XML-Based Representation Formats of Local Grammars for the NLP

Javier M. Sastre Martínez

Institut Gaspard Monge, Université de Marne-la-Vallée
77454 Marne-la-Vallée Cedex 2
sastre@univ-mlv.fr

Abstract

The construction of local grammars for the exact recognition of each valid structure of a natural language is a very long term task which consumes huge amounts of human resources and produces huge amounts of data to be accumulated over the years. The use of convenient computer assisted grammar construction (CAGC) tools alleviates this task, but the representation format of the generated data must not be tool-dependent because of two reasons: the data must persist across the time in spite of the short life span of such tools due to the fast computer technology evolution; cooperative work groups must be able to easily exchange their data even if they use different CAGC tools. We propose here two equivalent representation formats based in well-covered solid standards, like XML and XML-Schema, and conceived to serve as exchange formats between computer applications dealing with local grammars.

Introduction

Local grammars are conceived to exactly recognize specific natural language structures and to recursively serve as pieces of further more complex grammars recognizing higher level structures until the construction of an exact grammar covering the whole extension of a natural language is reached. Many new problems have arisen since this approach to the exact recognition was introduced (Gross, 1997). First of all, it was necessary to provide linguists with computer tools which would facilitate the construction of local grammars and would be able to apply them over real corpus in order to validate them. The Intex system (Silberztein, 1993) was an already existent tool allowing the construction of local grammars. Subsequently, the Unitex system (Paumier, 2002) was developed for this purpose. Both of them use graphical representations of grammars called graphs. The figure 1 shows an example of graph representing a local grammar semantically equivalent to the structure "in order to + verb". The code "<V:W>" represents any infinitive verb defined in an electronic dictionary and the code <a> corresponds to "a" or "an". Grey boxes represent references to other graphs. This way it is possible to reuse grammar definitions in order to easily built larger ones. In this case, the referenced grammars define verbal constructions. As we can see, this graphical representation is simple and easy to read: we can rapidly realize of the recognized sequences just by reading from left to right (e.g.: "in attempts to <V:W>", "in an attempt to <V:W>", etc).

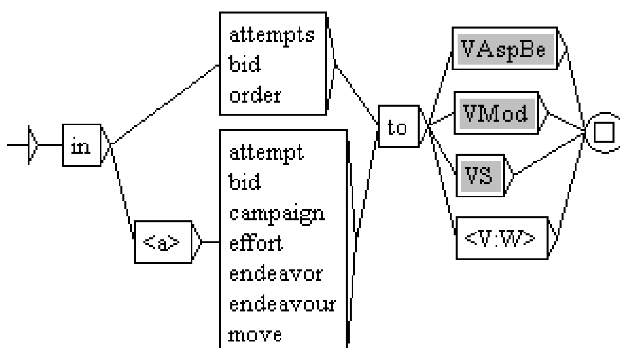

Figure 1: "In order to + verb" graph

As the number of constructed grammars increases, it becomes harder to efficiently store and selectively recover them so that we can reuse them for the construction of new ones. Consequently, a local grammar library allowing to search and extract them by means of grammar specific queries has already been conceived (Constant, 2003). However, it does not exist up to now a common graph representation format; it has been necessary the use of format conversion tools in order to edit graphs with other tools than the one used to create them. If a system for the massive storage and exchange of graphs amongst computer applications is to be built, it is firstly necessary to develop a graph exchange representation format easy to be supported, able to represent any natural language grammars and as most stable as possible. Especially, stability is an important factor since the migration from a representation format to a new one is a hard task that, if not correctly done, can lead to data corruption.

The World Wide Web Consortium (W3C) is an international recognized organization whose purpose is the development of common protocols allowing interoperability over the World Wide Web (WWW). The Extensible Markup Language, or XML (Skonnard, 2001), is a simple, very flexible text format derived from SGML and which is being widely used as exchange format of data on the Web and elsewhere. XML Schemas (Vlist, 2002) provide means for defining the structure, content and semantics of XML documents.

The Unicode Standard (Allen, 2003) is a character coding system designed to support the worldwide interchange, processing and display of the written texts of the diverse languages and technical disciplines of the modern world. In addition, it supports classical and historical texts of many written languages. XML relies on Unicode and closely tracks its revisions.

There exist several tools and libraries which facilitate the development of software dealing with XML and XML Schema based representation formats. For example, the Java Architecture for XML Binding (JAXB) is a Java technology in the Java Web Services Developer Pack, or JWSDP (Armstrong, 2003), that enables to automatically generate Java code for the analysis, validation and

synthesis of XML documents following a given XML Schema specification.

