By all these lovely tokens...*
Merging Conflicting Tokenizations

Christian Chiarcos, Julia Ritz and Manfred Stede
Sonderforschungsbereich 632 "Information Structure"
University of Potsdam
Karl-Liebknecht-Str. 24-25, 14476 Golm, Germany
{chiarcos|julia|stede}@ling.uni-potsdam.de

Abstract

Given the contemporary trend to modular NLP architectures and multiple annotation frameworks, the existence of concurrent tokenizations of the same text represents a pervasive problem in everyday's NLP practice and poses a non-trivial theoretical problem to the integration of linguistic annotations and their interpretability in general. This paper describes a solution for integrating different tokenizations using a standoff XML format, and discusses the consequences for the handling of queries on annotated corpora.

1 Motivation

1.1 Tokens: Functions and goals

For most NLP tasks and linguistic annotations, especially those concerned with syntax (part-of-speech tagging, chunking, parsing) and the interpretation of syntactic structures (esp., the extraction of semantic information), tokens represent the **minimal unit of analysis**: words (lexemes, semantic units, partly morphemes) on the one hand and certain punctuation symbols on the other hand. From a corpus-linguistic perspective, tokens also represent the **minimal unit of investigation**, the minimal character sequence that can be addressed in a corpus query (e.g. using search tools like TIGERSearch (König and Lezius, 2000) or CWB (Christ, 1994)). Tokens also constitute the basis for 'word' **distance** measurements. In many annotation tools and their corresponding formats, the order of tokens provides a **timeline** for the sequential order of *structural* elements (MMAX (Müller and Strube, 2006), GENAU (Rehm et al., 2009), GrAF (Ide and Suderman, 2007), TIGER XML (König and Lezius, 2000)). In several multi-layer formats, tokens also define the **absolute position** of annotation elements, and only by reference to a common token layer, annotations from different layers can be related with each other (NITE (Carletta et al., 2003), GENAU).

Thus, by their function, tokens have the following characteristics: (i) tokens are totally ordered, (ii) tokens cover the full (annotated portion of the) primary data, (iii) tokens are the smallest unit of annotation, and (iv) there is only one single privileged token layer. The last aspect is especially relevant for the study of richly annotated data, as an integration and serialization of annotations produced by different tools can be established only by reference to the token layer. From a corpus-linguistic perspective, i.e., when focusing on querying of annotated corpora, tokens need to be well-defined and all information annotated to a particular text is to be preserved without any corruption. We argue that for this purpose, characteristic (iii) is to be abandoned, and we will describe the data format and an algorithm for merging different tokenizations and their respective annotations.

Our goal is a fully automated merging of annotations that refer to different tokenizations (henceforth T_1 and T_2) of the same text. We regard the following criteria as crucial for this task:

Information preservation. All annotations applied to the original tokenizations should be preserved.

Theoretically well-defined notion of token. It should be possible to give a plausible list of positive criteria that define character sequences as tokens. Knowledge about the token definition is essential for formulating queries for words, e.g. in a corpus search interface.

Integrative representation. All annotations that are consistent with the merged tokenization should refer to the merged tokenization. This is necessary in order to query across multiple annotations orig-

*Taken from the poem *September* by Helen Hunt Jackson.

Proceedings of the Third Linguistic Annotation Workshop, ACL-IJCNLP 2009, pages 35–43,
Suntec, Singapore, 6-7 August 2009.

inating from different annotation layers or tools.

Unsupervised merging. The integration of conflicting tokenizations should not require manual interference.

1.2 Tokenization

Tokenization is the process of mapping sequences of characters to sequences of words (cf. Guo 1997). However, different research questions or applications induce different conceptions of the term 'word'. For a *shallow morphosyntactic analysis* (part of speech tagging), a 'simple' tokenization using whitespaces and punctation symbols as delimiters seems acceptable for the examples in (1). A *full syntactic analysis* (parsing), however, could profit from the aggregation of complex nominals into one token each.

(1) a. department store
b. Herzog-von der Heide[1]
c. Red Cross/Red Crescent movement

Similarly, examples (2a) and (2b) can be argued to be treated as one token for *(morpho)syntactic analyses*, respectively. Despite intervening whitespaces and punctuation symbols, they are complex instances of the 'classical' part-of-speech *adjective*. For certain *semantic analyses* such as in information extraction, however, it may be useful to split these compounds in order to access the inherent complements (*E 605, No. 22*).

