ProbModelXML. A format for encoding probabilistic graphical models

M. Arias **F. J. Díez** **M. A. Palacios**
Dept. Artificial Intelligence, UNED
Juan del Rosal 16, 28040 Madrid, Spain

Version 0.2.0 (March 13, 2012)

Abstract

ProbModelXML is an XML format for encoding probabilistic graphical models, with a special emphasis on dynamic models. The main advantages of this format are that it can represent several kinds of models, such as Bayesian networks, Markov networks, influence diagrams, LIMIDs, dynamic Bayesian networks, MDPs, POMDPs, DLIMIDs, etc., and the possibility of encoding new types of networks and user-specific properties without the need to modify the format definition.

Contents

1 Introduction **3**
1.1 Antecedents: Formats proposed previously 4
1.1.1 Formats for Bayesian networks and influence diagrams 4
1.1.2 Specific formats for Markov decision processes 6

2 Basic definitions and notation **7**
2.1 Probabilistic graphical models 7
2.2 Links and paths 7
2.3 Types of variables 8
2.4 Dynamic models 9
2.5 Use of information arcs 10

3 Overview of the format **10**
3.1 File content 12
3.2 Version numbers 12

4 Specification of probabilistic networks **13**
4.1 Network properties . . . 13
4.1.1 Network type . . . 13
4.1.2 Constraints . . . 14
4.1.3 Comment . . . 15
4.1.4 Decision criteria . . . 15
4.1.5 Agents . . . 16
4.1.6 Language . . . 16
4.1.7 Additional properties . . . 16
4.2 Variables . . . 18
4.2.1 Domain of a finite-states variable . . . 19
4.2.2 Domain of a numeric variable . . . 19
4.2.3 Domain of a discretized variable . . . 20
4.3 Links . . . 22
4.4 Potentials . . . 23
4.4.1 Uniform . . . 24
4.4.2 Table . . . 25
4.4.3 Delta . . . 30
4.4.4 Tree and ADD . . . 31
4.4.5 ICI model . . . 38
4.4.6 Sum and product . . . 39
4.4.7 Linear combination . . . 40
4.4.8 Logistic regression . . . 41
4.4.9 Conditional Gaussian . . . 43
4.4.10 Exponential and mixture of exponentials . . . 44

5 Special types of networks **48**
5.1 Dynamic models . . . 48
5.1.1 Network tags . . . 48
5.1.2 Dynamic variables . . . 49
5.1.3 Potentials for dynamic models . . . 49
5.2 Decision analysis networks (DANs) . . . 53

6 Additional information **55**
6.1 Inference options . . . 55
6.1.1 General inference options . . . 55
6.1.2 Inference options for dynamic models . . . 55
6.2 Evidence . . . 56
6.3 Policies and strategies . . . 56

7 Discussion **56**
7.1 About parsers and writers . . . 56
7.2 Future work . . . 57

8 Conclusion **57**

Appendix A Constraints used in OpenMarkov **58**
A.1 Constraints about nodes and variables 58
A.2 Constraints about links (structure of the graph) 60

Appendix B Changelog **62**

1. Introduction

A probabilistic graphical model (PGM) consists of a probability distribution and a graph, such that each node in the graph represents one of the variables on which the probability is defined, and the structure of the graph imposes some properties of independence on the probability distribution. Some PGMs are purely probabilistic, such as Bayesian networks and Markov networks [34], while others, such as influence diagrams [22] and decision analysis networks [15], include decisions and utilities.

There are two main types of temporal PGMs: dynamic models and event networks. Dynamic models [13, 32] discretize time in intervals of a fixed duration (cycle length) and create an instance of each variable for each time period. Therefore, in the context of PGMs "dynamic" is a synonym of "periodic". In contrast, in an event network, each variable represents an event and the values that the variable can take on represent the time at which the event may occur [2, 19, 20].

