NLP-NG - A New NLP System for Biomedical Text Analysis

Robert P. Futrelle, Jeff Satterley, Tim McCormack
Biological Knowledge Laboratory
College of Computer and Information Science
Northeastern University, Boston, MA 02115
{ futrelle, jsatt, timmc} @ccs.neu.edu

Abstract

NLP-NG is a new NLP system consisting of three components: NG-CORE (language processing), NG-DB (database management), and NG-SEE (interactive visualization and entry). The ultimate goal of NLP-NG is to produce information retrieval systems in which users can choose full-text schema, adding specific items to focus their queries. Schema are created by a normalization process which elides adjunctive constructions as well as replacing items by prototypes. Biomedical text contains domain-specific constructions which are revealed by normalization. NLP-NG is based on Construction Grammar. Computationally, all representations are integer-based, allowing efficient storage, indexing, and retrieval. SEE, an Ajax web browser client, allows developers, linguists, and users to view a corpus and modify its properties. NLP-NG uses a 300 million word BioMed Central corpus. NLP-NG does not focus on specific strategies to extract limited classes of information from papers. Instead, it is a universal approach that can codify a wide variety of text in papers..

1. Introduction

The research literature of Biology comprises billions of words in full text papers and abstracts. Many techniques are now available that can extract protein and gene entities, as well as protein interaction and pathway information. The goals of the work are generally to populate databases with facts. Only a modest fraction of the published text is mined by these techniques and only certain types of data are recovered. Scientists working on cutting edge research do not look at the literature as a collection of facts. Rather, they explore outstanding questions and new avenues of inquiry. In the scientific process:

- Experiments are designed.
- Some experiments are of the classic hypothesis-driven type.
- Others are designed to explore the unknown.
- Experiments are set up with organisms, biochemicals, conditions, etc.
- Once set up, the scientist steps back - physical, chemical, and genetic processes are in control.
- Instruments are used to visualize and measure what happens.
- Results are analyzed and discussed, leading to new discoveries.

All elements of the experimental process are of vital interest to scientists. But they cannot all be described as facts. (For another view, see [1]). This is because many results in science are not definite and conclusive. Instead, they are part of ongoing searches for understanding, with much uncertainty along the way. The challenge to natural language processing (NLP) is to extract and codify knowledge about all the steps in the scientific process. This may seem like an impossible quest, given the state of NLP today, but a closer look at the nature of the text in the Biology literature suggests otherwise.

2. The apparent complexity of Biology text

When communicating with their readers, scientists use a limited collection of domain-specific "idiomatic" forms which we call *constructions*. They do this to communicate their ideas and discoveries in clear and unambiguous ways. We came to this conclusion by way of statistical analyses of large collections of Biology papers. The apparent complexity is often due to the inclusion of a few specific terms in otherwise conventional constructions. A search in Google for the following construction, uses two wildcards which we label *1 and *2, and returns 2.4 million hits:

"suggest a role for *1 in *2 transport"

Here are a few of the fillers found.

For *1: PKC-, NAD+, "these motors in anterograde axonal", "caveolae", "clusterin", ...
For *2: DHEAS, "chloride", "the", "retrograde protein", "the regulation of glucose", ...

Unlike the facts so many seek, the word "suggest" tells us that this important construction deliberately avoids making a firm statement. Millions of suggestions such as these lead to new follow-up investigations. The example

978-1-4244-5121-0/09/$25.00 ©2009 IEEE

makes three important points. 1) Important (frequent) constructions are not statements of fact, 2) The high frequency of important constructions gives us the opening we need to apply statistical analyses to discover and codify the constructions scientists use, and 3) The role of wildcards suggests a strategy: *Normalization*.

3. Normalization reduces complexity

To apply NLP techniques, we can't merely guess what constructions might be important and then confirm them statistically. We need a discovery mechanism. A simple example illustrates the discovery strategy used in our NLP-NG system ("NLP New Generation"). Assume we create a large corpus and *normalize* it by substituting all acronyms such as DNA with the *prototype* "ABC". Statistical analysis of the text for n-grams of varying lengths would discover more than 1 million patterns of the form "lysates from ABC cells". Without this substitution, the n-gram frequencies would be much smaller, so this common construction might not rise above the noise. For example, "Lysates from 293T cells" appears at 1/30 the frequency of the ABC-substituted form, and "Lysates from McA cells" returned a total of 2 hits. Fishing something of frequency 2 from a sea of 1 million items demonstrates the power of normalization. Other normalization strategies could consist of replacing all numbers by "10" and all chemical substances by "carbon". An additional and important normalization strategy is the deletion (elision) of terms of little importance. Deletion of the adverb "extensively" from an entire corpus can raise the frequency of typical constructions by a factor of 50. "Extensively" is a *hedge*, a word that shades meanings, without altering them in a substantial way.

