Making sense of legal texts

de Maat, E.

Publication date
2012

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: https://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Appendix B  Detecting References using JAPE

In chapter 3 of this thesis, the different language formats that are used for references have been discussed. These language formats can be detected using patterns. In this appendix, it is shown how such patterns look like in JAPE, with the goal of applying annotations in the style of MetaLex 1.3.1. Note that these patterns are not the ones used in the experiment described in section 3.4, though they are very similar. The patterns used in the experiment were written down in a different notation, and we have opted to include the JAPE patterns instead as we have also used JAPE grammars in appendix A, as we feel these are easier to read.

Please refer to appendix A for a short introduction of JAPE. For the detection of references, we assume that a GATE tokeniser has been applied to the input. Furthermore, since the names of Dutch legal sources do not conform to a specific pattern, they must be found using a list of names. This is done using a separate GATE resource, a GATE Gazetteer, and is not part of this JAPE grammar (though the grammar uses the annotations left by the Gazetteer as input).

Since the patterns for references overlap with the patterns for headers, these patterns will also find annotate parts of headers, unless these have already been marked. It is therefore assumed that the headers have already been marked in such a way that they no longer appear as input for the reference, or that references found in headers will be filtered out afterwards.

B.1  Basic References

The simplest references consist of a category label and an index – similar to a heading. So, we can use a similar same pattern. To simplify the patterns, we first apply a temporary annotation Index to each token (or group of tokens) that may be an index, i.e. each number, single letter, a number followed by °, etc.

Now, for a reference to a list item, several labels are used: sub, onder and onderdeel. All of them may appear with or without a capital. So, the pattern for a list item is:

```
{ ( {Token.string == "Sub"} | {Token.string == "sub"} |
   {Token.string == "Onder"} | {Token.string == "onder"} |
   {Token.string == "Onderdeel"} | {Token.string == "onderdeel"} )
   ( Index )
 ) : ListItem
```

Now, we would like to mark these using a Cite annotation, with the information that is needed to resolve the reference (in this case, the information that it is a reference to a list item with the specific index found). The following block of code does that:

---

184 http://legacy.metalex.eu/
185 It is possible to simplify this pattern somewhat: {Token.string =~ "[Ss]ub") matches both Sub and sub.
gate.AnnotationSet set = (gate.AnnotationSet)bindings.get("ListItem");
gate.Annotation index = (gate.Annotation)set.get("Index");
gate.Annotation indexAnn = (gate.Annotation)aanduiding.iterator().next();
gate.FeatureMap features = Factory.newFeatureMap();
features.put("onderdeel", indexAnn.getFeatures().get("value"));
features.put("kind", "onderdeel");
outputAS.add(set.firstNode(), set.lastNode(), "Cite", features);
inputAS.removeAll(Index);

The first line selects all the annotations within the set marked as ListItem, i.e. the entire reference found. Then, the second line selects all the Indexes that appear within that set. The third line selects the first element of that set, i.e. the first Index (which should also be the only index).

Now, we start creating a set of attributes (called features in GATE). We first create a new set, add the attribute onderdeel and give it a value equal to indexAnn.getFeatures().get("value"), which is the actual index found. Also, we add an attribute kind which we give the value onderdeel, to indicate that this specific Cite annotation refers to a list item.

Finally, we add the actual Cite annotation and remove the temporary Index annotation.

### B.2 References to Ranges

In MetaLex, a reference to a range, such as items a to d, is marked up differently from simple references. The reference as a whole is marked as a CiteRange, with the first index of the range marked as CiteFrom and the last index as CiteTo, e.g.:

<CiteRange><items><CiteFrom>a</CiteFrom> to <CiteTo>d</CiteTo></CiteRange>

To simplify the patterns needed for ranges, we can use temporary annotations for the indices, such as:

