

XML Framework for Decision Diagrams

Stanislav Stankovic, Jaakko Astola
Tampere International Center for Signal Processing
Tampere University of Technology
FIN-33101 Tampere, Finland

Abstract

In this paper, we propose an XML (Extensible Markup Language) based standard for description of the structure of various types of decision diagrams for representation of discrete functions. Proposed standard describes elements of the structure common to various types of decision diagrams. It also provides facilities for storing of additional information specific to particular types of decision diagrams. Properties of XML enable us to define a standard that is flexible enough to be applicable to existing types of decision diagrams as well as new types that could be defined in future. The existence of such a standard permits efficient storage and exchange of data in decision diagram form between various software systems.

1. Introduction

The purpose of this paper is to describe an XML (Extensible Markup Language) based framework for manipulation and storing of decision diagrams.

Decision diagram is a data structure that permits efficient, in terms of space and time, representation of discrete functions [1], [7], [10].

There are standard methods for programming of decision diagrams. For example, CUDD library is used in many applications as a standard in this area [8], [9]. However, most of these software solutions use decision diagrams only internally and do not offer a possibility of storing the actual, possibly optimized, decision diagram of a given function for eventual further use or exchange with other similar software. In our best knowledge there is no commonly accepted standard for these specific tasks.

This state of affairs severely limits interoperability and comparison of performance of various software solutions. This article is meant as the first step towards resolving this problem.

We propose a robust and flexible standard based on XML for storing and exchange of various types of decision diagrams.

The inherent properties of XML permit us to develop a framework flexible enough to encapsulate the various

existing forms of decision diagrams [1], [2], [4]. It also provides us with possibility to further extend this standard in future to accommodate possible new forms of decision diagrams.

The existence of XML parsers for all major programming platforms makes the implementation of this framework for any purpose easy. The XML SOAP (Simple Object Access Protocol) technology aimed exactly at handling data with tree like hierarchical structure provides many ready made functions for manipulation and handling of such data.

2. Basic Concepts

In this section, we will briefly present basic notions of decision diagrams and XML that will be used in following sections.

2.1 Decision diagrams

Decision diagrams for representation of discrete functions are derived by reduction of decision trees through the application of a set of properly formulated reduction rules [1], [7], [10]. The rules for reduction of decision trees depend on the type of the decision tree in question [7]. In general, the reduction is performed by identifying identical sub-trees in the decision tree. Only one copy of the sub-tree is kept. Edges connecting nodes of the removed sub-tree are rearranged to point to the remaining copy.

Formally, a decision diagram is an acyclic directed graph consisting of nodes and edges that connect them. Nodes must store information about the incoming and outgoing edges as well as some additional information, depending on the type of the decision diagram. Each decision diagram has one root node representing the beginning of the directed graph, and a set of terminal nodes. From the point of view of data structure, terminal nodes differ from non-terminal ones only in the fact that they do not have any outgoing edges.

Different types of decision diagrams are defined depending on the number of outgoing edges per node permitted, and other criteria [1], [2], [4], [5], [6], [10], [11].

In this paper, we will focus on those elements of the structure that are common to all types of decision diagrams rather than on any particular type, which ensures

universality of the framework and its applicability to various diagrams.

2.3 XML basic concepts

XML [3], [12] represents a special purpose markup language. It is intended for the description of formats for recording various kinds of data with a complex internal structure and possibly mixed information content.

Rather than trying to specify a rigid data structure for any possible kind of data, and for each particular application, XML specifies a simple set of rules that an XML based file format should use in order to be understood by XML enabled software.

Each XML document is a structured text, while the information it contains is presented in the form of hierarchically arranged elements. The type of elements and their internal structure is specified separately according to syntax XML rules.

3. XML Framework for Decision Diagrams

The XML framework proposed in this paper consists of several separate components. We will list and briefly describe the purpose of each of them. The structure of some of the components is explained in detail in later sections.

The main components of the XML framework introduced are:

1. *XML Schema*, a description of the structure of XML documents containing a decision diagram.
2. *XML document* representing the decision diagram created based on the XML Schema for decision diagrams.
3. *XML parser*, a software component capable of reading and writing of decision diagrams according to specifications of the proposed XML Schema.

Fig. 1 shows the relationships of these components in the XML based framework. In the following, we describe in detail the first and most important component of the proposed XML framework.

The second component is in fact a simple XML document that conforms to specifications of the given XML Schema and needs no further clarification.

The third component represents the implementation of the described standard. It is meant to be a part of a decision diagram processing software. It is built on the basis of a standard XML parser and converts decision diagrams from the internal format of a particular application to the XML specified format.

The structure of this component depends on the software platform and application it will be the part of. XML parsers exist for all major platforms and programming environments [12].