Dynamic PGMs are a generalization of more simple Markovian models proposed several decades earlier [30]. Thus, dynamic Bayesian networks [13, 32] extend both Markov chains [30] and hidden Markov models [4] by allowing that the state of the system be represented by a set of variables rather than by a single variable. In the same way, Markov Decision Process (MDPs) [5] are extended by factored MDPs [8, 9]) and Partially Observable Markov Decision Process (POMDPs) [3] are extended by factored POMDPs [10] and dynamic LIMIDs [17, 43].

Several formats have been developed for encoding PGMs , but almost all of them are designed for a single software tool. One exception is Fabio Cozman's XMLBIF (see Sec. 1.1.1.c), that has been implemented by several software tools, but it is restricted to Bayesian networks containing only discrete variables, with a very limited set of features. Another exception was DSC, proposed by Microsoft as a standard format for Bayesian networks and influence diagrams, that would receive contributions from the UAI (uncertainty in artificial intelligence) community; however, some time later Microsoft removed the web pages of DSC and developed a new XML format, MSBNx, limited to Bayesian networks.

For this reason, we decided to develop a new format for PGMs, satisfying the following requisites. First, it should be designed as a common language for several research groups and several software tools. As a consequence, the format should accommodate different types of models and a wide range of properties. However, given that it is impossible to foresee from the beginning all the needs that will emerge in the future, the format should be extensible, i.e., it should be able to represent new types of models and new properties without changing the

specification of the format. Second, the syntax and the semantics of the format should be clearly documented, in order to avoid ambiguities and misinterpretations. Third, the syntax of the format should be based on the Extensible Markup Language (XML) specification produced by the World Wide Web Consortium (W3C)[1], mainly because XML is much easier to parse than other types of syntaxes; in fact, there exist many utilities for parsing XML from several programming languages (Java, C++, etc.). There are also several utilities for writing XML files from those languages, as well as other tools for specifying XML formats and for validating them: DTD, XML Schema (XSD), Relax NG, ISO DSDL, etc.

The format proposed in this paper is called ProbModelXML.[2] It presents two main advantages with respect to previous proposals. First, it can encode several types of PGMs: its current version includes Bayesian networks, Markov networks, influence diagrams, LIMIDs, dynamic Bayesian networks, MDPs, POMDPs, and DLIMIDs, and it also permits to encode new models by combining the existing *constrains* or by defining new ones (see Sec. 4.1.2). The second advantage is that it that can encode user-specific features by using the **`AdditionalProperties`** tag (see Sec. 4.1.7).

The rest of the paper is structured as follows: in Section 1.1 we review preceding XML formats for PGMs. Section 2 presents the basic notions of PGMs. Section 3 contains an overview of ProbModelXML. The next sections explain how to encode different types of information: probabilistic networks (Sec. 4), inference options (Sec. 6.1), evidence (Sec. 6.2), policies (Sec. 6.3). There are two sections devoted to specific types of models: decision analysis networks (Sec. 5.2) and dynamic models (Sec. 5.1). We discuss the advantages and limitations of the new format in Section 7 and conclude in Section 8.[3]

1.1. Antecedents: Formats proposed previously

Several formats have been developed for probabilistic graphical models (PGMs) and Markov decision processes (MDPs). In this section we review briefly those that are more related to the ProbModelXML format proposed in this paper.

1.1.1. Formats for Bayesian networks and influence diagrams

a) DNET (Netica)

DNET was developed by Norsys Software Corp. as the default format for their software package, Netica.[4] This format can represent Bayesian networks, influence diagrams, MDPs and POMDPs,

[1] See www.w3.org/XML.

[2] Its original name was CarmenXML, because it was the default format for an open-source tool called Carmen [1]. When the name of the tool changed from Carmen to OpenMarkov, the name of the format was changed to OpenMarkovXML. In May 2011, we changed it again into ProbModelXML, because since the beginning we intended to propose it as a common format for the interchange of PGMs. Therefore we thought that the name of our format should not be attached to any particular software tool.

