A Formalism for Universal Segmentation of Text

Julien Quint
GETA-CLIPS-IMAG, BP 53, F-38041 Grenoble Cedex 9, France
Xerox Research Centre Europe, 6, chemin de Maupertuis, F-38240 Meylan, France
e-mail: `julien.quint@imag.fr`

Abstract

Sumo is a formalism for universal segmentation of text. Its purpose is to provide a framework for the creation of segmentation applications. It is called "universal" as the formalism itself is independent of the language of the documents to process and independent of the levels of segmentation (e.g. words, sentences, paragraphs, morphemes...) considered by the target application. This framework relies on a layered structure representing the possible segmentations of the document. This structure and the tools to manipulate it are described, followed by detailed examples highlighting some features of Sumo.

Introduction

Tokenization, or word segmentation, is a fundamental task of almost all NLP systems. In languages that use word separators in their writing, tokenization seems easy: every sequence of characters between two whitespaces or punctuation marks is a word. This works reasonably well, but exceptions are handled in a cumbersome way. On the other hand, there are languages that do not use word separators. A much more complicated processing is needed, closer to morphological analysis or part-of-speech tagging. Tokenizers designed for those languages are generally very tied to a given system and language.

However, the gap becomes smaller when we look at sentence segmentation: a simplistic approach would not be sufficient because of the ambiguity of punctuation signs. And if we consider the segmentation of a document into higher-level units such as paragraphs, sections, and so on, we can notice that language becomes less relevant.

These observations lead to the definition of our formalism for segmentation (not just tokenization) that considers the process independently from the language. By describing a segmentation system formally, a clean distinction can be made between the processing itself and the linguistic data it uses. This entails the ability to develop a truly multilingual system by using a common segmentation engine for the various languages of the system; conversely, one can imagine evaluating several segmentation methods by using the same set of data with different strategies.

Sumo is the name of the proposed formalism, evolving from initial work by (Quint, 1999; Quint, 2000). Some theoretical works from the literature also support this approach: (Guo, 1997) shows that some segmentation techniques can be generalized to any language, regardless of their writing system. The sentence segmenter of (Palmer and Hearst, 1997) and the issues raised by (Habert et al., 1998) prove that even in English or French, segmentation is not so trivial. Lastly, (Aït-Mokhtar, 1997) handles all kinds of presyntactic processing in one step, arguing that there are strong interactions between segmentation and morphology.

1 The Framework for Segmentation

1.1 Overview

The framework revolves around the document representation chosen for Sumo, which is a layered structure, each layer being a view of the document at a given level of segmentation. These layers are introduced by the author of the segmentation application as needed and are not imposed by Sumo. The example in section 3.1 uses a two-layer structure (figure 4) corresponding to two levels of segmentation, characters and words. To extend this to a sentence segmenter, a third level for sentences is added.

These levels of segmentation can have a linguistic or structural level, but "artificial" levels can be introduced as well when needed. It is also interesting to note that several layers can belong to the same level. In the example of section 3.3, the result structure can have an indefinite number of levels, and all levels are of the same kind.

We call *item* the segmentation unit of a document at a given segmentation level (e.g. items of the word level are words). The document is then represented at every segmentation level in terms of its items; because segmentation is usually ambiguous, *item graphs* are used to factorize all the possible segmentations. Ambiguity issues are further addressed in section 2.3.

The main processing paradigms of Sumo are *identification* and *transformation*. With identification, new item graphs are built by identifying items from a source graph using a *segmentation resource*. These graphs are then modified by transformation processes. Section 2 gives the details about both identification and transformation.

1.2 Item Graphs

The item graphs are directed acyclic graphs; they are similar to the word graphs of (Amtrup et al., 1996) or the string graphs of (Colmerauer, 1970). They are actually represented by means of finite-state automata (see section 2.1). In order to facilitate their manipulation, two additional properties are enforced: these automata always have a single start-state and finite-state, and no dangling arcs (this is verified by pruning the automata after modifications). The examples of section 3 show various item graphs.

An item is an arc in the automaton. An arc is a complex structure containing a label (generally the surface form of the item), named attributes and relations. Attributes are used to hold information on the item, like part of speech tags (see section 3.2). These attributes can also be viewed as *annotations* in the same sense as the annotation graphs of (Bird et al., 2000).

1.3 Relations

Relations are links between levels. Items from a given graph are linked to items of the graph from which they were identified. We call the first graph the *lower* graph and the graph that was the source for the identification the *upper* graph. Relations exist between a path in the upper graph and either a path or a subgraph in the lower graph.

