

# Universal Segmentation of Text with the Sumo Formalism

## Abstract

We propose a universal formalism for the segmentation of text documents called Sumo. Its main purpose is to help creating segmentation systems for documents in any language. Because the processing is independent of the language, any level of segmentation (be it character, word, sentence, paragraph, etc.) can be considered. We will argue about the usefulness of such a formalism, describe the framework for segmentation on which Sumo relies, and give detailed examples to demonstrate some of its features.

## 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 formal-

ism, evolving from initial work by (Quint, 1999). 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, (A?-Mokhtar, 1997) handles all kinds of presyntactic processing in one step, arguing that there are strong interactions between segmentation and morphology.

## 1 A Framework for Segmentation

### 1.1 Overview

Sumo stores a document as a layered structure. Each layer of the structure is a view of the document at a given level of segmentation; the number and exact nature of each level of segmentation are not fixed by Sumo but defined by the author of the segmentation system. The example in section 3.1 uses a two-layer structure (figure 4) corresponding to two levels of segmentation, characters and words. A sentence segmenter built this way needs a third level for sentences.

The levels of segmentation do not have to have any linguistic or structural meaning, so that artificial levels can be introduced 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 2.2.

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 weighted finite-state automata (Mohri, 1996). In order to facilitate their manipulations, two additional properties are also enforced: these automata always have a single finite-state and no dangling arcs (this can be enforced by pruning the automata after every modification). 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), a weight, named attributes and relations. Attributes are used to hold information on the item, like part of speech tags (see section 3.2).

As is the case with finite-state automata nodes do not carry information *per se*, but the order of the outgoing arc is important as it allows to rank paths in the graph.

### 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 2 illustrates the other kind of relation

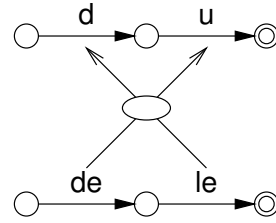


Figure 1: A path relation

called *subgraph relation*. In this example the sentence “ABCDEFGG.” (we can imagine that A through G are Chinese characters) is related to several possible segmentations.

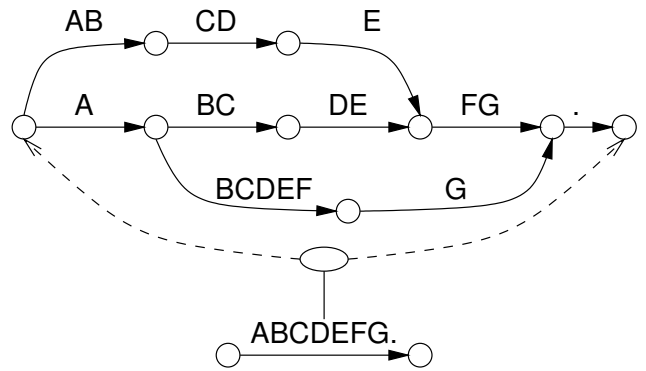


Figure 2: A graph relation

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 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. Finite-state transducers are used to store segmentation resources; identification is performed by applying the transducer to the source automaton to produce the target automaton.

Segmentation resources are a collection of identification rules, an example of which is

shown in section 3.3. The left hand side of the rule is a path in the source graph. The right hand side describes the associated path in the upper graph. A relation is created between these two paths; it is either a path relation (if the left hand side describes an item or a sequence of items) or a graph relation (if the left hand side describes item boundaries).

Since resources are represented as transducers, they can be combined to create new resources. Sumo uses the usual operators from finite-state calculus (e.g. union, intersection, composition, etc). Two more operators are added: the cascade operator that applies a sequence of transducers in order until one of them matches (this is similar to the union of several transducers, except that *all* transducers are applied in this case). The “star” or “iteration” operator, inspired from (Roche, 1994), applies the same transducer iteratively until a fixed point is reached (an application of this operation is found in the example of section 3.3). Another benefit of using transducers for segmentation resources is that resources can then be created from segmentation graphs during the processing of the document, as (Quint, 1999) details.