The common graph exchange formats presented in this paper are based on XML and formally defined by means of XML Schemas, thus fulfilling all the presented requirements: stability, language independency and easiness of being supported.

Graph object description

Graphs are graphical objects equivalent to transducers within a recursive transition network (RTN) conceived to facilitate the construction and comprehension of grammars for the NLP. As we can see in the example graph (figure 2), a graph is composed by a set of linked boxes of different types.

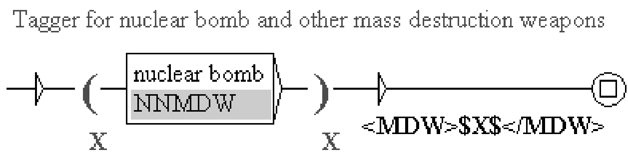

Figure 2: example graph

Transduction boxes

Transduction boxes recognize a set of input token sequences and optionally generate an output character sequence. If no output is defined, the empty sequence is assumed by default. Graphs can also be referenced within a transduction box allowing the box to recognize every sequence recognized by the referenced graph; from a procedural point of view, we say that the referenced graph is "called" by the graph containing the referencing box. Every directly or indirectly called graph is a subgraph of the calling graph, and recursive calls are also allowed. Each white background line of a transduction box corresponds to an input sequence (e.g.: "nuclear bomb") and each gray background line to a subgraph call (e.g.: "NNMDW" subgraph). An additional line under the box corresponds to the box output sequence (e.g.: "<MDW>X</MDW>"), if defined. Once a transduction box recognizes an input sequence fragment, its associated output sequence is appended to the graph output. Transduction boxes that only recognize the empty word are represented by an empty triangle with the associated output below.

Initial and final boxes

Initial and final boxes are to be contained in every well-formed graph, a single one of each type per graph. Only transduction or record boxes are to be the initial box of a graph, which is distinguished from the others by an incoming-from-no-box transition. The application process of a graph starts at its initial box and follows the outgoing links or transitions as the following boxes are able to recognize the next input fragment. By default, transitions start at the box right sides and end at the left box sides. The transition direction can be switch for the representation of right-to-left language grammars. A whole input sequence is recognized by a graph if its last fragment is recognized by a box having an outgoing link towards the final box, which is represented by a circle containing a square. Opposite to the initial box, the final box has no other purpose than marking the recognition of a sequence accepted by the graph.

Record start/stop boxes

Record start/stop boxes represent points within a graph where the input sequence recording mechanism is switched: once a record start box is reached, the associated variable (e.g.: "X") is defined and initialized to an empty sequence value and every later recognized input sequence fragment is appended to the variable content until a record stop box associated to the same variable is attained. Once a record stop box is reached, the associated variable remains defined but its content is no longer modified until a start box associated to the same variable is reached. Recorded input fragments may be parts of transduction box character sequence outputs by specifying the correspondent initialized variable identifier within the output sequence specification (e.g.: "X"; we surround variable identifiers with dollar symbols in order to distinct them from merely output character sequences). Once the final box is reached, defined variables within the graph become undefined: variable values are not conserved once a graph application has ended. Called subgraphs may access variables initialized by their calling graphs independently of the call depth. In case of variable identifier conflict between successive graph calls, the inner call graph variable value is assumed (outer variable definitions are hidden by same identifier inner variable definitions). Apart from the recording mechanism, record boxes behave as transduction boxes recognizing the empty word and generating an empty output sequence.

Comment boxes

Comment boxes (e.g.: "Tagger for nuclear bomb and other mass destruction weapons") contain meta-information conceived to facilitate the interpretation of the graph. The presence or absence of such boxes does not alter the represented grammar.

From graphs to FSTs

Let G be a graph, we call compilation of G the transformation process of G and its subgraphs into minimal finite-state transducers (FSTs) and further optimization of the resulting RTN. We call G the main graph of the compilation. Every information w.r.t. the representation format of the graphs is lost. The structure of the resulting RTN may be modified in order to boost its application. The compilation process, as usual, is not reversible.

Opposite to standard FSTs, whose operations associated to their transitions involve uniquely input recognition/output generation, graph equivalent FSTs may have three extra operation types: sub-FST calling, record start and record stop operations. Every transition associated to a sub-FST calling operation is equivalent to an epsilon transition towards the initial state of the called sub-FST and further coming back from any of its final states towards the arriving state of the calling transition. Transitions associated to record start/stop operations are equivalent to epsilon transitions that switch the input sequence recording mechanism analogous to the graph's one. We call main FST of a RTN the first one to be considered during the RTN application process. We say

that an FST is a sub-FST of another one iif the former is called within any of the transition operations of the latter.