Figure 1 illustrates the first kind of relation, called *path relation*. This example in French is a relation between the two characters of the word "du" which is really a contraction of "de le".

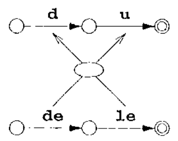

Figure 1: A path relation

Figure 2 illustrates the other kind of relation called *subgraph relation*. In this example the sentence "ABCDEFG." (we can imagine that A through G are Chinese characters) is related to several possible segmentations.

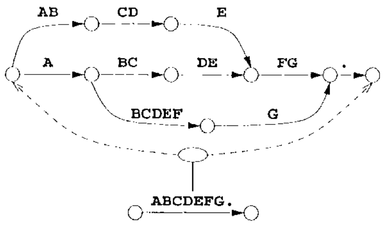

Figure 2: A graph relation

The interested reader may refer to (Planas, 1998) for a comparable structure (multiple layers of a document and relations) used in translation memory.

2 Processing a Document

2.1 Description of a Document

The core of the document representation is the item graph, which is represented by a finite-state automaton. Since regular expressions define finite-state automata, they can be used to describe an item graph. However, our expressions are extended because the items are more complex than simple symbols; new operators are introduced:

- attributes are introduced by an @ sign;
- path relations are delimited by { and };
- the information concerning a given item are parenthesized using [and].

As an exemple, the relation of figure 1 is described by the following expression:

```
[ de le { d u } ]
```

2.2 Identification

Identification is the process of identifying new items from a source graph. Using the source graph and a segmentation resource, new items are built to form a new graph. A segmentation resource, or simply resource, describes the vocabulary of the language, by defining a mapping between the source and the target level of segmentation. A resource is represented by a finite-state transducer in Sumo; identification is performed by applying the transducer to the source automaton to produce the target automaton, like in regular finite-state calculus.

Resources can be compiled by regular expressions or indentification rules. In the former case, one can use the usual operations of finite-state calculus to compile the resource: union, intersection, composition, etc.[1] A benefit of the use of Sumo structures to represent resources is that new resources can be built easily from the document that is being processed. (Quint, 1999) shows how to extract proper nouns from a text in order to extend the lexicon used by the segmenter to provide more acurate results.

In the latter case, rules are specified as shown in section 3.3. The left hand side of a rule describes a subpath in the source graph, while the right hand side describes the associated subpath in the target graph. A path relation is created between the two sequences of items. In an identification rule, one can introduce variables (for callback), and even calls to transformation functions (see next section). Naturally, these possibilities cannot be expressed by a strict finite-state structure, even with our extended formalism; hence, calculus with the resulting structures is limited.

A special kind of identification is the automatic segmentation that takes place at the entry point of the process. A character graph can be created automatically by segmenting an input text document, knowing its encoding. This text document can be in raw form or XML format. Another possibility for input is to use a graph of items that was created previously, either by Sumo, or converted to the format recognized by Sumo.

[1]The semantics of these operations is broadened to accomodate the more complex nature of the items.

2.3 Transformation

Ambiguity is a central issue when talking about segmentation. The absence or ambiguity of word separators can lead to multiple segmentations, and more than one of them can have a meaning. As (Sproat et al., 1996) testify, several native Chinese speakers do not always agree on one unique tokenization for a given sentence.

Thanks to the use of item graphs, Sumo can handle ambiguity efficiently. Why try to fully disambiguate a tokenization when there is no agreement on a single best solution? Moreover, segmentation is usually just a basic step of processing in an NLP system, and some decisions may need more information than what a segmenter is able to provide. An uninformed choice at this stage can affect the next stages in a negative way. Transformations are a way to modify the item graphs so that the "good" paths (segmentations) can be kept and the "bad" ones discarded. We can also of course provide full disambiguation (see section 3.1 for instance) by means of transformations.

In Sumo transformations are handled by transformation functions that manipulate the objects of the formalism: graphs, nodes, items, paths (a special kind of graph), etc. These functions are written using an imperative language illustrated in section 3.1. A transformation can either be applied directly to a graph or attached to a graph relation. In the latter case, the original graph is not modified, and its transformed counterpart is only accessible through the relation.

Transformation functions allow to control the flow of the process, using looping and conditionals. An important implication is that a same resource can be applied iteratively; as shown by (Roche, 1994) this feature allows to implement segmentation models much more powerful than simple regular languages (see section 3.3 for an example). Another consequence is that a Sumo application consists of one big transformation function returning the completed Sumo structure as a result.

3 Examples of Use

3.1 Maximum tokenization

Some classic heuristics for tokenization are classified by (Guo, 1997) under the collective moniker of *maximum tokenization*. This section describes how to implement a "maximum tokenizer" that tokenizes raw text documents in a given language and character encoding (e.g. English in ASCII, French in Iso-Latin-1, Chinese in Big5 or GB).