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.

## 2.2 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 seg-

menter 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. They can also be written in the same way than identification rule, with the difference that source and target levels are confounded; the target items replacing the source items.

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.

## 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.fst;
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 “ABCDEFGG” 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.

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).

```

1 function ft (n: node) -> path {
2   if final(n) {
3     return ();
4   } else {
5     longest: item <- n.items[1];
6     foreach it in n.items[2..] {
7       if it.length > longest.length {
8         longest <- it;
9       }
10    }
11    return (longest # ft(longest.dest));
12  }
13 }

```

Figure 5: The `ft` function

### 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.

Figure 6 shows a function called `bt` that 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 stops when a node with no incoming arcs is found (lines 2-3), which can only be the start node. Another implementation of this function would be to apply `ft` on the reversed graph and then reversing the path obtained.

```

1 function bt (n: node) -> path {
2   if n.incoming.length = 0 {
3     return ();
4   } else {
5     longest: item <- n.incoming[1];
6     foreach it in n.incoming[2..] {
7       if it.length > longest.length {
8         longest <- it;
9       }
10    }
11    return (bt(longest.src) # longest);
12  }
13 }

```

Figure 6: The `bt` function

### 3.1.4 Shortest Tokenization

Shortest Tokenization is concerned with minimizing the overall number of tokens in the text.

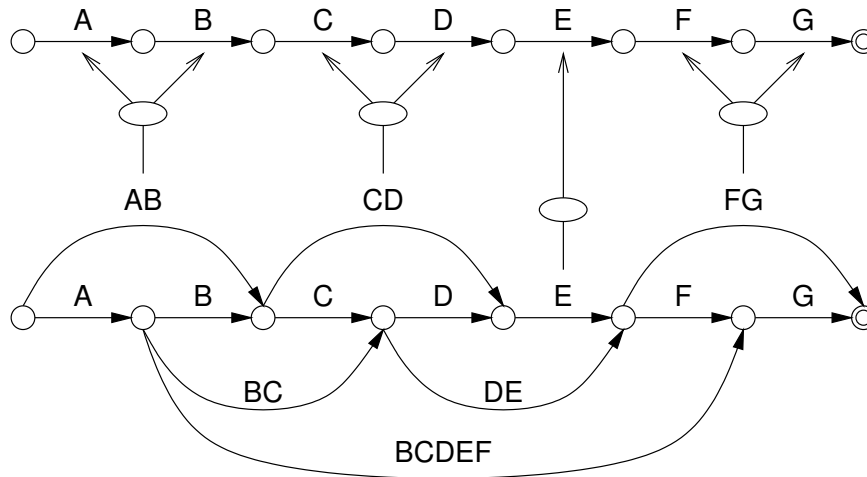


Figure 4: Exhaustive tokenization of the string ABCDEFG

On the example of figure 4, the tokenization of the input string would yield A/BCDEF/G under shortest tokenization.

Figure 7 shows a function called `st` that finds the shortest path in the graph. It works in a very similar way to `ft`, except that for each item starting from this node, we are not interested in the longest one but in the one leading to the path of minimum length (lines 4-14).

```

1 function st (n: node) -> path {
2   if final(n) {
3     return ();
4   } else {
5     p, mp: path;
6     i, mi: item;
7     mp <- st(n.items[1].dest)
8     foreach i in n.items[2..] {
9       p <- st(i.dest);
10      if p.length < mp.length {
11        mp <- p;
12        mi <- i;
13      }
14    }
15    return (mi # mp);
16  }
17 }

```

Figure 7: the `st` function

### 3.1.5 Combination of Maximum Tokenization techniques

One of the features of Sumo is to allow the comparison of different segmentation strategies us-

ing 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).