(2) a. E 605-intoxicated
b. No. 22-rated

Finally, (3) illustrates a *morphology-based* tokenization strategy: the principle of splitting at morpheme boundaries (Marcus et al., 1993, PTB) (token boundaries represented by square brackets). Morphological tokenization may help distributional (co-occurrence-based) semantics and/or parsing; however, the resulting tokens might be argued as being less intuitive to users of a corpus search tool.

(3) a. [Mitchell]['s], [they]['ve], [do][n't]
b. [wo][n't], [ca][n't], [ai][n't]

These examples show that different applications (tagging, parsing, information extraction) and the focus on different levels of description (morphology, syntax, semantics) require specialized tokenization strategies. When working with multiple tools for standard NLP tasks, thus, it is the norm rather than the exception that they disagree in their tokenization, as shown in ex. (4).

(4) doesn't
a. [does][n't] (Marcus et al., 1993, PTB)
b. [doesn]['][t] (Brants, 2000, TnT)

When creating a corpus that is annotated at multiple levels and/or using several tools, different tokenizations are not always avoidable, as some tools (automatic NLP tools, but also tools for manual annotation) have integrated tokenizers. Another challenge is the representation of token boundaries. Commonly, token boundaries are represented by a line break ('\n') or the whitespace 'character' (' ') – in which case token-internal whitespaces are replaced, usually by an underscore ('_') –, thereby corrupting the original data. This practice makes reconciling/merging the data a difficult enterprise.

Given this background, we suggest an XML-based annotation of token boundaries, such that token boundaries are marked without affecting the original primary data. In a straightforward XML model, tokens are represented by XML elements enclosing primary text slices (c.f. the BNC encoding scheme (Burnard, 2007)). However, treating tokens as spans of text by means of the XML hierarchy is impossible for tokenization conflicts as in (4.a) and (4.b).

2 Conflicting tokenizations: Straightforward strategies

By 'straightforward strategies', we mean approaches that aim to preserve the definition of tokens as atomic, minimal, unambiguous units of annotation when unifying different tokenizations (henceforth T_1 and T_2) of the same text. By 'unsupervised straightforward strategies', we mean tokenization strategies that operate on the primary data only, without consulting external resources such as dictionaries or human expertise.

Unsupervised straightforward strategies to the task include:

1. no merging In a conservative approach, we could create independent annotation projects for every tokenization produced, and thus represent all tokenizations independently. This, however, rules out any integration or combined evaluation of annotations to T_1 and annotations to T_2.

[1] Double surname consisting of *Herzog* and *von der Heide*.

2. normalization Adopt one of the source tokenizations, say T_1, as the 'standard' tokenization. Preserve *only* the information annotated to T_2 that is consistent with T_1. Where tokenization T_2 deviates from T_1, all annotations to T_2 are lost.[2]

3. maximal tokens For every token boundary in T_1 that is also found in T_2, establish a token boundary in the merged tokenization (cf. Guo's 1997 'critical tokenization'). However, with tokens assumed to be the minimal elements of annotation, we lose linguistic analyses of fine-grained tokens. With respect to (4.a) and (4.b), the maximal token would be the whole phrase *doesn't*. Again, this results in a loss of information, as all annotations applied to *does*, *doesn*, *n't*, *'* and *t* refer to units that are smaller than the resulting token.

4. maximal common substrings For every token boundary in T_1 *or* T_2, establish a token boundary, thereby producing **minimal tokens**: one token for every maximal substring shared between T_1 and T_2 (cf. Guo's 1997 'shortest tokenization'). By defining the original tokens ('supertokens') as annotations spanning over tokens, all annotations are preserved. However, the concept of 'token' loses its theoretical motivation; there is no guarantee that maximal common substrings are meaningful elements in any sense: The maximum common substring tokenization of 4.a and 4.b is *[does][n]['][t]*, but *[n]* is not a well-defined token. It is neither defined with respect to morphology (like PTB tokens) nor is it motivated from orthography (like TnT tokens), but it is just the remainder of their intersection.

As shown in Table 1, none of the strategies sketched above fulfills all criteria identified in Section 1.1: Avoiding a merging process counteracts data integration; token normalization and maximal tokens violate information preservation, and maximal common substrings violate the requirement to specify a theoretically well-defined notion of token.

As an alternative, we propose a formalism for the lossless integration and representation of conflicting tokenizations by abandoning the assumption that tokens are an atomic, primitive concept that represents the minimal unit of annotation. Rather, we introduce annotation elements smaller than the actual token – so-called *terminals* or *term*s for short – that are defined according to the maximum common substrings strategy described above.