[3] The web page www.cisiad.uned.es/ProbModelXML contains additional information, including several networks encoded in this format, and possibly an updated version of this document.

[4] See www.norsys.com/netica.html.

with both discrete (finite-states) or continuous variables.[5] The specification of this format is available at:
www.norsys.com/downloads/DNET_File_Format.txt

b) Elvira

Elvira [18] started in 1997 as a join project of several Spanish universities.[6] The Elvira format, which uses a C-like syntax, can represent Bayesian networks and influence diagrams with continuous and finite-state variables, canonical models [14], uncertain parameters (defined by intervals), etc. The specification of the Elvira format can be found at
leo.ugr.es/elvira/devel/Formato/formato.html
Some of the features of the Elvira format were inspired on DNET (Netica's format), and in turn many of the features of ProbModelXML are inspired on the Elvira format.

c) XMLBIF

It was proposed by Fabio Cozman, with suggestions from Marek Druzdzel, Daniel García, and others.
www.poli.usp.br/p/fabio.cozman/Research/InterchangeFormat
This format is restricted to the representation of Bayesian networks with finite-state variables. It is the default format for Fabio Cozman's JavaBayes tool.[7] Weka[8] and many of the tools for PGMs can read and write Bayesian networks in this format.

d) XBN

It is an XML format proposed by Microsoft as the default format for their Microsoft Bayesian Network (MSBNx) tool, thus replacing the previous format DSC, which was not XML.[9] This format can only encode Bayesian networks.
research.microsoft.com/en-us/um/redmond/groups/adapt/msbnx/msbnx/File_Formats.htm
http://xml.coverpages.org/xbn.html

e) XDSL

It is the default format for SMILE and GeNIE, two programs developed by Marek Druzdzel's group at the University of Pittsburgh.[10] SMILE is the inference engine. GeNIE, which offers an intuitive and powerful GUI for Windows, is a front-end for SMILE. The XDSL format has a

[5] Unfortunately, this specification document, which is very clear in general, does not explain how to encode MDPs and POMDPs.

[6] See leo.ugr.es/elvira and www.ia.uned.es/~elvira.

[7] See www.pmr.poli.usp.br/ltd/Software/javabayes.

[8] See www.cs.waikato.ac.nz/ml/weka.

[9] See research.microsoft.com/en-us/um/redmond/groups/adapt/msbnx.

[10] See genie.sis.pitt.edu.

simple version for SMILE and an extended version for GeNIE. The XML schemas (XSDs) for this format can be downloaded from
genie.sis.pitt.edu/SMILEHelp/Appendices/XDSL_File_Format_-_XML_Schema_Definitions.htm

The schema for GeNIE can be seen in HTML at
www.openmarkov.org/OpenMarkovXML/temp/genie/genie-xsd.html
Similarly to DNET (Netica's format), XDSL can represent Bayesian networks, influence diagrams, and some dynamic models, with both continuous and finite-state variables. It can also represent canonical models.

1.1.2. Specific formats for Markov decision processes

a) Cassandra's format

Anthony Cassandra has used a format for flat (i.e., non-factored) POMDPs, that is available at the following links:
www.cassandra.org/pomdp/code/pomdp-file-grammar.shtml
www.cassandra.org/pomdp/code/pomdp-file-spec.shtml
www.cassandra.org/pomdp/examples

The software package Perseus [39] can also read files in Cassandra's format, by using this MATLAB parser:
staff.science.uva.nl/~mtjspaan/software/pomdp.

b) SPUDD

SPUDD, which stands for "Stochastic Planning using Decision Diagrams", is a computer program for evaluating MDPs and POMDPs [21]. The format used by this program is able to represent factored MDPs and POMDPs; potentials are represented by algebraic decision diagrams (ADDs), which we describe in Section 4.4.4.b.

Some examples of POMDPs are included in a `tar.gz` file, together with SPUDD C++ source code, which is available at:
www.computing.dundee.ac.uk/staff/jessehoey/spudd/index.html
(The examples are contained in files with the extension `.txt`.)