3.1.1 Common set-up

Our tokenizer is built with two levels: the input level is the character level, automatically segmented using the encoding information. The token level is built from these characters, first by an exhaustive identification of the tokens, then by reducing the number of paths to the one considered the best by the Maximum Tokenization heuristic.

The system works in three steps, with complete code shown in figure 3. First, the character level is created by automatic segmentation (lines 1-5, `input level` being the special graph that is automatically created from a raw file through stdin). The second step is to create the word graph by identifying words from character using a dictionary. A resource called `ABCdic` is created from a transducer file (lines 6-8), then the graph `words` is created by identifying items from the source level `characters` using the resource `ABCdic` (lines 9-12). The third step is the disambiguation of the word level by applying a Maximum Tokenization heuristic (line 13).

```
 1 characters: input level {
 2   encoding: <ASCII, UTF-8, Big5...>
 3   type: raw;
 4   from: stdin;
 5 }
 6 ABCdic: resource {
 7   file: ''ABCdic.sumo'';
 8 }
 9 words: graph <- identify {
10   source: characters;
11   resource: ABCdic;
12 }
13 words <- ft(words.start-node);
```

Figure 3: Maximum Tokenizer in Sumo

Figure 4 illustrates the situation for the input string "ABCDEFG" where A through G are characters and A, AB, B, BC, BCDEF, C, CD, D, DE, E, F, FG and G are words found in the resource `ABCdic`. The situation shown is after line 12 and before line 13.

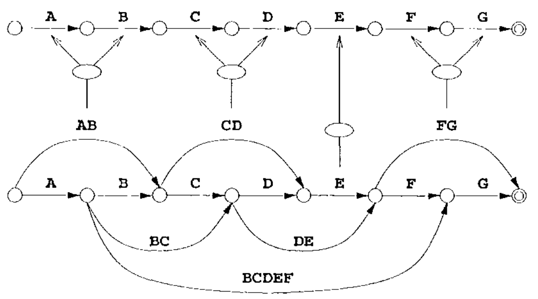

Figure 4: Exhaustive tokenization of the string ABCDEFG

We will see in the next three subsections the different heuristics and their implementations in Sumo.

3.1.2 Forward Maximum Tokenization

Forward Maximum Tokenization consists of scanning the string from left to right and selecting the token of maximum length any time an ambiguity occurs. On the example of figure 4, the result tokenization of the input string would be AB/CD/E/FG.

Figure 5 shows a function called `ft` that builds a path recursively by traversing the token graph, appending the longest item to the path at each node. `ft` takes a node as input and returns a path (line 1). If the node is final, the empty path is returned (lines 2-3), otherwise the array of items of the nodes (`n.items`) is searched and the longest item stored in `longest` (lines 4-10). The returned path consists of this longest item prepended to the longest path starting from the destination node of this item (line 11).

3.1.3 Backward Maximum Tokenization

Backward Maximum Tokenization is the same as Forward Maximum Tokenization except that the string is scanned from right to left, instead of left to right. On the example of figure 4, the tokenization of the input string would yield A/BC/DE/FG under Backward Maximum Tokenization.

A function `bt` can be written. It is very similar to `ft`, except that it works backward by looking at incoming arcs of the considered node. `bt` is called on the final state of the graph and

```
function ft (n: node) -> path {
  if final(n) {
    return ();
  } else {
    longest: item <- n.items[1];
    foreach it in n.items[2..] {
      if it.length > longest.length {
        longest <- it;
      }
    }
    return (longest # ft(longest.dest));
  }
}
```

Figure 5: The ft function

stops when at the initial node. Another implementation of this function is to apply `ft` on the reversed graph and then reversing the path obtained.

3.1.4 Shortest Tokenization

Shortest Tokenization is concerned with minimizing the overall number of tokens in the text. On the example of figure 4, the tokenization of the input string would yield A/BCDEF/G under shortest tokenization.

Figure 6 shows a function called `st` that finds the shortest path in the graph. This function is adapted from an algorithm for single-source shortest paths discovery in a DAG given by (Cormen et al., 1990). It calls another function, `t_sort`, returning a list of the nodes of the graph in topological order. The initializations are done in lines 2-6, the core of the algorithm is in the loop of lines 7-14 that computes the shortest path to every node, storing for each node its "predecessor". Lines 15-20 then build the path, which is returned in line 21.