Table 1: Deficits of 'straightforward' merging approaches

	no merge	normalize	max. tokens	max. common substrings
information preservation	+	−	−	+
well-defined tokens	+	+	(−)	−
integrative	−	+	+	+
unsupervised	(+)	+	+	+

Then, tokens are defined as nodes that span over a certain range of terms similar to phrase nodes that dominate other nodes in syntax annotations. The representation of conflicting tokenizations, then, requires a format that is capable to express conflicting hierarchies. For this purpose, we describe an extension of the PAULA format, a generic format for text-oriented linguistic annotations based on standoff XML.

3 Conflicting tokenizations in the PAULA format

3.1 Annotation structures in PAULA 1.0

The PAULA format (Dipper, 2005; Dipper and Götze, 2005) is a generic XML format, used as a pivot format in NLP pipelines (Stede et al., 2006) and in the web-based corpus interface ANNIS (Chiarcos et al., 2008). It uses standoff XML representations, and is conceptually closely related to the formats NITE XML (Carletta et al., 2003) and GraF (Ide and Suderman, 2007).

PAULA was specifically designed to support the lossless representation of different types of text-oriented annotations (layer-based/timeline annotations, hierarchical annotations, pointing relations), optimized for the annotation of multiple layers, including conflicting hierarchies and simple addition/deletion routines for annotation layers. Therefore, primary data is stored in a separate

[2] Alternatively, transformation rules to map annotations from T_2 to T_1 would have to be developed. This does, however, not guarantee information preservation, and, additionally, it requires manual work, as such transformations are annotation-specific. Thus, it is not an option for the fully automated merging of tokenizations.

Table 2: PAULA 1.0 data types

nodes (structural units of annotation)

- **token** character spans in the primary data that form the basis for higher-level annotation
- **markable** (spans of) token(s) that can be annotated with linguistic information. Markables represent flat, layer-based annotations defined with respect to the sequence of tokens as a general timeline.
- **struct** hierarchical structures (DAGs or trees) are formed by establishing a dominance relation between a struct (e.g., a phrase) node as parent, and tokens, markables, or other struct nodes as children.

edges (relational units of annotation, connecting tokens, markables, structs)

- **dominance relation** directed edge between a struct and its children
- **pointing relations** directed edge between nodes in general (tokens, markables, structs)

labels (annotations: node or edge labels)

- **features** represent annotations attached to a particular (structural or relational) unit of annotation

file. Multiple annotations are also stored in separate files to avoid interference between concurrent annotations. Annotations refer to the primary data or to other annotations by means of XLinks and XPointers.

As types of linguistic annotation, we distinguish nodes (token, markable, struct), edges (dominance and pointing relations) and labels (annotations), as summarized in Table 2. Each type of annotation is stored in a separate file, so that competing or ambiguous annotations can be represented in an encapsulated way.

PAULA 1.0 is already sufficiently expressive for capturing the data-heterogeneity sketched above, including the representation of overlapping segments, intersecting hierarchies, and alternative annotations (e.g., for ambiguous annotations), but only for annotations *above* the token level. Further, PAULA 1.0 relies on the existence of a unique layer of non-overlapping, atomic tokens as minimal units of annotation: For all nodes, their position and sequential order is defined with respect to the absolute position of tokens that they cover; and for the special case of markables, these are defined solely in terms of their token range.

Finally, PAULA 1.0 tokens are *totally ordered*, they *cover* the (annotated) primary data *completely*, and they are *non-overlapping*. Only on this basis, the extension and (token-)distance of annotated elements can be addressed; and only by means of unambiguous reference, information from different layers of annotation can be combined and evaluated.

3.2 Introducing terminal nodes

In our extension of the PAULA format, we introduce the new concept of *term* nodes: atomic terminals that directly point to spans of primary data. *Terms* are subject to the same constraints as tokens in PAULA 1.0 (total order, full coverage, non-overlapping). So, terms can be used in place of PAULA 1.0 *tokens* to define the extension and position of super-token level and sub-token level annotation elements.

Markables are then defined with respect to (spans of) terminal nodes rather than tokens, such that alternative tokenizations can be expressed as markables in different layers that differ in their extensions.

Although terms adopt several functions formerly associated with tokens, a privileged token layer is still required: In many query languages, including ANNIS-QL (Chiarcos et al., 2008), tokens define the application domain of regular expressions on the primary data. More importantly, tokens constitute the basis for conventional ("word") distance measurements and ("word") coverage queries. Consequently, the constraints on tokens (total order, full coverage and absence of overlap) remain.