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, combining the three paths into a graph. We would then obtain the output graph shown in figure 8, by changing line 13 in figure 3 to:

```

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

```

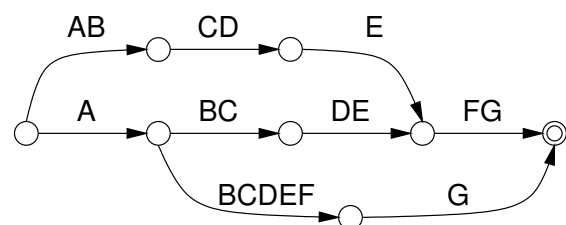


Figure 8: Three maximum tokenizations

### 3.2 Statistical Tokenization and Part of Speech Tagging

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

The basic set-up is the same as the one used previously: a level for the characters, from which a level for words is built by identification using dictionaries, then this level is disambiguated.

#### 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 surface form being used as the label for the item; the only exception is the cost, which is stored as a weight on an epsilon arc. Figure 9 shows the identification of the words “cats” which is identified as the noun “cat”, with cost 200.

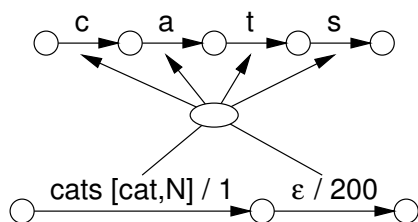


Figure 9: Identification of the word “cats”

#### 3.2.2 Statistical Disambiguation

The disambiguation method relies on a bigram model: each pair of successive items has a “connectivity cost”. These costs are encoded with an epsilon arc between two successive items with a weight reflecting the connectivity cost and the cost of the morpheme itself (figure 10). Since Sumo manipulates weighted finite-state

automata, it provides functions for computing the cost of a path and can rank all the possible paths according to their cost. Disambiguating the output is choosing the path with optimal cost, but we may rather want to select all solutions above a given threshold, or the  $n$  best ones.

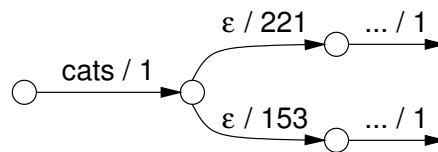


Figure 10: Connectivity costs

### 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\* \_ b \_ b\* \_ c \_ c\* # -> S

The left part is the path to be matched in the lower graph, described by a regular expression. The # symbol indicates the beginning or the end of the lower graph. The \_ symbol is used to distinguish the item being identified from its context.

What this rule does is identify a string of the form  $S?aa^*bb^*cc^*$  and replace the first triplet  $abc$  (with a possible  $S$  in front) with  $S$ , keeping the context intact. Figure 11 illustrates the first application of this rule to input  $aabbcc$ , creating the first intermediate level. This rule is then applied again to this level to create a new inter-

mediary level, and so on until it cannot be applied anymore, using or “star” operator on this resource.

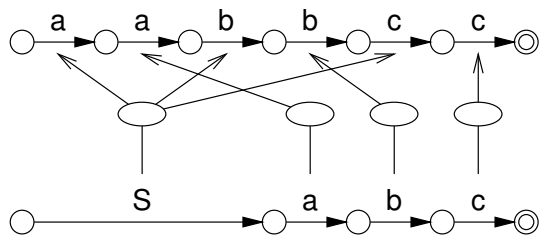


Figure 11: First application of the rule

## 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 implemented by the author.

## References

- Salah A?-MOKHTAR, “Du texte ASCII au texte lemmatis?: la pr?yntaxe en une seule ?ape”, 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, Universit? Hamburg, <http://www.dfki.de/>, December, 1997.
- Alain COLMERAUER, *Les syst?es Q ou un formalisme pour analyser et synth?iser des phrases sur ordinateur*, Publication interne num?o 43, Universit?de Montr?l, 1970.
- 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 MAR?IL, 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.

Mehryar MOHRI, Fernando PEREIRA and Michael RILEY, “Weighted Automata in Text and Speech Processing”, in *Proceedings of the ECAI 96 Workshop*, pages 46-50, 1996.

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 Fond? sur la M?oire*, Th?e d’Informatique, Universit?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.

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.