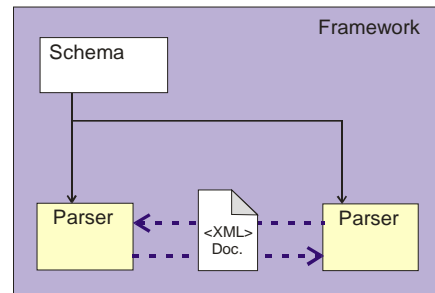


Fig. 1 Components of XML framework

4. Data Structures for Representation of Decision Diagrams

The XML framework we propose is designed to represent decision diagrams purely as directed graphs. It focuses on the representation of decision diagrams in general, i.e., without imposing any restriction on the number of incoming or outgoing edges to each node. Furthermore, it does not deal with the process of reduction of decision tree or with the set of reduction rules applied.

It describes the common structure that all the types of decision diagrams share. It also provides facilities for storing of information that is specific to a particular type of decision diagram.

To accommodate these demands, we propose the following data structure architecture.

Information about parents and descendants of each node is stored in the form of two separate, structurally identical linked lists. Each element of such a linked list consists of two fields:

1. A pointer to the parent or child node related to the node in question, which represents a real link existing in the decision diagram.
2. A pointer to another field in the linked list which serves a function of internal navigating through the list.

Two such linked lists are created and attached to each node in a decision diagram, representing parents and descendants of that particular node as shown in Fig. 2

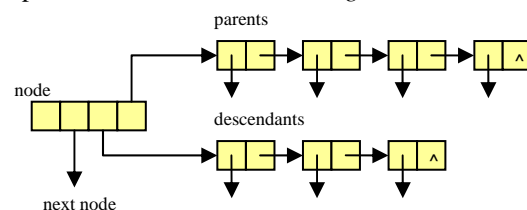


Fig. 2. Diagram of Node data structure

The nodes themselves are also stored in the form of linked lists. Links between elements of these lists do not have any

semantic meaning in the context of the decision diagram itself, but represent just means of accessing elements in the data structure. Fig 3. gives a better view of data structures discussed.

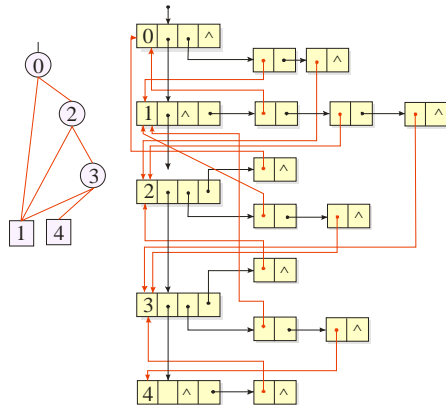


Fig 3. BDD and the corresponding data structure.

An additional attribute field of each node element is dedicated to storing any additional information, i.e., a numerical value, matrix, function, etc., that we want to associate with the node.

This architecture is identical to the standard way graphs are usually described in data structure theory. That permits us to use the standard, well adapted and efficient algorithms for addition and removing of nodes, accessing elements of the graph, etc. It is also consistent with the usual way decision diagram have been represented in various previous applications aimed at dealing with them [5], [8], [9].

5. XML Implementation

This section deals with specific XML implementation of decision diagram data structures described in the previous section. We use the mechanisms of the XML Schema system to specify the custom XML data types that correspond to elements of our architecture.

The core element of the described architecture is a singular node of the decision diagram. In our XML schema specification we define a basic complex element type `NodeType`.

```
<xsd:complexType name="NodeType">
  <xsd:sequence>
    <xsd:element name="next" type="NodeType"
      minOccurs="0" maxOccurs="1"/>
    <xsd:element name="children"
      type="PointType" minOccurs="0" maxOccurs="1"/>
    <xsd:element name="parents" type="PointType"
      minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
  <xsd:attribute name="ID" type="xsd:ID"/>
  <xsd:attribute name="Level"
    type="xsd:integer"/>
  <xsd:attribute name="Value"
    type="xsd:integer"/>
</xsd:complexType>
```

The linked list of nodes is represented by the use of a recursive declaration in this element type where each

`NodeType` element has one nested element of the same type in its structure, i.e., the following element in the list of nodes. Each node element contains also two nested objects of `PointType` type which represent the links to the roots of parents and descendants linked lists.

The additional information about the node, i.e., the ID and the level to which node belongs are stored in appropriate attribute elements.

`PointType` elements are defined in much the same. We use the same mechanism of nested elements to represent links in the linked lists as each `PointType` element of a linked list has one element of the same type as a nested object.

```
<xsd:complexType name="PointType">
  <xsd:sequence>
    <xsd:element name="next" type="PointType"
      minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
  <xsd:attribute name="point" type="xsd:integer"/>
</xsd:complexType>
```

This way of representing element links and hierarchy is native to XML and it is understood and supported by all XML parsers permitting us to use this mechanism to automatically generate corresponding data structures based on the XML document.

However, these links represent only the links that are a part of the structure we designed to represent a decision diagram and are not inherent to the decision diagram itself. The possible complexity of node connections in the decision diagram forces us to use another method. Each `PointType` element contains a field carrying the ID of the actual node to which the actual decision diagram link should point to.

