witch gave the house-elf to Harry".

We can relativize the NP argument of "give" (that is, the NP "the house-elf") to form:

"the house-elf who the witch gave [...] to Harry".

Moreover, we can also relativize the NP that occurs in the PP "to Harry". That is, we can take "Harry" out of this position, leaving the "to", behind to form:

"Harry, who the witch gave the house-elf to [...]".

We would like our DCG to handle such examples.

5.9. Gap as features: Multiple Extractions

But note — there are also some things that the DCG should not do, namely perform **multiple extraction**. There are now two possible NPs that can be moved: the first argument NP, and the NP in the PP. Can both be moved at once? In some languages, yes. In English, no. That is, in English

> "*The house-elf who the witch gave",

is not an NP.

So when we write our DCG, not only we have to make sure we **generate the NPs we want**, we also have to make sure that we **don't build NPs using multiple extraction**.

Now, we can develop our previous DCG to do this — but the result is not something a linguist (or indeed, a computational linguist) should be proud of. Let's take a closer look.

5.10. Gap as features: PP rules

First of all, we need a bit more lexicon: we'll add the verb "gave" and the preposition "to":

```
datv --> [gave].
p --> [to].
```

As we are going to need to build prepositional phrases, we need a rule to build them.

The rule has to say that a prepositional phrase can be built out of a preposition followed by an NP.

Moreover, we have to use the extra argument to pass up the value of the Gap feature from the NP to the PP. So the PP will know whether the NP is ordinary one, or a gap.

```
pp(Gap) --> p np(Gap).
```

5.11. Gap as features: VP rules

Now comes the crucial part: the new VP rules. We need to allow single extractions, and to rule out double extractions. Here's how this can be done — and this is the part **linguists won't like**:

```
vp(Gap) --> datv np(nogap),pp(Gap).
vp(Gap) --> datv np(Gap),pp(nogap).
```

We have added two VP rules.

1. The first rule insists that the NP argument be gap-free, but allows the possibility of a gap in the PP.
2. The second rule does the reverse: the NP may contain a gap, but the PP may not.

5.12. Gap as features: VP rules (Cont'd)

Either way, at least one of the VPs two arguments must be gap-free, so **multiple extractions are ruled out**.

Now, this does work since it generates the grammatical sentences we saw, and moreover, it refuses to accept multiple extractions.

So why would a linguist not approve of this DCG?

5.13. Is not a good grammar

Because we are handling **one construction** — the formation of VPs using DATV verbs — with **two rules**.

The **role of syntactical rules** is to make a structural claim about the combination possibilities in our language. Thus there should be one rule for building VPs out of DATV verbs, for there is only one structural possibility: datv verbs take an NP and a PP argument, in that order.

We used two rules not because there were two possibilities, but simply to do a bit of **'feature hacking'**: by writing two rules, we found a cheap way to rule out multiple extractions. But this is a silly way to write a grammar.

As we saw when we discussed the case example (3 lesson), one of the roles of features is precisely to help us **minimize the number of rules** we need — so to add extra rules to control the features is sheer craziness!

5.14. Practical Disadvantages

There are also practical disadvantages to the use of two roles. For a start, many **unambiguous sentences now receive two analyses**. For example,

> "The witch gave the house-elf to Harry"

is analyzed two distinct ways.

Such **spurious analyses** are a real nuisance — natural language is ambiguous enough anyway. We certainly don't need to add to our troubles by writing DCGs in a way that guarantees that we generate too many analyses!

Furthermore, adding extra rules is just not going to work in the long run. As we add more verb types, we are going to need more and more duplicated rules. The result will be a **mess**, and the **grammar will be ugly and hard to maintain**. We need a better solution — and there is one.

... that Mantas and Linas are going to study and explain to us

;)

6. Conclusion

Hence, the answer to the question

> Can CFG deal with long-distance dependencies?

is YES.

The mechanism is known as Gap Threading and makes use of difference list to obtain the feature passing.

Next week we will compare some well known Formal Grammars used by currently used by Computational Linguists and answer the question of:

> Is NL a Context Free Language or do we need a more expressive Formal Grammar?

Remark If you have not done it yet, please let me know by Tuesday which paper you will review.