The tool Symbolic Perseus,[11] by Pascal Poupart [35], uses a subset of the SPUDD format, whose grammar is specified in this file:
www.cs.uwaterloo.ca/~ppoupart/software/symbolicPerseus/problems/SYNTAX.txt
Some examples encoded in this simplified version of SPUDD can be found at
www.cs.uwaterloo.ca/~ppoupart/software/symbolicPerseus/problems.

c) PomdpX

It is an XML format for POMDPs, developed at the University of Singapore:

[11] See www.cs.uwaterloo.ca/~ppoupart/software.html. The file `ParseSPUDD.java` contains the parser for the SPUDD format.

bigbird.comp.nus.edu.sg/pmwiki/farm/appl/index.php?n=Main.PomdpXDocumentation
It admits flat POMDPs, as well as MOMDPs. In a MOMDP the state space is factored into two variables, X (observable) and Y (unobservable), and there is a third variable in each time slice, O, which provides indirect information about Y [33].

There is a companion XML format for representing the policy obtained when evaluating a POMDP:
bigbird.comp.nus.edu.sg/pmwiki/farm/appl/index.php?n=Main.PolicyXDocumentation

2. Basic definitions and notation

In this paper we denote the variables by capital letters, such as X, and their values by the corresponding lowercase letters, such as x. A set of variables is denoted by a bold capital letter, $\mathbf{X} = \{X_1, \ldots, X_n\}$, and a configuration by a bold lowercase letter, $\mathbf{x} = (x_1, \ldots, x_n)$.

2.1. Probabilistic graphical models

A probabilistic graphical model (PGM) consists of a set of variables, $\mathbf{V}$, a graph $\mathcal{G}$ such that each node represents a variable in $\mathbf{V}$, and a probability distribution $P(\mathbf{v})$ that satisfies certain properties of independence dictated by the structure (the links) of the graph [34]. Some PGMs designed for decision analysis have, additionally, utility functions.

Given the one-to-one correspondence between variables and nodes, we will refer to them indifferently, and use the same letter to represent a node and the variable it represents.

2.2. Links and paths

A *link* in a graph is a pair of nodes, (X, Y), which may be ordered or not. In the first case, we say that the link is *directed* and denote it as $X \to Y$; in the graphical representation, it is drawn as an arrow from X to Y. In the second case, we say that the link is *undirected* and denote it as $X - Y$; in the graphical representation, it is drawn as a line between X and Y. Link $X - Y$ is the same as $Y - X$, while $X \to Y$ is different from $Y \to X$.

A *path* is an ordered set of links, complemented with an ordering of the nodes in each link such that the second node in a link is the same as the first node in the next. Given a directed link in a path, if the order of its nodes in the path is the same as the order in the definition of the link, we say that the path traverses the link *forwards*; otherwise, we say that the path traverses the link *backwards*. For example, the path $X \to Z \leftarrow Y$ traverses the link $X \to Z$ forwards and the link $Y \to Z$ backwards.

A path is *closed* if the second node in the last link is the same as the first node in the first link. Given a closed path, if we move its first link to the last place and shift the i-th link to the $(i-1)$-th position, the new path is considered to be equal to the first. For example, the three closed paths, $A \to B \to C \to A$, $B \to C \to A \to B$, $C \to A \to B \to C$, are considered to be equal.

If a closed path traverses all its directed links forwards, then it is called a *cycle*; otherwise it is a *loop*. Figure 1 illustrates the difference between cycles and loops.