3.1. Normalization in information retrieval

In the future, we will develop retrieval systems built on NLP-NG. Our approach to IR will be situated between wildcards and term-expansion. Wildcards are indiscriminate. Term-expansion techniques cannot deal, for example, with the tens of thousands of distinct numbers that appear in papers. Normalization replaces all numbers by "10". Once found, the system could drill down to more specific constructions. Normalization is a preprocessing step which avoids complex pattern searches during retrieval.

4. Construction Grammar

Construction grammar (CxG) was developed to go beyond the limits of generative grammar [2, 3]. "Construction" refers to a wide range of linguistic phenomena, from word morphology, to sentences, to entire documents. CxG assumes a direct mapping between form and content. CxG points out that many constructions resist analysis by the classical techniques of generative grammar or compositional semantics. In spite of this, such constructions have meanings that are readily apparent. Our approach goes beyond this in assuming that because of the form/content mapping, a single form in some domain may appear at high frequency to unambiguously deliver a fixed meaning that all speakers who know the domain understand. For example, "resuspended in * buffer", returns 14M hits in Google. There is no reasonable compositional semantics analysis of this construction. But its meaning is clear to any cell biologist - it is an intermediate or final step in washing cells by one or more centrifugation-resuspension steps.

In NLP-NG, the forms produced by normalization should not be thought of as literal forms, but as *construction schema* that often can represent millions of specific instances. The normalization by "ABC" substitution we mentioned creates *acronym schema*. A substitution of numbers by "10", creates *number schema*.

5. NLP-NG Implementation

NLP-NG is a new computational infrastructure we are developing that implements the underlying NG-CORE, NG-DB, and NG-SEE systems. The system is implemented in Java. NG-DB uses embedded Apache Derby as its underlying database engine. NG-SEE uses the Google Web Toolkit (GWT) to build Ajax, web-based tools.

5.1. NG-CORE - Data structures

NG-CORE is the major system that reads the corpus, builds and works with all data structures, drives NG-DB, and functions as the server for NG-SEE.

Tokenization. Tokenization normally separates tokens by intervening whitespace and tries to identify the nature of the tokens, e.g., *real number*, *capitalized*, etc. We have developed *Extreme Tokenization*, which creates only six types of tokens called *atoms*: alphabetic characters only, digits only, whitespace, XML or HTML tags, punctuation, and Unicode characters, e.g., '&916;' representing 'Δ'. Atoms can later be re-combined to create *compounds.* Many tokenizers make premature decisions without using adequate context or domain-specific information. Extreme tokenization shifts these decisions to later, specialized procedures. Atoms are stored in a *Constructicon*, *CC*, whose entries contain only an integer UID and the atom's string. The Constructicon is the only place in NLP-NG that contains strings. All further references use UIDs. The *Token Sequence* is then just a sequence of atom UIDs with each token indexed by

its sequence SID. For a given corpus, the Constructicon and Token Seqeucne are append-only structures.

Compounds are stored in the Compound Constructicon, *CCC*. The CC and CCC are illustrated in Fig. 2. A typical compound of biological interest is *E. coli*, made up of six atoms: "<it>", "E", ".", whitespace, "coli", "</it>".

Relations and properties. Fig. 2 illustrates a number of *relations*, e.g., the relation between the three tokens of the compound "sodium chloride" in CCC-5, and the atoms that make it up, in the the CC. *Properties* are attribute-value pairs. Each attribute is represented by a database table. The tables in figures 2 and 3 have version numbers attached, e.g., "Enzyme-4". An obvious need for such versions would be for part of speech tagging for various tagsets such as Hepple (Penn), Biber, and BNC Basic (C5). As in Fig. 3, a *Part-of-Speech* table would be a table of pairs, (not shown), the Token Sequence SID, and attributes such as NN, JJ, and IN (Penn tagset). Tags are designated by an integer ID which is a component of an NG-CORE *Symbol*, a Java class. Another attribute is *Suffix*. In Suffix-6 in Fig. 2, 347 refers to the token "GTPase", and 350 is the CC entry for "ase". Suffix tokens such as "ase" are added to CC, in addition to the tokens found by tokenization of the corpus. An entry in the Constructicon such as 347 ("GTPase") could have attributes "CAPITALIZED" (a binary attribute) and "LENGTH" (an integer value). NLP-NG uses *stand-off annotation* throughout [4].