Rule: Range

```java
{({Index}):Start
  {Token.string == ")"
   |
   {Token.string == "tot"
    {Token.string == "en"}
    {Token.string == "met"}
  }
}
({Index}):End

} -->
```

```java
gate.AnnotationSet start = (gate.AnnotationSet)bindings.get("Start");
gate.AnnotationSet index = (gate.AnnotationSet)start.get("Index");
gate.Annotation indexAnn = (gate.Annotation)index.iterator().next();
gate.FeatureMap features = Factory.newFeatureMap();
```
This rule recognises patterns like 2 – 5 and 3 up to 5, and marks such patterns with a Range annotation, with the indices marked as CiteFrom and CiteTo, which will make it easy to convert the Range to a CiteRange. No kind attribute is added to the CiteFrom and CiteTo attributes, as it is not yet known what kind of element (list item, subparagraph, article, etc.) they refer to. An attribute value holds the actual index found. After all references have been found, any remaining Range annotations can be removed, together with the corresponding CiteFrom and CiteTo annotations.

Rule: EnkelOnderdeelRange
{
    ({Token.string == "Sub"}||{Token.string == "sub"}||{Token.string == "Onder"}||{Token.string == "onder"}||{Token.string == "Onderdeel"}||{Token.string == "onderdeel"}||{Token.string == "Onderdelen"}||{Token.string == "onderdelen"})
    (Range)
} Onderdeel -->
{
gate.AnnotationSet set = (gate.AnnotationSet)bindings.get("Onderdeel");
gate.AnnotationSet start = (gate.AnnotationSet)inputAS.get("CiteFrom", set.firstNode().getOffset(), set.lastNode().getOffset());
gate.AnnotationSet end = (gate.AnnotationSet)inputAS.get("CiteTo", set.firstNode().getOffset(), set.lastNode().getOffset());
gate.AnnotationSet range = (gate.AnnotationSet)set.get("Range");

gate.Annotation startAnn = (gate.Annotation)start.iterator().next();
startAnn.getFeatures().put("onderdeel",
startAnn.getFeatures().get("value"));
startAnn.getFeatures().remove("value");
gate.Annotation endAnn = (gate.Annotation)end.iterator().next();
endAnn.getFeatures().put("onderdeel",
endAnn.getFeatures().get("value"));
endAnn.getFeatures().remove("value");
With this range annotation, the pattern for the range becomes similar to the pattern for a simple annotation: a category label[186] followed by a range token:

```
{|(Token.string == "Sub")|{Token.string == "sub"} |
|{Token.string == "Onder"}||{Token.string == "onder"}) |
|{Token.string == "Onderdeel"}||{Token.string == "onderdeel")} |
|{Token.string == "Onderdelen"}||{Token.string == "onderdelen")}
```

In order to apply the correct annotation, we need to perform the following steps:

- update the `CiteFrom` and `CiteTo` annotations to indicate they refer to a list item;
- remove the temporary `Range` annotation;
- add a `CiteRange` annotation around the entire reference.

Adding the `CiteRange` and removing the `Range` is done in the same way as new annotations were added and temporary annotations were removed in the examples above.

Adding the additional feature to the `CiteRange` annotation is done using the following code:

```java

gate.AnnotationSet set =
  (gate.AnnotationSet)bindings.get("ListItemRange");
gate.AnnotationSet start = (gate.AnnotationSet)inputAS.get("CiteFrom",
  set.firstNode().getOffset(), set.lastNode().getOffset());
gate.Annotation startAnn = (gate.Annotation)start.iterator().next();
startAnn.getFeatures().put("onderdeel",
  startAnn.getFeatures().get("value"));
startAnn.getFeatures().remove("value");
```

The first line selects all the annotations within the text labeled as `ListItemRange`, i.e. the entire reference that has been found. The second line selects all `CiteFrom` annotations, and the third line selects the first annotation from that set, which should also be the only `CiteFrom` annotation in the range. The fourth line adds an attribute `onderdeel` to the annotation, giving it the same value as the attribute `value`. This attribute value was added when we detected the range, and holds the value of the index, e.g. "1", "a" or "1°". After copying the attribute `value`, it is deleted. (So, effectively we have renamed the `value` attribute to `onderdeel`.)