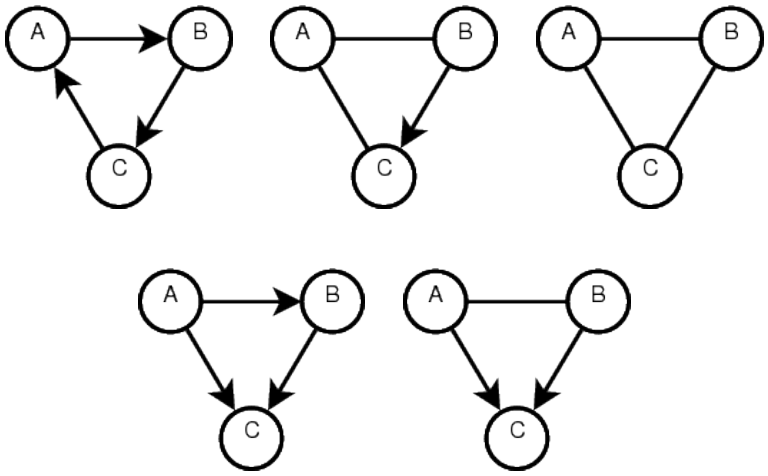

Figure 1: Closed paths: three cycles (upper row) and two loops (lower row).

A *self-loop* is a link whose nodes are identical; for example, A — A or $A \rightarrow A$. It can also be defined as a closed path consisting of only one link.

2.3. Types of variables

Variables can be classified according to their domains. Following the terminology of the format Elvira, *finite-states variables* are those whose domain is given by a finite set of values. These variables are sometimes called *discrete*, but this term is not accurate, because the domain of a discrete variable is not necessarily bounded in advance; for instance, the number of books in a library is discrete because it cannot take on non-integer values, but it should not be represented by a finite-state variable. Finite-state variables are also similar to *categorical variables*, but this term is generally used in opposition to *ordinal variable*, while finite-state variables are ordered in many cases.

A variable is *numeric* when it represents the result of a measurement, including the counting of the number of elements in a set (its cardinal). A *continuous* variable can take on any value in a given interval; therefore, it is the opposite of *discrete* variable. However, a numeric variable can be discretized by defining a set of intervals. When the number of intervals is finite, the discretized variable can be treated for some operations as if it were a finite-state variable.

In practice, the set of intervals is defined by a set of thresholds. A *threshold* in ProbModelXML is essentially the same as a *level* in DNET (Netica's format); the only difference is that ProbModelXML specifies whether the number used to establish the threshold belongs to the left or to the right interval. For example, there are two ways of partitioning the real numbers using 0 as a cutting point: $\{(-\infty, 0], (0, +\infty)\}$ and $\{(-\infty, 0), [0, +\infty)\}$. In the first partition, 0 belongs to the interval on the left, while in the second it belongs to the one on the right. In Section 4.2.3 we explain how to specify in ProbModelXML the thresholds used to discretize numeric variables.

2.4. Dynamic models

As mentioned in the introduction, there are several types of dynamic PGMs. In ProbModelXML we have included six of them: dynamic Bayesian networks [13, 32], simple Markov models [16], factored MDPs [8, 9]), factored POMDPs [10], Dec-POMDPs [6], and dynamic LIMIDs [17, 43].

In dynamic models, some variables are indexed by time: $t \in \{0,\ldots,h\}$, where h is the horizon of the model, which can be infinite.[12] If there is a variable X^t for a certain t, then there is also a variable $X^{t'}$ for each $t' \in \{0,\ldots,h\}$—see Figure 2, where the number between brackets denotes the index t. The subgraph that contains all the nodes X^t having the same temporal index t and the links between them is known as the t-th *time slice*.