Many properties for the Constructicon, such as "CAPITALIZED", would appear to occupy large tables, in this case, dominated by "FALSE" values. Only the "TRUE" entries are stored in NG-DB database system. When brought into memory, they are effectively *joined* with the Constructicon, matching on SID values, and added to a *Cache sequence*, but only for the SID positions which have "TRUE" values. The Constructicon cache sequence is not shown in the Fig. 2, but is similar to Cached spans-20 in Fig. 3. An example of a join instance is the join of UID2 347 in Suffix-6 in Fig. 2 with the UID 347 in CC. In some cases, a hashmap is more efficient than an array-based like the one shown in Fig. 3.

Token Sequence Spans. All annotations of the Token Sequence are implemented by *Spans*. A Span is designated by its start and end SIDs as in the Elision-9 table in Fig. 3. A Span entry may have additional information, as in Prototype-7 in Fig. 3.

Elisions - Virtual deletions. An important normalization procedure is *elision* that can hide non-required elements such as parenthesized items. When a subsequence of the Token Sequence is to be elided, a Span entry is made in an Elision table such as the one covering SIDs 890-892 in Elision-9 in Fig. 3. The Token Sequence itself is untouched. Elision sets can be large, e.g., the elision of all figure captions

Prototypes - Virtual substitutions. The other major Normalization procedure is the result of substituting one term by another (Prototype-7 in Fig. 3). For example, numbers such as "0.005" could be replaced by the prototype, "10", and all chemical substances could be replaced by "carbon". There might be situations in which an insertion needs to be made. This could be done by replacing a whitespace (a span of length 1) with: whitespace <inserted term>, whitespace.

Views. Views in NLP-NG are analogous to views in database systems. Views are the result of the virtual operations of elisions and substitutions. Views can be combined by set operations. Thus, the figure caption spans of an article can be subtracted from the body span to yield only the non-caption text. A view of a Token Sequence is fully represented by its Cached span, as in Fig. 3. This means that it can be stored as a single entity set in a database.

5.2. NG-CORE - Pattern Matcher

Many of the data structures beyond the Constructicon and the Token Sequence are created by the NG-CORE *Pattern Matcher* which implements analogs of most of the capabilities of the well-known character-based matchers such as grep. Pattern elements can be combined using alternations and sequences, which can be repeated using quantifiers, either with specified minimum and maximum repetitions, or with the Kleene operators (*, +, and ?). Quantifiers can also be specified as greedy, lazy, or possessive. However, unlike character-based matchers, NG's pattern elements can specify token strings, properties and/or span types that should or should not be matched, not individual characters. Character-based matching on atoms and compounds uses existing pattern matchers, and indicates matches with properties. For example, all tokens that match the pattern "*ing" will have a property whose attribute is "SUFFIX" with a CC UID value referencing "ing". The property can then be found by NG's matcher. The pattern matcher is well-integrated with Views, allowing searches to automatically skip over elided parts of the corpus, continuing the match after the elided span, and using prototypes instead of the terms they replace.

The Pattern Matcher is important in the larger tasks we are pursuing, most importantly the discovery of a large numbers of significant constructions. This is based on gathering statistics on n-grams using views with a large number of elisions and prototypes.

5.3. NG-DB - The database system

The data structures shown in Figs. 2 and 3 are stored in the NG-DB database system, implemented using embedded Apache Derby. By working with large amounts of main memory and portions of our 300M word corpus, we can hold all the needed data in main memory. The system does not have to use complex SQL commands or rely on the database for joins and related processing. Instead, tables are brought into memory in their entirety. It is not necessary to write out entire tables when small changes are made. For example, to remove a sentence boundary, the tuples corresponding to two sentences could be deleted and a single new tuple could be inserted. Relational database systems such as Derby are designed to perform such operations efficiently. Given that our corpus could have on the order of ten million sentences, efficiency is important.

Apache Derby is free to download and use. The basic system is contained in a Java jar file of less than 3MB. Each database created is contained in a folder chosen by the user. This allows a database to be easily copied or moved around on a machine, the internet, or put on a DVD. Derby can be used on the command line, embedded using JDBC, or in server mode, behind an HTTP server or a servlet container such as Apache Tomcat. The limits for the number of attributes and tables, as well capacity limits, go far beyond anything needed by NLP-NG.