XML Terms

Terms are the atomic information units to be recognized and which compose input sequences. We represent them as the following XML elements:

1. <epsilon/>: empty word.
2. <blank/>: mandatory blank space occurrence before the next term.
3. <noblank/>: no blank space occurrence before the next term.
4. <symbol>: specified symbol (not a letter).
5. <word/>: any word (sequence of letters).
6. <word case=”lowercase”/>: any lowercase word.
7. <word case=”uppercase”/>: any uppercase word.
8. <word case=”firstupper”/>: any uppercase starting word.
9. <word> specified word (sequence of letters) or any of its variants (e.g., case variants).
10. <word case-sensitive=”false”> specified word.
11. <token/>: any token (symbol or word).
12. <token case=”lowercase”>: any symbol or lowercase word.
13. <token case=”uppercase”>: any symbol or uppercase word.
14. <token case=”firstupper”>: any symbol or uppercase starting word.
15. <token> specified token or any of its variants.
16. <token case-sensitive=”false”> specified token.
17. <dicentry>: specified dictionary entry.1
18. <dicword/>: any dictionary defined word.
19. <dicword>: any dictionary defined word having the set of properties specified using the dictionary’s notation.
20. <punctuation>: any punctuation symbol.
21. <number>: any sequence of numbers.
22. <sentencetag>: end of sentence.

Additionally, any term representing a set of tokens have the optional attributes “complement” and “filter”. A term having the former attribute set to “true” represents the complement of the set represented by the original term. If the attribute is not set or is set to “false”, default behavior is assumed. If the considered term is <dicword>, the domain of the complementation is restricted to the set of dictionary-defined words. The “filter” attribute value is a POSIX regular expression. When present, the token to be recognized by the term must match the regular expression. This attribute represents a further restriction by means of a regular expression filter. If both complementation and filtering operations are to be applied, the former is considered before the latter.

XML output char subsequences

Output character subsequences are represented by <chars> elements or by <var varid=”…”/> elements. The former contains the character subsequence to be generated (direct addressing) and the latter specifies by means of its “varid” attribute value the identifier of the variable containing the subsequence (indirect addressing). An output character sequence is represented by a sequence of one or more <chars> and/or <var/> subsequence elements. Alternate direct/indirect addressing within an output sequence specification is allowed.

XML graphs

A graph is represented by an element <graph pathname=”…”>, whose attribute “pathname” value is the address of the represented graph. This element has an optional element <format> and an element <boxes>. The former contains information about the graphical representation of the graph and the second the list of boxes that compose the graph. This list must contain an initial box, zero, one or more non initial or final boxes and a single final box. The element <boxes> must have an attribute “initialbox” whose value is the identifier of the initial box.

Box elements

Every box element representing a non-comment box, and therefore a reachable box, has an attribute “id” whose value identifies uniquely the box. Every box element has a pair of optional attributes “posx” and “posy”, which represent the coordinates in a Cartesian system where the box is to be located in the graphical representation of the graph. Record start and record stop boxes are represented by <recordstart> and <recorstop> elements respectively. Both of them have an attribute “varid” whose value is the identifier of the variable associated. Comment boxes are represented by <comment> elements, whose only content is the comment text. The final box is represented by the element <final/>, which has no content. Transduction boxes are represented by <transduction> elements. The elements representing record boxes or transduction boxes may contain an element <transitions>. Such element contains the list of box identifiers, separated by a single space, which are pointed by the box outgoing transitions.

Inputs and output elements

Additionally, each <transduction> element contains an <inputs> element, which defines every input recognized by the box, and optionally an <output> element, which defines the output character sequence to be generated by the box. If no output is to be generated, the <output> element is omitted. The element <inputs> contains one or more elements <sequence> or <subgraph/>. The former represents a sequence of terms and the latter a subgraph call. The element <sequence> must contain a sequence of at least one term element. The <subgraph/> elements have an attribute “pathname” whose value is the address of the subgraph to be called. The element <output> contains a sequence of at least one output character subsequence element.

Input subsequence elements

The input subsequence elements <symbols>, <words>, and <tokens> represent sequences of one or more consecutive symbols, words or tokens respectively. Consecutive sequences of <symbol>, <word> or <token> elements can be represented by an input subsequence

1 A dictionary entry corresponds to a non-ambiguous lemma description. Dictionary entry references are to be used in lexically disambiguated corpus by means of added tags.

element containing the list of the correspondent tokens separated by a single space. Input subsequence elements have two optional attributes: “case-sensitive” and “blanks”. The former indicates if word variants are accepted (“true” value) or not (“false”, default value). The latter indicates if blank spaces between tokens are optional (“optional”, default value), mandatory (“mandatory” value) or forbidden (“forbidden” value).