The resulting specifications for structural units of annotation are summarized in Table 3. Distinguishing terminal elements and re-defining the token layer as a privileged layer of markables allows us to disentangle the technical concept of 'atomic element' and 'token' as the conventionally assumed minimal unit of linguistic analysis.

3.3 A merging algorithm

In order to integrate annotations on tokens, it is not enough to represent two tokenizations side by side with reference to the same layer of terminal nodes. Instead, a privileged token layer is to be established and it has to be ensured that annotations can be queried *with reference to the token layer*.

Table 3: PAULA extensions: revised node types

terms	specify character spans in the primary data that form the basis for higher-level annotation
markable	defined as above, with terms taking the place of tokens
structs	defined as above, with terms taking the place of tokens
tokens	sub-class of structs that are non-overlapping, arranged in a total order, and cover the full primary data

Then, all annotations whose segmentation is consistent with the privileged token layer are directly linked with tokens.

Alg. 3.1 describes our merging algorithm, and its application to the four main cases of conflicting tokenization is illustrated in Figure 1.[3] The following section describes its main characteristics and the consequences for querying.

4 Discussion

Alg. 3.1 produces a PAULA project with one single tokenization. So, it is possible to define queries spanning across annotations with originally different tokenization:

Extension and **precedence** queries are tokenization-independent: Markables refer to the $term$ layer, not the tok layer, structs also (indirectly) dominate $term$ nodes.

Dominance queries for struct nodes and tokens yield results whenever the struct node dominates only nodes with tok-compatible source tokenization: Structs dominate tok nodes wherever the original tokenization was consistent with the privileged tokenization tok (case A and C in Fig. 1).

Distance queries are defined with respect to the tok layer, and are applicable to all elements that are are defined with reference to the tok layer (in figure 1: tok_{1a}, tok_{2a}, tok_{1b}, tok_{2b} in case A; tok_{ab} in case B; tok_{a}, tok_{b}, tok_{ab} in case C; tok_{ab}, tok_{c} in case D). They are not applicable to elements that do not refer to the tok layer (B: tok_{a}, tok_{b}; D: tok_{a}, tok_{bc}).

[3] Notation: $prim$ – primary data / $tok, term$ – annotation layers / $t \in L$ – t is a node on a layer L / $a..b$ – continuous span from tok/term a to tok/term b / a, b – list of tok/term/markable nodes a, b / $t = [a]$ – t is a node (struct, markable, tok) that points to a node, span or list a

The algorithm is unsupervised, and the token concept of the output tokenization is well-defined and consistent (if one of the input tokenizations is adopted as target tokenization). Also, as shown below, it is integrative (enabling queries across different tokenizations) and information-preserving (reversible).

4.1 Time complexity

After a PAULA project has been created, the time complexity of the algorithm is quadratic with respect to the number of characters in the primary data n. This is due to the total order of tokens: Step 2 and 3.a are applied once to all original tokens from left to right. Step 5 can be reformulated such that for every *terminal node*, the relationship between the directly dominating tok_1 and tok_2 is checked. Then, Step 5 is also in $O(n)$. In terms of the number of markables m, the time complexity in Step 3.b is in $O(n\ m)$: for every markable, the corresponding $term$ element is to be found, taking at most n repositioning operations on the $term$ layer. Assuming that markables within one layer are non-overlapping[4] and that the number of layers is bound by some constant c[5], then $m \leq n\ c$, so that 3.b is in $O(n^2\ c)$.

For realistic scenarios, the algorithm is thus quadratic.

4.2 Reversibility

The merging algorithm is reversible – and, thus, lossless – as shown by the splitting algorithm in Alg. 3.2. For reasons of space, the correctness of this algorithm cannot be demonstrated here, but broadly speaking, it just removes every node that corresponds to an original token of the 'other' tokenization, plus every node that points to it, so that only annotations remain that are directly applied to the target tokenization.

4.3 Querying merged tokenizations

We focus in this paper on the merging of analyses with different tokenizations for the purpose of users *querying a corpus across multiple annota-*

[4] Although PAULA supports overlapping markables within one single layer, even with identical extension, this is a reasonable assumption: In practice, overlapping markables within one single layer are rare. More often, there is even a longer sequence of primary data between one markable of a particular layer and the next. In our experience, such 'gaps' occur much more often than overlapping markables.

[5] Again, this is a practical simplication. Theoretically, the number of layers is infinite.

Alg. 3.1 Merging different tokenizations

0. assume that we have two annotations $analysis_1$ and $analysis_2$ for the same primary data, but with different tokenizations

1. create PAULA 1.0 annotation projects for $analysis_1$ and $analysis_2$ with primary data files $prim_1$ and $prim_2$ and token layers tok_1 and tok_2 respectively.