---

[186] There are two additional labels, as for a range, the plural of one of the labels is also used.
B.3 Multiple References

A reference may also have several different targets that are not grouped together in a range (though some of them may be). In MetaLex, these are marked using a CiteGroup tag, with any individual indices marked with Cite and each range marked with a CiteRange (with the indices of the range marked using CiteFrom and CiteTo, as in the previous section). For example, we want to recognise a reference like items a, b to f and g, and mark it up as follows:

```
<CiteGroup>items <Cite>a</Cite>, <CiteRange><CiteFrom>b</CiteFrom> to <CiteTo>f</CiteTo></CiteRange> and <Cite>g</Cite></CiteGroup>
```

Using the Index and Range temporary annotations, the pattern for such a reference can be described as:

```
{ (Token.string == "Sub")|{Token.string == "sub"} 
|{Token.string == "Onder"}|{Token.string == "onder"}) 
|{Token.string == "Onderdeel"}|{Token.string == "onderdeel"}) 
|{Token.string == "Onderdelen"}|{Token.string == "onderdelen"})

{ {{Index}}|{{Range}} 
|{{Token.string == ","}}{{Index}}|{{Range}})* 
|{{Token.string == "en"}}|{{Token.string == "of"}} 
|{{Index}}|{{Range}}

}):ListItemGroup
```

The pattern starts with a category label, followed by an index or a range, then optionally several times a comma followed by another index or range, and finally the word and or or and one more index or range.

In order to achieve the desired mark-up, the following steps need to be taken:

- add an CiteGroup annotation to the entire reference;
- replace all Index temporary tags by a Cite tag, and change the value attribute to an onderdeel attribute;
- replace all Range temporary tags by a CiteRange tag;
- change the value attribute on the CiteFrom and CiteTo tags to an onderdeel attribute.

We have seen these operations in the example above. The main difference is that there may now be multiple annotations of the same type. This means that we will have to do certain operations for all the elements in a given set. The code below shows how this is done for the set of all Indices (which should be replaced by a Cite tag).

```java
gate.AnnotationSet set = (gate.AnnotationSet)bindings.get("ListItemGroup");
gate.AnnotationSet index = (gate.AnnotationSet)set.get("Index");
gate.Annotation current;
```
As before, the first two lines select the set of all the indices. Then, we declare an iterator, which is a JAVA object that allows us to select all the items of the set by one. For each element of the set, a new Cite annotation is added. Finally, the entire set of temporary Index annotations is removed.

B.4 Zooming-In Layered References

The patterns above can be used to detect references that refer to a single “layer” of a document, i.e. only items, or only articles. But in chapter 3, we also mentioned the layered references, which refer to a specific part of a document and then zoom in to a specific subpart, such as article 12, item b.

Such patterns can be detected by applying multiple phases for each level. For example, if we first detect all references at item level, followed by all references at subparagraph level, a part of the example above will already have been tagged, as follows:

```
article 12, <Cite>item b</Cite>
```

The pattern for detecting such a reference is the same as the pattern for detecting a single article reference followed by a Cite tag. Since the Cite tag may also mark a subparagraph, or a layered reference to an item within a subparagraph, such a pattern deals with several possibilities in one go. Written out in JAPE, the pattern is:

```
( |
  (Token.string == "Artikel")|{Token.string == "artikel"})
  |{Token.string == "Art"}{Token.string == ","})
  (Token.string == "art"){Token.string == "."})
 (Index)
 (Token.string == ",")
(Cite)
):ArticleLayered
```

As with the previous sections, we now need to modify the existing annotations to get the correct result. In this case, we need to add a new Cite annotation. This should have a kind attribute (with value artikel) and an artikel attribute (with the value of the index). Also, we need to copy any attributes of the lower level Cite (i.e. the attributes for the subparagraph and/or item) to this new Cite annotation. Then, we can delete the old Cite annotation.
Similar patterns can be used articles that include a CiteGroup or CiteRange sub layer. In those cases, the existing Cite, CiteFrom and CiteTo will need to be updated with the information that they are part of a reference to an article. Likewise, a reference to multiple articles, each of which may have a sub layer, can be dealt with in such a manner.