3.1.5 Combination of Maximum Tokenization techniques

One of the features of Sumo is to allow the comparison of different segmentation strategies using the same set of data. As we have just seen, the three strategies described above can indeed be compared efficiently by modifying only part of the third step of the processing. Letting the system run three times on the same set of input documents can then give three different sets of results to be compared by the author of the system (against each other and against a reference tokenization, for instance).

```
 1 function st (g:graph) -> path {
 2   d: list <- ();  // distances
 3   p: list <- ();  // predecessors
 4   foreach n in (g.nodes) {
 5     d[n] = integer.max;  // ''infinite''
 6   }
 7   foreach n: node in t_sort(g.nodes) {
 8     foreach it in n.items {
 9       if (d[it.dest] > d[n] + 1) then {
10         d[it.dest] = d[n] + 1;
11         p[it.dest] = n;
12       }
13     }
14   }
15   n <- g.end;  // end state
16   sp: path <- (n);  // path
17   while (n != g.start) {
18     n = p[n];
19     sp = (n # sp);
20   }
21   return sp;
22 }
```

Figure 6: the st function

And yet a different set-up for our "maximum tokenizer" would be to select not just the optimal path according to one of the heuristics, but the paths selected by the three of them, as shown in figure 7. Combining the three paths into a graph is performed by changing line 13 in figure 3 to:

```
words <- ft(words.start-node) |
         bt(words.end-node) |
         st(words.start-node);
```

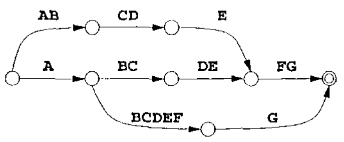

Figure 7: Three maximum tokenizations

3.2 Statistical Tokenization and Part of Speech Tagging

This example shows a more complicated tokenization system, using the same sort of set-up as the one from section 3.1, with a disambiguation process using statistics (namely, a bigram model). Our reference for this model is the Chasen Japanese tokenizer and part of speech tagger documented in (Matsumoto et al., 1999). This example is a high-level description of how to implement a similar system with Sumo.

The set-up for this example adds a new level to the previous example: the "bigram level." The word level is still built by identification using dictionaries, then the bigram level is built by computing a connectivity cost between each pair of tokens. This is the level that will be used for disambiguation or selection of the best solutions.

3.2.1 Exhaustive Segmentation

All possible segmentations are derived from the character level to create the word level. The resource used for this is a dictionary of the language that maps the surface form of the words (in terms of their characters) to their base form, part of speech, and a cost (Chasen also adds pronunciation, conjugation type, and semantic information). All this information is stored in the item as attributes, the base form being used as the label for the item. Figure 8 shows the identification of the word "cats" which is identified as "cat", with category "noun" (i.e. `@CAT=N`) and with some cost k (`@COST=k`).

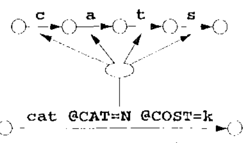

Figure 8: Identification of the word "cats"

3.2.2 Statistical Disambiguation

The disambiguation method relies on a bigram model: each pair of successive items has a "connectivity cost". In the bigram level, the "cost" attribute of an item W will be the connectivity cost of W and a following item X. Note that if a same W can be followed by several items X, Y, etc. with different connectivity costs for each pair, then W will be replicated with a different "cost" attribute. Figure 9 shows a word W followed by either X or Y, with two different connectivity costs k and k'.

The implementation of this technique in Sumo is straightforward. Assume there is a function `f` that, given two items, computes their connectivity cost (depending on both of their category, individual cost, etc.) and returns the first item

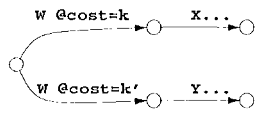

Figure 9: Connectivity costs for W

with its modified cost. We write the following rule and apply it to the word graph to create the bigram graph:

```
_ [$w1 = . @.] _ [$w2 = . @.]
  -> eval(f($w1, $2))
```

This rule can be read as: for any word `$w1` with any attribute ("`.`" matches any label, "`@.`" any set of attributes) followed by any word `$w2` with any attribute ("`_`" being a context separator), create the item returned by the function `f($w1, $w2)`.

Disambiguation is then be performed by selecting the path with optimal cost in this graph; but we can also select all paths with a cost corresponding to a certain threshold or the n best paths, etc. Note also that this model is easily extensible to any kind of n-grams. A new function `f($w1, ..., $wn)` must be provided to compute the connectivity costs of this sequence of items, and the above rule must be modified to take a larger context into account.

3.3 A Formal Example

This last example is more formal and serves as an illustration of some powerful features of Sumo. (Colmerauer, 1970) has a similar example implemented using Q systems. In both cases the goal is to transform an input string of the form $a^n b^n c^n, n \geq 0$ into a single item S (assuming that the input alphabet does not contain S), meaning that the input string is a word of this language.