2. harmonize primary data
   if $prim_1$ equals $prim_2$, then
   (i) rename $prim_1$ to $prim$
   (ii) set all references in $analysis_2$ from $prim_2$ to $prim$
   (iii) create a new annotation project $analysis$ by copying $prim$ and all annotation layers from $analysis_1$ and $analysis_2$

   otherwise terminate with error msg

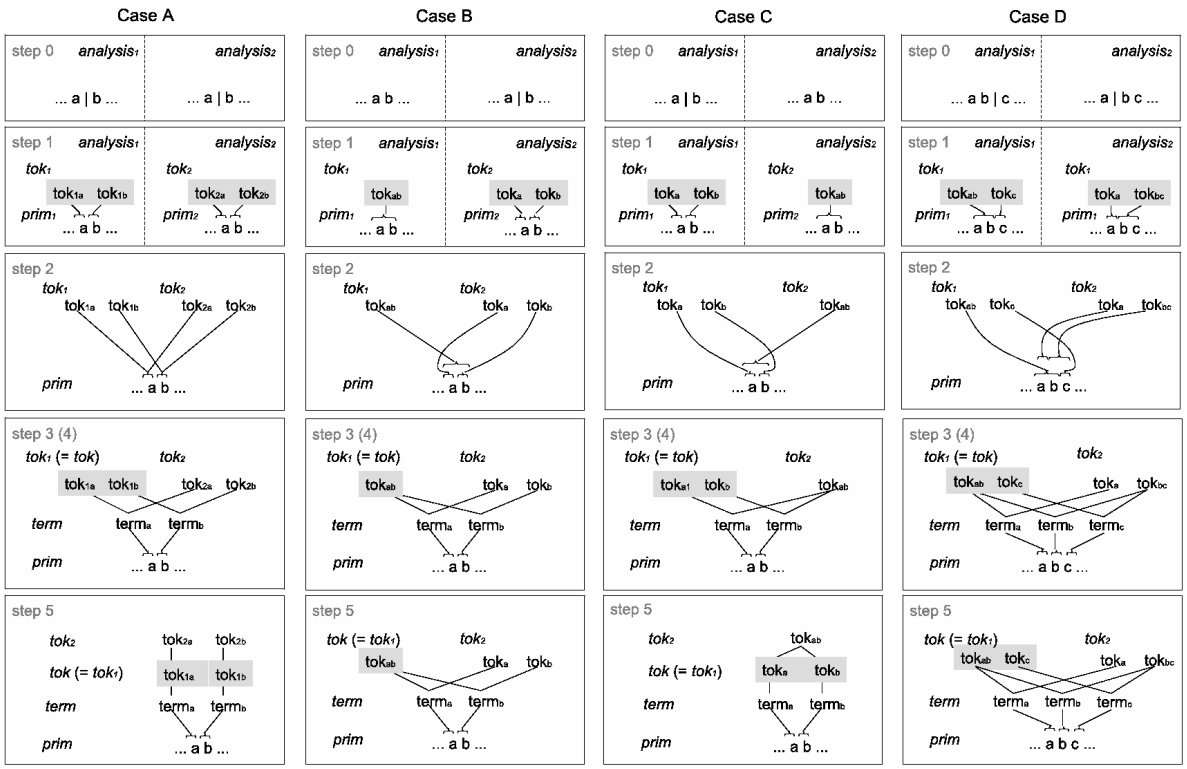
3. harmonize terminal nodes
   create a new annotation layer $term$, then
   (a) for all overlapping tokens $t_1 \in tok_1$ and $t_2 \in tok_2$: identify the maximal common substrings of t_1 and t_2
   for every substring s, create a new element $term_s$ pointing to the corresponding character span in the primary data
   for every substring s, redefine t_1 and t_2 as markables referring to $term_s$

   (b) redefine markable spans as spans of terminal nodes
   for every token $t = [term_{s_1}..term_{s_2}] \in tok_1 \cup tok_2$ and every markable $m = [w..xty..z]$: set $m = [w..xterm_{s_1}..term_{s_2}y..z]$

4. select token layer
   rename tok_1 to tok, or rename tok_2 to tok, (cf. the normalization strategy in Sect. 2) or
   rename $term$ to tok (cf. the minimal tokens strategy in Sect. 2)