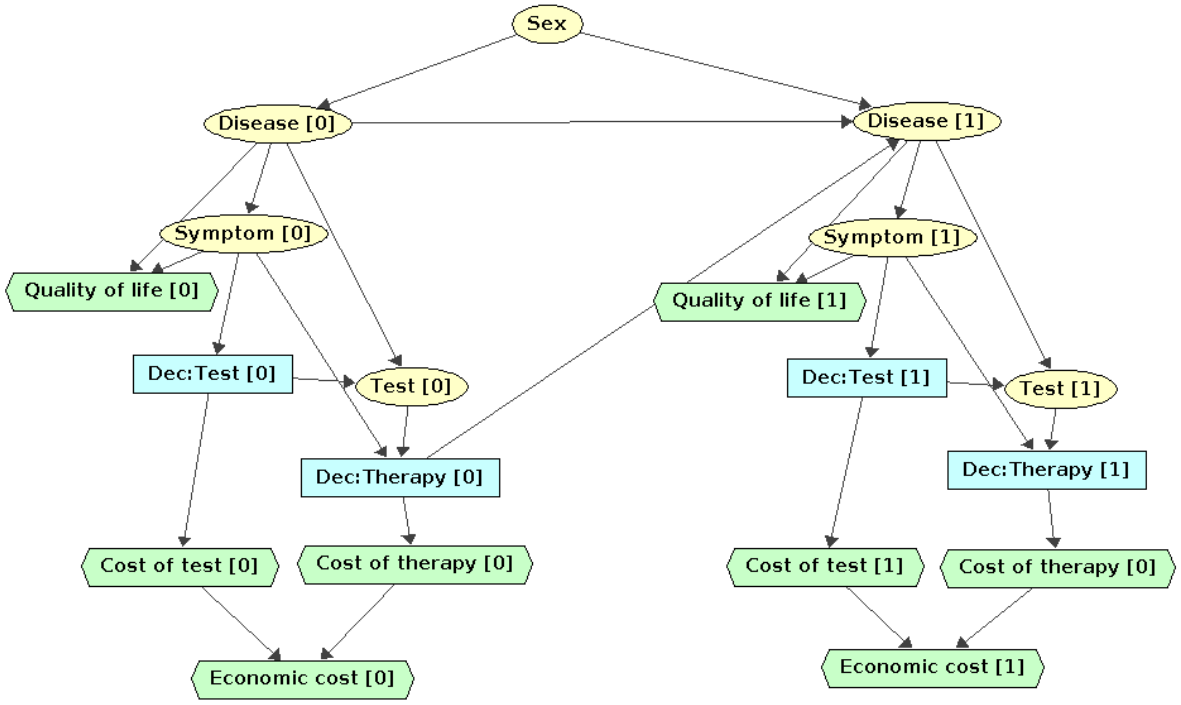

Figure 2: Compact representation of a simple Markov model: a stationary first-order model can be represented by just two time slices. Ovals represent chance variables, rectangles represent decisions, and hexagons represent utilities.

A constraint of dynamic models is that they can not have links to the past, i.e., of the form $X^t \rightarrow Y^{t'}$ with $t > t'$. If there is a $k \in \mathbb{N}^+$ such that every link $X^t \rightarrow Y^{t'}$ satisfies that $t' - t \leq k$ (which means that all the parents of a node $Y^{t'}$ are in the t'-th slice or in the previous k slices), we say that the model is *of k-th order*. In practice, it is usual to work with first-order models ($k = 1$).

[12] In standard MDPs and POMDPs *all* the variables are indexed by time. However, we admit the possibility of atemporal variables, whose value does not change with time, and therefore do not need to have an instance of each of them in each time slice. For instance, the variable Sex in Figure 2 is atemporal.

A variable X is *stationary after* k if it has the same neighbors and the same potentials (probabilities and utilities) after the k-th slice. Put formally, for each $t > k$,

- there is a link between X^t and Y^t if and only if there is a link of the same type between X^k and Y^k;
- if there is a conditional probability $P(x^k|pa(X^k))$, then $P(x^t|pa(X^t)) = P(x^k|pa(X^k))$;
- if there is a utility function $U^k(pa(U^k))$, then $U^t(pa(U^t)) = U^k(pa(U^k))$;

and some of these properties are not fulfilled for any $k' < k$.