XML RTNs

An RTN is represented by an <rtn pathname=”…”> element, whose attribute “pathname” value is the address of the represented RTN. This element contains an element <fsts> and an element <operations>. The former represents the list of FSTs that compose the RTN and the latter the set of operations associated to the transitions of all the FSTs within the RTN.

FST elements

The element <fsts> must contain an element <main>, which represents the main FST, and a list of zero, one or more elements <sub>, which represent the sub-FSTs of the main FST. The structure of both <main> and <sub> elements is the same: they have an attribute “id”, whose value identifies uniquely the FST within the RTN, and contain an element <initialstate> and zero, one or more elements <state>. The element <initialstate> corresponds to the initial state of the FST and the <state> elements to the rest of its states. Those elements share the same structure: they have an attribute “id”, whose value identifies uniquely the state within the FST, and an attribute “final”, whose value indicates if the state is final (true) or not (false, default value). They contain zero, one or more <transition> elements, each one representing a state outgoing transition. Those elements have an attribute “operationid”, whose value is the identifier of the operation associated to the transition, and an attribute “stateid”, whose value corresponds to the identifier of the arriving state.

Operation set element

The <operations> element contains zero, one or more operation type elements, each one representing a single operation to be associated to any of the FST transitions. Every operation type element has an attribute “id” whose value identifies uniquely the operation within the whole RTN.

Transduction operation element

An element <transduction> within the <operations> element represents an input term recognition/output character sequence generation operation. The input term is represented by a mandatory term child element. In case an output is to be generated, it is represented by an <output> element containing a sequence of at least one output character subsequence element.

Sub-FST call operations

FSTs within a RTN are also considered operations when associated to a transition: the transduction operation defined by the FST. We represent a sub-FST call operation by setting the “operationid” attribute value of the correspondent <transition> element to the identifier of the FST to be called. FST and operation identifiers must be different since they share the same space.

Record operation elements

Record start and record stop operations are respectively represented by <recordstart> and <recordstop> elements within the <operations> element content. In addition to their “id” attribute, they have a “varid” attribute whose value is the identifier of the recording variable.

Conclusion

In this paper we described XML-based representation formats of graphs and RTNs. Those formats have been formally described by means of XML Schemas. Both XML and XML Schemas are solid standards, thus the presented formats are expected to last and consequently to save the effort of developing new formats and to avoid data corruption due to format migration. Thanks to the compatibility of XML with Unicode, any natural language is to be supported. Thanks to tools like JAXB, it will be relatively easy from now on to add support for the presented formats to existing or new computer applications and, therefore, to enable them to exchange local grammars. Graph and RTN format conversion tools have already been developed and tested for the migration from Unitex native formats to the presented XML-based formats. The 88% of the resultant Java source code was automatically generated by JAXB. Now that we have set an appropriated media for local grammar representation we can proceed with the implementation of a local grammar management system for their efficient storage and diffusion among the scientific community.

References

Armstrong E., Ball J., S. Bodoff and Jendrock E. (2003). The Java™ Web Services Tutorial. http://java.sun.com/webservices/tutorial.html. Viewed 21st April 2004. Sun Microsystems.

Constant M. (2003). Grammaires locales pour l’analyse automatique de textes : Méthodes de construction et outils de gestion. http://www-igm.univ-mlv.fr/LabInfo/theses/2003/constant.pdf. Viewed 20th September 2003. Université de Marne-la-Vallée.

Gross M. (1997). The Construction of Local Grammars. In Roche E. and Schabes Y. (eds.), Finite State Language Processing, Cambridge, Mass., The MIT Press, pp. 329-352.

Paumier S. (2002). Unitex manuel d’utilisation. http://www-igm.univ-mlv.fr/~unitex/manuelunitex.pdf. Viewed 28th February 2004. Université de Marne-la-Vallée.

Silberztein M. (1993). Dictionnaires électroniques et analyse automatique de texts: le système INTEX. Mason Ed. : Paris. ISBN 2225841578.

Skonnard A. and Gudgin M. (2001). Essential XML Quick Reference: A Programmer’s Reference to XML, XPath, XSLT, XML Schema, SOAP and More. Addison-Wesley Pub Co; 1st edition. ISBN 021740958.

Vlist E. (2002). XML Schéma. O’Reilly, Paris. ISBN 2841772152.

Allen J. and Becker J. (Eds.) (2003). The Unicode Standard Version 4.0. The Unicode Consortium