5. token integration
   for every original token $ot = [a..b] \in (tok_1 \cup tok_2) \setminus tok$:
   if there is a token $t \in tok$ such that $t = [a..b]$, then define ot as a struct with $ot = [t]$, else
   if there are tokens $t_1, .., t_n \in tok$ such that $t_1..t_n$ form a continuous sequence of tokens and $t_1 = [a..x]$ and $t_n = [y..b]$, then define ot as a struct such that $ot = [t_1, .., t_n]$,
   otherwise: change nothing

Figure 1: Merging divergent tokenizations



Alg. 3.2 Splitting a PAULA annotation project with two different tokenizations

0. given a PAULA annotation project $analysis$ with token layer tok, terminal layer $term$, and two layers l_1 and l_2 (that may be identical to $term$ or tok) that convey the information of the original token layers tok_1 and tok_2
1. create $analysis_1$ and $analysis_2$ as copies of $analysis$
2. if l_1 represents a totally ordered, non-overlapping list of nodes that cover the primary data completely, then modify $analysis_1$:
   a. for every node in l_1: substitute references to tok_1 by references to $term_1$
   b. remove l_2 from $analysis_1$
   c. if $l_1 \neq tok_1$, remove tok_1 from $analysis_1$
   d. for every annotation element (node/relation) e in $analysis_1$ that directly or indirectly points to another node in $analysis_1$ that is no longer present, remove e from $analysis_1$
   e. remove every annotation layer from $analysis_1$ that does not contain an annotation element
   f. for every markable in l_1: remove references to $term_1$, define the extension of l_1 nodes directly in terms of spans of text in $prim_1$
   g. if $l_1 \neq term_1$, remove $term_1$
3. perform step 2. for l_2 and $analysis_2$

Alg. 3.3 Iterative merging: modifications of Alg. 3.1, step.3

if $analysis_1$ has a layer of terminal nodes $term_1$, then let $T_1 = term_1$, otherwise $T_1 = tok_1$
if $analysis_2$ has a layer of terminal nodes $term_2$, then let $T_2 = term_2$, otherwise $T_2 = tok_2$
create a new annotation layer $term$, then

1. for all overlapping terminals/tokens $t_1 \in T_1$ and $t_2 \in T_2$: identify the maximal common substrings of t_1 and t_2
   for every substring s, create a new element $term_s$ pointing to the corresponding character span in the primary data
   for every substring s, redefine t_1 and t_2 as markables referring to $term_s$
2. redefine markable spans as spans of terminal nodes
   for every node $t = [term_{s_1}..term_{s_2}] \in T_1 \cup T_2$ and every markable $m = [w..xty..z]$: set $m = [w..xterm_{s_1}..term_{s_2}y..z]$
3. for all original terminals $t \in T_1 \cup T_2$: if t is not directly pointed at, remove t from $analysis$

tion layers. Although the merging algorithm produces annotation projects that allow for queries integrating annotations from analyses with different tokenization, the structure of the annotations is altered, such that the behaviour of merged and unmerged PAULA projects may be different. Obviously, token-level queries must refer to the privileged tokenization T_1. Operators querying for the relative **precedence or extension** of markables are not affected: in the merged annotation project, markables are defined with reference to the layer $term$: originally co-extensional elements E_1 and E_2 (i.e. elements covering the same tokens in the source tokenization) will also cover the same *terminals* in the merged project. **Distance operators** (e.g. querying for two tokens with distance 2, i.e. with two tokens in between), however, will operate on the new privileged tokenization, such that results from queries on $analysis$ may differ from those on $analysis_2$. **Dominance operators** are also affected, as nodes that directly dominated a token in $analysis_1$ or $analysis_2$ now indirectly dominate it in $analysis$, with a supertoken as an intermediate node.

Accordingly, queries applicable to PAULA projects *before* the merging are not directly applicable to merged PAULA projects. Users are to be instructed to keep this in mind and to be aware of the specifications for the merged tokenization and its derivation.[6]

5 Extensions

5.1 Merging more than two tokenizations

In the current formulation, Alg. 3.1 is applied to two PAULA 1.0 projects and generates extended PAULA annotation projects with a $term$ layer. The algorithm, however, may be applied iteratively, if step 3 is slightly revised, such that extended PAULA annotation projects can also be merged, see Alg. 3.3.

5.2 Annotation integration

The merging algorithm creates a struct node for every original token. Although this guarantees reversibility, one may consider to remove such redundant structs. Alg. 3.4 proposes an optional postprocessing step for the merging algorithm. This step is optional because these operations are

[6] The information, however, is preserved in the format and may be addressed by means of queries that, for example, operate on the extension of terminals.