5.4. NG-SEE - An Ajax web client for visualization and interaction

NLP-NG supports both automated and user-based analysis. The NG-SEE module, or "SEE", supports visualization and interaction. SEE is an Ajax, browser-based application. NG-CORE acts as the web server for the SEE client. SEE was built using the Google Web Toolkit (GWT) [5]. Implementation in GWT uses Java which GWT converts to Javascript, which then executes in browser clients.

When NG-CORE receives a request from SEE, it packages data into a container and sends it to SEE. The data includes both annotated text and, in many cases, its context. SEE renders the data and gives the user options for interacting with it. If the user creates or modifies annotations and transformations on the data, SEE sends the new information back to NG-CORE. NG-CORE initially sends the user a choice of *projects*, each with a corpus and related data.

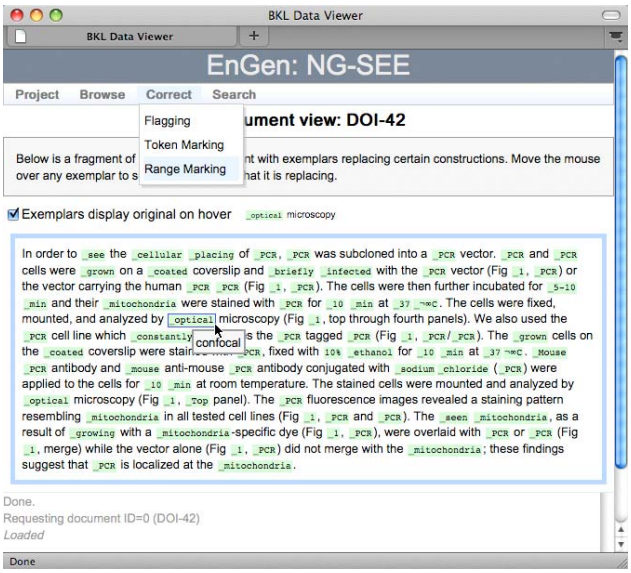

Figure 1. The NG-SEE system displaying a document containing a number of prototypes, such as "_cellular", "_PCR", etc., which replace the original document words. The user can hover over an prototype to see the text it is replacing. In the figure, the user is hovering over the prototype "_optical" to reveal the word it replaced, "confocal". The checkbox on the upper left has activated the highlighting and hover functionality. Some of the other available tools are shown in the drop-down menu. *Flagging* allows a user to declare an auto-generated property as invalid. *Token marking* allows a user, for example, to indicate that a token should be classed as a hedge. *Range marking* allows a user to add or edit a span. For example for the text "sodium chloride", only "sodium" might be highlighted as a chemical substance. The user could change that to the pair "sodium chloride". ("EnGen" in the title bar refers to a previous version of NLP=NG.)

Document viewer. One tool currently supported in NLP-NG and SEE is the *document viewer*, Fig. 1. NG-CORE sends SEE an HTML string containing span tags, which surround tokens to indicate their properties. SEE highlights these properties in the browser using JavaScript and CSS. The user can turn highlighting on or off for each type of property, as in Fig. 1. The SEE system can switch between showing the original and augmented text, especially useful for viewing prototypes and elided constructions.

Manual Entry & Correction. It is often difficult to design an algorithm to correctly find all instances of a construction. So it is important for a researcher to be able to correct erroneously identified constructions. NG-SEE has tools that allow a user to check constructions for correctness. SEE highlights all instances of a particular type of construction so that the researcher can flag or alter incorrect ones and inform NG-CORE. Supervised

learning requires training sets. For these tasks, a user, for example, could select a subsequence of the displayed document and choose a type of construction. The user would mark the appropriate construction, which would then be returned to NG-CORE. For example, prepositional phrases are sometimes treated as non-required elements. But if the preposition is part of a prepositional verb ("phrasal verb") it would be a mistake to class the following NP as the argument of the preposition rather than its correct classification as the argument of the verb. SEE allows a researcher to manually identify constructions such as prepositional verbs.

6. Discussion

Comparison to other NLP systems. Prominent NLP systems such as GATE [6] and NLTK [7] are used extensively and have a wide range of capabilities. The NLP-NG system has a number of important design and implementation features that sets it apart from the two and from virtually all other NLP systems. No other NLP systems appear to have been built with construction grammar at their core as NLP-NG has. The strategy of *normalization* appears to be a unique aspect of NLP-NG. Though a few systems, and GATE in particular, use relational databases, our use of large numbers of "narrow" attribute tables, only a few columns each, creates a highly modular system. In addition, each such table functions as an inverted index. We are adding a full inverted index for the token sequence. When the pattern matcher is enhanced to take advantage of the inverted indexes it will be very efficient. The use of Derby makes it simple to distribute our databases.