**B.5 Zooming-Out Layered References**

In addition to zooming-in layered references, there are also zooming-out layered references, as well as references that first zoom in, then zoom-out (see section 3.2), such as article 3, sub 3 of the Coin Act. If we first run the patterns as described above for all structure levels, then the different elements of such a reference will already be marked using Cite, CiteGroup and CiteRange annotations. In the case of the example given above, the annotations would be:

```
<Cite>article 3, sub 3</Cite> of <Cite>the Coin Act</Cite>
```

So, the pattern that is needed here (including the possibility of a comma to separate the two parts) amounts to:

```
{Cite} ({Token.string == ","})? {Token.string == "van"} {Cite}
```

Similar patterns should be added for a CiteGroup and CiteRange that are part of some higher level Cite. We can add a check to see whether the levels being merged are appropriate, which results in something like:

```
{
    {Cite.kind == "artikel"}|{Cite.kind == "paragraaf"}|
    {Cite.kind == "afdeling"}|{Cite.kind == "titel"}|
    {Cite.kind == "hoofdstuk"}|{Cite.kind == "deel"}|
    {Cite.kind == "boek"}
}

{Token.string == ","})? {Token.string == "van"}
{Cite.kind == "wet"}
```

So, we use the kind attribute that we added to each of the Cites, CiteGroups and CiteRanges to determine whether the first reference can actually be a subpart of the second reference.\(^{187}\)

After two references have been found that are related, they need to be replaced by a single Cite (or CiteGroup or CiteRange) that combines the attributes of the different parts. This can be done with code similar to that presented in the previous sections.

**B.6 Putting It All Together**

As was mentioned in the previous sections, the different patterns rely on other patterns being applied before them. The result is a multi-phase JAPE grammar, with a number of phases.

The first phase adds the temporary Index annotations. The second phase adds temporary Range annotations. After that, the actual detecting of references starts with a phase that detects references to list items: single references, references to ranges and group references. The next two phases combines two references that form a zooming-in reference from a list item to a sub list item.

\(^{187}\) This check is most likely superfluous, as a text like article 12 of article 28 is unlikely to appear in a legislative source.
Then follow the phases that detect references to the higher structure levels, starting with a phase that detects references to subparagraphs, including zooming-in references to sub-items (as discussed in section B.4). This phase is followed by similar phases at the level of articles, sections, etc. After that, phases follow to combine references that form a single zooming-out reference, starting at the level of list item and going up. Finally, there is a clean-up phase that removes any temporary annotations that still remain, and that apparently were not part of a reference.

In order to detect zooming-in and zooming-out references, this method orders the different structure layers in the way that is prescribed in the official guidelines (i.e. section is the lowest level, followed by division, title, chapter, part and book). This means that a multi-layered reference to such a layer in a document that does not follow this structure may not be recognised correctly. For example, when referring to a document which has chapters within a division, a reference to division 3, chapter 2 is a layered reference, while this approach will annotate it as two separate references. For this specific example, it would be better if the order was not prescribed in the grammar and instead we simply assumed that the first level mentioned was the higher level. After all, if the intention was to have two separate references, it would have been written as division 3 and chapter 2. However, this assumption does not always hold if the reference is part of a list, like division 3, chapters 2 and 3.

In general, though, such references seldom occur. Most references are aimed at the article level or below, and do not need to address the higher-level layers because the articles have been numbered uniquely throughout the document. Hence, the number of errors introduced by choosing one method over the other is very low. In a complete system, which includes a resolver which attempts to attach a URL to each reference found, it will be better to include the assumption mentioned above, as the resolver is more likely to detect errors introduced because of that assumption.