Alg. 3.4 Annotation integration: Optional post-processing for merging algorithm

6.a. remove single-token supertoken
for every original token $ot = [t] \in tok_1 \cup tok_2$ with $t \in tok$: replace all references in $analysis$ to ot by references to t, remove ot

6.b. merging original token layers tok_1 and tok_2 (if $tok_1 \neq tok$ and $tok_2 \neq tok$)
define new 'super token' layer $stok$.
for every $ot \in tok_1 \cup tok_2$:

if $ot = [t]$ for some $t \in tok$, then see 6.a

if $ot = [t_1, .., t_n]$ for some $t_1, .., t_n \in tok$, and there is $ot_2 = [t_1, .., t_n] \in tok_1 \cup tok_2 \cup stok$, then replace all references in $analysis$ to ot_2 by references to ot, move ot to layer $stok$, remove ot_2 from $analysis$

move all remaining $ot \in tok_1 \cup tok_2$ to $stok$, remove layers tok_1 and tok_2

6.c. unify higher-level annotations
for every markable $mark_1 = [term_1..term_n]$ and $term_1, .., term_n \in term$:

if there is a markable $mark_2$ in $analysis$ such that $mark_2 = [term_1..term_n]$, then replace all references in $analysis$ to $mark_2$ by references to $mark_1$, remove $mark_2$

for every struct $struct_1 = [c_1, .., c_n]$ that covers exactly the same children as another struct $struct_2 = [c_1, .., c_n]$, replace all references to $struct_2$ by references to $struct_1$, remove $struct_2$

destructive: We lose the information about the origin ($analysis_1$ vs. $analysis_2$) of $stok$ elements and their annotations.

6 Summary and Related Reasearch

In this paper, we describe a novel approach for the integration of conflicting tokenizations, based on the differentiation between a privileged layer of tokens and a layer of atomic terminals in a standoff XML format: Tokens are defined as structured units that dominate one or more terminal nodes.

Terminals are atomic units only *within* the respective annotation project (there is no unit addressed that is smaller than a terminal). By iterative applications of the merging algorithm, however, complex terms may be split up in smaller units, so that they are not atomic in an absolute sense.

Alternatively, terms could be identified a priori with the minimal addressable unit available, i.e., characters (as in the formalization of tokens as *charspan*s and *charseq*s in the ACE information extraction annotations, Henderson 2000). It is not clear, however, how a character-based term definition would deal with sub-character and zero extension terms: A character-based definition of terms that represent traces is possible only by corrupting the primary data.[7] Consequently, a character-based term definition is insufficient unless we restrict ourselves to a particular class of languages, texts and phenomena.

The role of terminals can thus be compared to timestamps: With reference to a numerical timeline, it is always possible to define a new event between two existing timestamps. Formats specifically designed for time-aligned annotations, e.g., EXMARaLDA (Schmidt, 2004), however, typically lack a privileged token layer and a formal concept of tokens. Instead, tokens, as well as longer or shorter sequences, are represented as markables, defined by their extension on the timeline.

Similarly, GrAF (Ide and Suderman, 2007), although being historically related to PAULA, does not have a formal concept of a privileged token layer in the sense of PAULA.[8] We do, however, assume that terminal nodes in GrAF can be compared to PAULA 1.0 tokens.

For conflicting tokenizations, Ide and Suderman (2007) suggest that 'dummy' elements are defined covering all necessary tokenizations for controversially tokenized stretches of primary data. Such dummy elements combine the possible tokenizations for strategies 1 (no merging) and 3 (maximal tokens), so that the information preservation deficit of strategy 3 is compensated by strategy 1, and the integrativity deficit of strategy 1 is compensated by strategy 3 (cf. Table 1). However, tokens, if defined in this way, are overlapping and thus only partially ordered, so that distance operators are no longer applicable.[9]

[7] Similarly, phonological units that are not expressed in the primary data can be subject to annotations, e.g., short *e* and *o* in various Arabic-based orthographies, e.g., the Ajami orthography of Hausa. A term with zero extension at the position of a short vowel can be annotated as having the phonological value *e* or *o* without having character status.

[8] `https://www.americannationalcorpus.org/graf-wiki/wiki/WikiStart#GraphModel`, 2009/05/08

[9] This can be compensated by marking the base segmentation differently from alternative segmentations. In the abstract GrAF model, however, this can be represented only by means of labels, i.e., annotations. A more consistent con-

Another problem that arises from the introduction of dummy nodes is their theoretical status, as it is not clear how dummy nodes can be distinguished from annotation structured on a conceptual level. In the PAULA formalization, dummy nodes are not necessary, so that this ambiguity is already resolved in the representation.