Resources. Our major resource is the collection of 50,000+ papers in XML format from the BioMed Central open access publisher, BMC. There are important corpora that have been given detailed and useful markup, primarily part-of-speech and chunking. Two that are readily available are the GENIA corpus of 2,000+ Medline abstracts [8], and the American National Corpus (ANC) [9] which currently contains 22M words, with a goal of 100M words. Both come with their own set of tools.

Other domains. Beyond biology, the nature of any focused domain will be reflected in the constructions it uses. As one example, consider the schema, "To predict the label of * using *". This returns 160,000 hits, with instances such as, "To predict the class label of a tuple using naïve Bayesian classification". Another: "assess the seismic * bridges", returns 150,000 hits, with instances such as, "assess the seismic fragility of bridges".

Future plans for NLP-NG. This paper is our first about NLP-NG. It describes our initial implementation of the computational, database, and web-based portions of the system. The next steps are clear and under way: Normalization has to be implemented on a large scale. Extensive statistics will be generated to identify important constructions. It will take some time to assess and understand the nature of the constructions we discover. Widespread distribution of the software and databases is another obvious goal. For the BMC corpus: "Anyone is free: to copy, distribute, and display the work; to make derivative works; to make commercial use of the work." The NG-SEE system, by its very nature, allows collaboration among multiple users anywhere on the internet working on the same corpus at the same time. Such collaborative projects can be further coordinated and integrated using chat, video chat, and shared web documents such as Google Docs. Our long term goal is to build information retrieval systems that take advantage of NLP-NG's unique and powerful capabilities.

7. References

[1] A. d. Waard, "A Pragmatic Structure for Research Articles," in *2nd Intl. Conf. on the Pragmatic Web* Tilburg, The Netherlands: ACM, 2007, pp. 83-89.

[2] A. Goldberg, *Constructions at work: The nature of generalization in language*: Oxford University Press, USA, 2006.

[3] D. Schönefeld, "Constructions," *Constructions, SV,* vol. 1, 2006.

[4] H. Thompson and D. McKelvie, "Hyperlink semantics for standoff markup of read-only documents," in *SGML Europe 97* Barcelona, 1997.

[5] R. Dewsbury, *Google web toolkit applications*: Prentice-Hall, 2008.

[6] K. Bontcheva, V. Tablan, D. Maynard, and H. Cunningham, "Evolving GATE to meet new challenges in language engineering," *Natural Language Engineering,* vol. 10, pp. 349-373, 2004.

[7] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*: O'Reilly Media, 2009.

[8] J.-D. Kim, T. Ohta, Y. Teteisi, and J. i. Tsujii, "GENIA corpus - a semantically annotated corpus for bio-textmining," *Bioinformatics,* vol. 19(suppl. 1), pp. i180-i182, 2003.

[9] N. Ide and K. Suderman, "The American National Corpus First Release," in *Fourth Language Resources and Evaluation Conference (LREC)* Lisbon, 2004, pp. 1681-1684