These connections cannot be automatically stored or generated in the XML and need to be reestablished after an XML document has been parsed. The main reason for this is a possibility of establishing a loop in the graph which would pose an impassable obstacle for any XML parser. Although close loops are not permitted in acyclic directed graphs, there is no simple way of specifying this in terms of data structure itself since data structure theory does not make strict distinction between an acyclic and cyclic graph.

The control of this must be implemented separately during the process of creation of the decision diagram. The functions for reestablishing of the real node connections in the graph must also be implemented separately based on the XML parser.

The `TreeType` element represents a wrapper element whose task is to encapsulate the decision diagram structure itself and store the additional information regarding the nature of the decision diagram, i.e., the type of the decision diagram, number of variables, nodes, levels, edges, decomposition rules applied, etc. This element stores

additional information either in the form of XML attributes or additional elements.

```
<xsd:complexType name="TreeType">
  <xsd:all>
    <xsd:element name="root" type="NodeType"
minOccurs="1" maxOccurs="1"/>
  </xsd:all>
  <xsd:attribute name="TType"
type="xsd:string"/>
</xsd:complexType>
```

Furthermore, not all of these features need to be specified in advance. Additional storage elements can be declared and included in each separate XML file since XML itself provides facilities for handling the elements that are not previously specified.

This ensures flexibility of the framework and makes possible to deal with different types of decision diagrams for various classes of discrete functions, and open for further extensions and generalizations.

Extensions to the decision diagram XML Schema can also be included. In this way, the XML framework can be specifically tailored for every application.

6. Conclusion

In this paper, we have pointed out the lack of a standard for storage and exchange of data represented in the decision diagram form. We have presented an elegant and efficient solution based on the XML.

Inherent properties of XML permit us to develop the framework flexible enough to encapsulate many various existing forms of decision diagrams [7]. It also provides us with a possibility to further extend this standard in future to accommodate possible new forms of decision diagrams.

The existence of XML parsers for all major programming platforms would make the implementation of this framework for any purpose very easy.

The possibilities for practical applications of standardized framework are numerous. For instance, it permits exchange of generated and possibly optimized diagrams between various applications. Another possible application of this framework is to convert some of optimal forms of decision diagrams of the standard benchmark functions in the XML based format [6] in order to make the task of evaluation of results of various applications easier and more accurate.

Furthermore, if a logic function can be in the minimal form represented with its optimal decision diagram only that decision diagram needs to be stored and transmitted. This feature of decision diagrams can yet prove to be of great value for future applications.

This solution could be linked on various levels with other members of the broad XML family, such as SVG and MathML which would provide various new possibilities interesting primarily for scientific workers in this field, as instant rendering of a decision diagram in SVG vector format, for example.

References

- [1] Bryant, R.E., "Graph-based algorithms for Boolean functions manipulation", *IEEE Trans. Comput.*, Vol. C-35, No. 8, 1986, 667-691.
- [2] Drechsler, R.; Becker, B., "Ordered Kronecker functional decision diagrams - a data structure for representation and manipulation of Boolean functions", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 17, No. 10, 1998, 965 – 973.
- [3] Harold E. R., Means W. S., *XML in a Nutshell, 2nd Edition*, O'Reilly, 2002.
- [4] Hasan Babu, H.M.; Sasao, T., "Representations of multiple-output switching functions using multiple-valued pseudo-Kronecker decision diagrams", *Proc. 30th Int. Symp. on Multiple-Valued Logic*, May 23-25, 2000, 147 - 152.
- [5] Nagayama, S., Sasao, T., "Compact representations of logic functions using heterogeneous MDDs," *Proc. 33rd International Symposium on Multiple-Valued Logic*, Tokyo, Japan, May 15-18, 2003, 247-252.
- [6] Sasao, T.; Butler, J.T., "A design method for look-up table type FPGA by pseudo-Kronecker expansion", *Proc. 24th Int. Symp. on Multiple-Valued Logic*, May 25-27, 1994, 97 – 106.
- [7] Sasao, T., Fujita, M., (ed.), *Representations of Discrete Functions*, Kluwer Academic Publishers, 1996.
- [8] Somenzi, F., "Efficient manipulation of Decision diagrams", *Int. Journal on Software Tools for Technology Transfer*, 3, 2001, 171-181.
- [9] Somenzi, F., "CUDD Decision diagram Package", <http://bessie.colorado.edu/~fabio/CUDD>
- [10] Stankovic, R.S., Astola, J.T., *Spectral Interpretation of Decision diagrams*, Springer, 2003.
- [11] Stankovic, R.S.; Drechsler, R., "Circuit design from Kronecker Galois field decision diagrams for multiple-valued functions", *Proc. 27th International Symposium on Multiple-Valued Logic*, May 28-30, 275-280, 1997.
- [12] "W3 XML Primer", www.w3.org