References

Thorsten Brants. 2000. *TnT A Statistical Part-of-Speech Tagger*. In Proceedings of the Sixth Conference on Applied Natural Language Processing ANLP-2000. Seattle, WA.

Lou Burnard (ed.). 2007. Reference Guide for the British National Corpus (XML Edition). `http://www.natcorp.ox.ac.uk/XMLedition/URG/bnctags.html`.

Jean Carletta, Stefan Evert, Ulrich Heid, Jonathan Kilgour, Judy Robertson, and Holger Voormann. 2003. *The NITE XML Toolkit: Flexible Annotation for Multi-modal Language Data*. Behavior Research Methods, Instruments, and Computers 35(3), 353-363.

Christian Chiarcos, Stefanie Dipper, Michael Götze, Ulf Leser, Anke Lüdeling, Julia Ritz, and Manfred Stede. 2009. *A Flexible Framework for Integrating Annotations from Different Tools and Tagsets* TAL (Traitement automatique des langues) 49(2).

Oli Christ. 1994. *A modular and flexible architecture for an integrated corpus query system*. COMPLEX'94, Budapest, Hungary.

Stefanie Dipper. 2005. *XML-based Stand-off Representation and Exploitation of Multi-Level Linguistic Annotation*. In Rainer Eckstein and Robert Tolksdorf (eds:): Proceedings of Berliner XML Tage, pages 39-50.

Stefanie Dipper and Michael Götze. 2005. *Accessing Heterogeneous Linguistic Data — Generic XML-based Representation and Flexible Visualization*. In Proceedings of the 2nd Language & Technology Conference 2005, Poznan, Poland, pages 23–30.

Stefanie Dipper, Michael Götze. 2006. ANNIS: Complex Multilevel Annotations in a Linguistic Database. *Proceedings of the 5th Workshop on NLP and XML (NLPXML-2006): Multi-Dimensional Markup in Natural Language Processing*. Trento, Italy.

Jin Guo. 1997. *Critical Tokenization and its Properties*, Computational Linguistic, 23(4), pp.569-596.

ception would encode structural information on the structural level, and only linguistic annotation and metadata on the contents level.

John C. Henderson. 2000. *A DTD for Reference Key Annotation of EDT Entities and RDC Relations in the ACE Evaluations* (v. 5.2.0, 2000/01/05), `http://projects.ldc.upenn.edu/ace/annotation/apf.v5.2.0.dtd` (2009/06/04)

Nancy Ide and Keith Suderman. 2007. *GrAF: A Graph-based Format for Linguistic Annotations*. In Proceedings of the Linguistic Annotation Workshop,held in conjunction with ACL 2007, Prague, June 28-29, 1-8.

Esther König and Wolfgang Lezius. 2000. *A description language for syntactically annotated corpora*. In: Proceedings of the COLING Conference, pp. 1056-1060, Saarbrücken, Germany.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. *Building a large annotated corpus of English: the Penn treebank.* Computational Linguistics 19, pp.313-330.

Christoph Müller and Michael Strube. 2006. *Multi-Level Annotation of Linguistic Data with MMAX2*. In: S. Braun et al. (eds.), Corpus Technology and Language Pedagogy. New Resources, New Tools, New Methods. Frankfurt: Peter Lang, 197–214.

Georg Rehm, Oliver Schonefeld, Andreas Witt, Christian Chiarcos, and Timm Lehmberg. 2009. *SPLICR: A Sustainability Platform for Linguistic Corpora and Resources*. In: Text Resources and Lexical Knowledge. Selected Papers the 9th Conference on Natural Language Processing (KONVENS 2008), Berlin, Sept. 30 – Oct. 2, 2008. Mouton de Gruyter.

Helmut Schmid. 2002. *Tokenizing & Tagging*. In Lüdeling, Anke and Kytö, Merja (Hrsg.) Corpus Linguistics. An International Handbook. (HSK Series). Mouton de Gryuter, Berlin

Thomas Schmidt. 2004. Transcribing and Annotating Spoken Language with Exmaralda. *Proceedings of the LREC-workshop on XML Based Richly Annotated Corpora*. Lisbon, Portugal. Paris: ELRA.

Manfred Stede, Heike Bieler, Stefanie Dipper, and Arthit Suriyawongkul. 2006. SUMMaR: Combining Linguistics and Statistics for Text Summarization. *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI-06)*. pp 827-828. Riva del Garda, Italy.

Ralph Weischedel, Sameer Pradhan, Lance Ramshaw and Linnea Micciulla. 2006. OntoNotes Release 1.0. Linguistic Data Consortium, Philadelphia.