The set-up here is once again to start with a lower level automatically created from the input, then to build intermediate levels until a final level containing only the item S is produced (at which point the input is recognized), or until the process can no longer carry on (at which point the input is rejected).

The building of intermediary levels is handled by the identification rule below:

```
# S? a [$A=a*] b [$B=b*] c [$c=c*] #
  -> S $A $B $C
```

What this rule does is identify a string of the form $S?aa^*bb^*cc^*$, storing all a's but the first one in the variable `$A`, all b's but the first one in `$B` and all c's but the first one in `$C`. The first triplet abc (with a possible S in front) is then absorbed by S, and the remaining a's, b's and c's are rewritten after S.

Figure 10 illustrates the first application of this rule to the input sequence $aabbcc$, creating the first intermediate level; subsequent applications of this rule will yield the only item S.

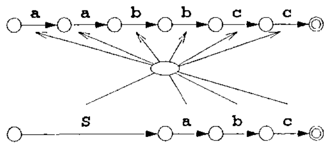

Figure 10: First application of the rule

Conclusion

We have described the main features of Sumo, a dedicated formalism for segmentation of text. A document is represented by item graphs at different levels of segmentation, which allows multiple segmentations of the same document at the same time. Three detailed examples illustrated the features of Sumo discussed here. For the sake of simplicity some aspects could not be evoked in this paper, they include: management of the segmentation resources, efficiency of the systems written in Sumo, larger applications, evaluation of segmentation systems.

Sumo is currently being prototyped by the author.

References

Salah AÏT-MOKHTAR, "Du texte ASCII au texte lemmatisé : la présyntaxe en une seule étape", in *Proceedings of TALN-97*, pages 60-69, Grenoble, France, June, 1997.

Jan W. AMTRUP, Henrik HEINE and Uwe JOST, *What's in a Word Graph. Evaluation and Enhancement of Word Lattices*, Verbmobil report 186, Universität Hamburg, `http://www.dfki.de/`, December, 1997.

Steven BIRD, David DAY, John GAROFOLO, John HENDERSON, Christophe LAPRUN and Mark LIBERMAN, "ATLAS: A Flexible and Extensible Architecture for Linguistic Annotation", in *Proceedings of LREC 2000*, Athens, Greece, May, 2000.

Alain COLMERAUER, *Les systèmes Q ou un formalisme pour analyser et synthétiser des phrases sur ordinateur*, Publication interne numéro 43, Université de Montréal, 1970.

Thomas H. CORMEN, Charles E. LEISERSON and Ronald L. RIVEST, *Introduction to Algorithms*, MIT Press, Cambridge, Massachussets, 1990.

Jin GUO, "Critical Tokenization and its Properties", in *Computational Linguistics*, 23(4), pages 569-596, December, 1997.

B. HABERT, G. ADDA, M. ADDA-DECKER, P. BOULA DE MARËUIL, S. FERRARI, O. FERRET, G. ILLOUZ and P. PAROUBEK, "Towards Tokenization Evaluation", in *Proceedings of LREC-98*, pages 427-431, 1998.

Yuji MATSUMOTO, Akira KITAUCHI, Tatsuo YAMASHITA et Yoshitaka HIRANO, *Japanese Morphological Analysis System ChaSen version 2.0 Manual*, Technical Report NAIST-IS-TR99009, Nara Institute of Science and Technology, Nara, April, 1999.

David D. PALMER and Marti A. HEARST, "Adaptative Multilingual Sentence Boundary Disambiguation", in *Computational Linguistics*, 23(2), pages 241-267, June, 1997.

Emmanuel PLANAS, *TELA. Structures et algorithmes pour la Traduction Fondée sur la Mémoire*, Thèse d'Informatique, Université Joseph Fourier, Grenoble, 1998.

Julien QUINT, "Towards a formalism for language-independent text segmentation", in *Proceedings of NLPRS'99*, pages 404-408, Beijing, November, 1999.

Julien QUINT, "Universal Segmentation of Text with the Sumo Formalism", im *Proceedings of NLP 2000*, pages 16-26, Patras, Greece, June, 2000.

Emmanuel ROCHE, "Two Parsing Algorithms by Means of Finite-State Transducers", in *Proceedings of COLING-94*, pages 431-435, 1994.

Richard SPROAT, Chilin SHIH, William GALE and Nancy CHANG, "A Stochastic Finite-State Word-Segmentation Algorithm for Chinese", in *Computational Linguistics* 22(3), pages 377-404, 1996.