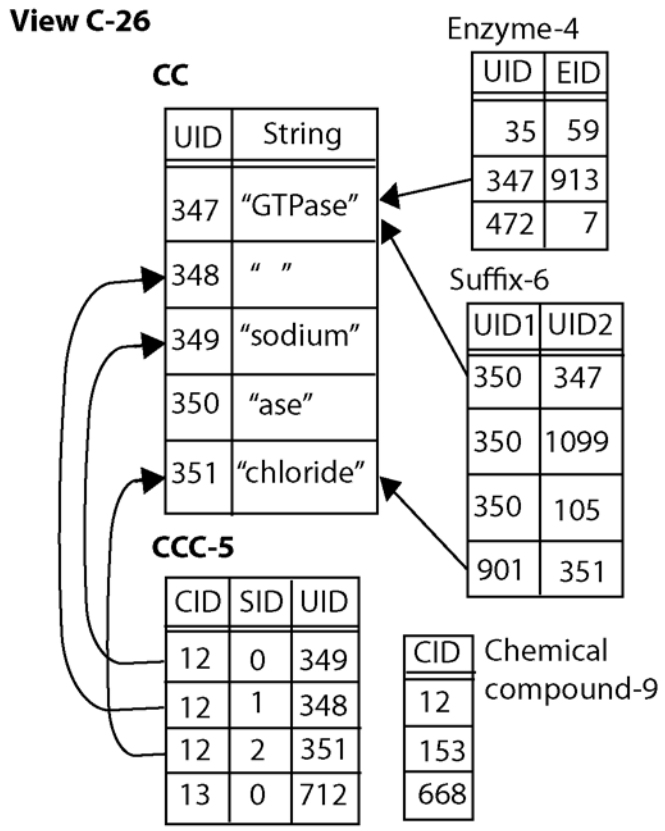

Figure 2. This shows a portion of the data structures related to NLP-NG tokens and compounds. The structures are depicted as relational database tables, which reside in the database and are mirrored by in-memory structures, e.g., Java collections. Tokens, atomic elements created by Extreme Tokenization, are stored in the Constructicon, CC. This is the only place in all of NLP-NG that contains strings. All further references are handled by the integer UIDs. The most important related structure is the Compound Constructicon, CCC. In this example, the three token compound, "sodium", whitespace, "chloride", has CID = 12 and consists of the tokens, 349, 348, and 351, in that order. Stand-off annotations are contained in separate tables, which may exist as various alternates. Thus, Chemical compound-9 is alternate #9 of the collection of Chemical compound tables. In this case, its CID = 12 appears in both CCC-5 and Chemical compound-9 and designates the chemical compound sodium chloride. (Note the two distinct uses of "compound" here - clear from their contexts.) The Enzyme-4 and Suffix-6 tables can be understood by examining them carefully. The data structures in this example comprise View C-26.

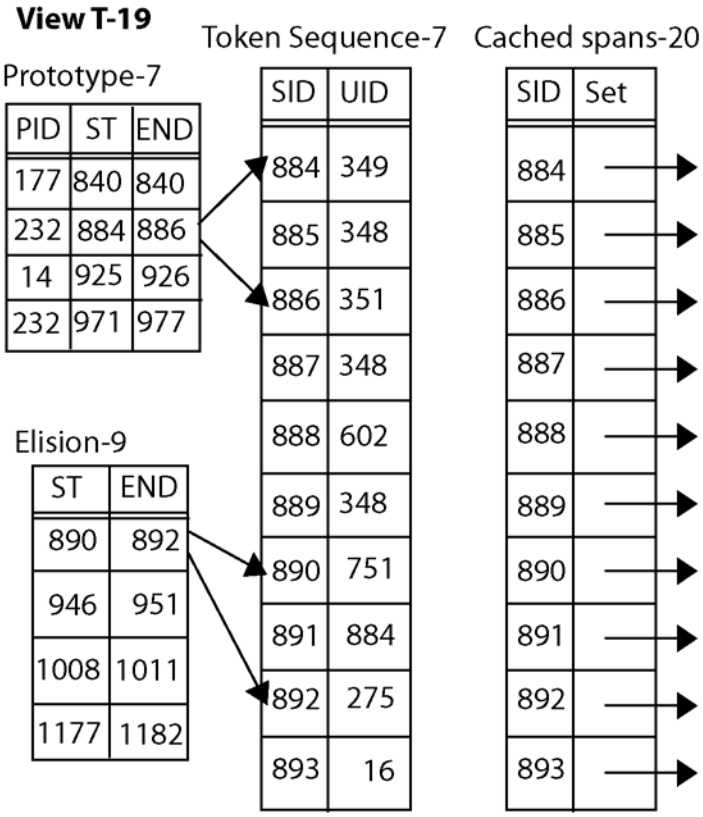

Figure 3. Token Sequence-7 contains the text of one or more articles in a corpus, #7. In this Token View T-19, standoff data is contained in Prototype-7 and Elision-9. Spans, designated by their start and end SID values, designate portions of a Token Sequence. Spans can range in length from entire article sets, to paragraphs, to single tokens. The Prototype, PID = 232, designates that the three token sequence, 884, 885, and 886, (sodium chloride) is to be read as prototype 232, referring to an entry in the **CC** or **CCC**. The prototype in this case, and for all chemicals, could be chosen as carbon (the backbone for the chemical compounds which are the basis of life on Earth). Elisions are equally important in NLP-NG. E.g., they would typically elide phrases such as, "In this study", or "approximately" in "by approximately five-fold". In the example, a three token sequence, with no intervening space is marked for elision - it is hidden from the pattern matcher. The Cached spans-20 sequence collects together all the attributes for each SID position. It contains all the data for the Token View, T-19. Similar caches exist for the **CC** and **CCC** (not shown).