A dynamic model is *stationary after* k if all its variables are stationary after k and there is no other $k' < k$ that makes the model stationary. Such a model admits a *compact representation* that contains only the first k slices. Stationary first-order models, which constitute the most common case, can be represented by only the first two time slices, as in Figure 2. MDPs and POMDPs are almost always represented in this compact form. If the horizon h of the DLIMID is finite, we can unroll it completely by cloning the last slice $h-k$ times.

A model that is stationary after k may contain variables that become stationary earlier, i.e., after k', with $k' < k$. The repeated nodes can be removed from the compact representation, to make it even more concise without loss of information, as we have done in Figure 3, assuming that the probabilities and utilities for all the removed nodes are the same in both time slices. When unrolling the model, we must replace back those nodes and their links.

2.5. Use of information arcs

In a PGM, an information arc is a directed link from a chance or a decision node X to a decision node D. It means that the value of variable X is known when making decision D. Information arcs where used by the first time in influence diagrams [22]. We will use them in the ProbModelXML to indicate which nodes in a factored POMDP [10] or in a DLIMID [43, 17] are observable. For example, in the DLIMID in Figure 2 the link from "Symptom [0]" to "Dec:Test [0]" means that when we make the decision about whether to perform the test, we already know whether the patient has the symptom.

On the contrary there are other models that do not require information links. One type of them are MDPs [5, 9], which assume that all the variables are observable. Other type are decision analysis networks (Sec. 5.2), which use *revelation arcs* to indicate when some variables become observed [15].

3. Overview of the format

The recommended extension for ProbModelXML files is `.pgmx`, which stands for *probabilistic graphical models in XML*.

The skeleton of a ProbModelXML file is:

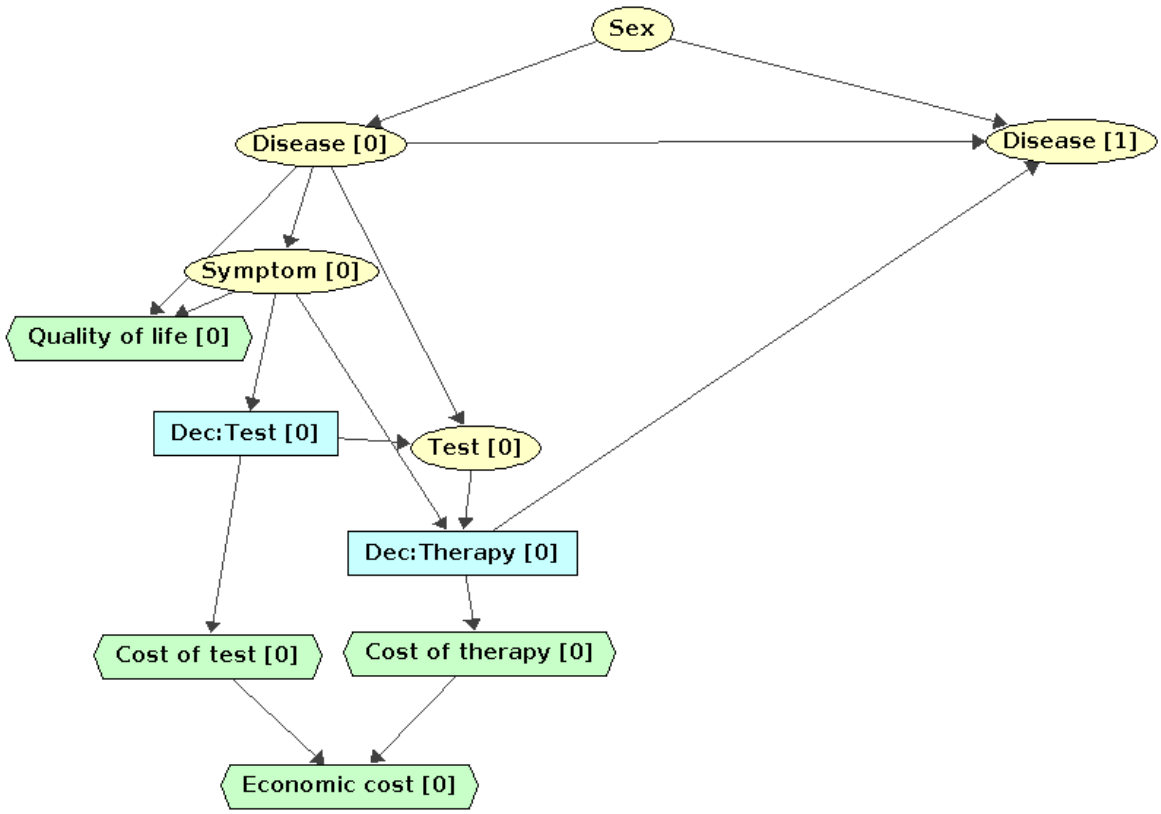

Figure 3: A compact representation, such as the one in Fig. 2, can be made even more concise by omitting the nodes that repeat themselves, with identical potentials, after a certain time slice.

```
<?xml version="1.0" encoding="UTF-8" ?>
<ProbModelXML formatVersion=string >
   <ProbNet type=enumNetworkType />0..1
   <InferenceOptions />0..1
   <Policies />0..1
   <Evidence />0..1
</ProbModelXML>
```

The first line specifies that the file contains an XML document using version 1.0 of the XML specification. The default encoding for ProbModelXML is UTF-8, the same as for XML. Therefore, it is redundant to write encoding="UTF-8", but we prefer to say it explicitly to avoid any confusion. It is allowed to use a different encoding, but we do not recommend it.

The second line indicates that the file uses the ProbModelXML format. The version is a string consisting of three non-negative integers, separated by dots, as explained in Section 3.2.

3.1. File content

In most cases, the content of a ProbModelXML file is a probabilistic network, given by a set of variables, a set of links, a set of potentials, and some additional information (Sec. 4). The *inference options* may include, for example, an inference algorithm chosen by the designer of the network or a near-optimal elimination ordering obtained off-line—see Section 6.1. These options can be stored with the network in order to evaluate it more efficiently.

Each *policy* is associated with a decision; it may be an optimal policy obtained by an algorithm or a sub-optimal policy imposed by the user (for example, to perform what-if reasoning [26])—see Section 6.3. Policies can be stored independently of the network for which they are intended, but in some cases it may be convenient to store them together. For example, a POMDP designed for controlling a robot may be stored in the same file as a near-optimal policy obtained off-line.

The *evidence* consists of one or several evidence cases, each one consisting in turn of one or several findings (Sec. 6.2). In general, evidence should not be stored with the network, but in some cases the user might decide to store them together in the same file to do some experiments.

3.2. Version numbers

As mentioned above, the version is indicated by string consisting of three non-negative integers. A change in the third one (for example, from 1.0.0 to 1.0.1) indicates an extension of the format due to the inclusion of new tags or attributes or to enlarging the domain of some enumerations—for instance, by defining a new type of network or potential. This way, a parser for version $x.y.z$ should also be able to parse a file in version $x.y.z'$, with $z' > z$, unless it encounters a property that it does not recognize. If the extension of the format is due to a new value for an enumeration (for example, a new type of network), the parser should throw an error or an exception. However, when the parser encounters a new tag or a new attribute, it may ignore the unrecognized item, limiting itself to writing a warning to the standard output or to the log file.

An increase in the second integer (for example, from 1.0.0 to 1.1.0) means a change in the syntax that specifies the model. For example, if a version uses the syntax **<Coordinates>***integer integer***</Coordinates>**, and a posterior version uses the syntax **<Coordinates** x=*integer* y=*integer* **/>**, this change would imply an increase in the second of the integers that denote the version. Therefore, a parser for version $x.y.z$ will throw an error or an exception when the file is encoded in version $x.y'.z'$ with $y > y'$; at least, it should warn the user that the parsing may give unpredictable results.

A change in the first integer (for example, from 1.0.0 to 2.0.0) would mean a drastic change in the specification of the ProbModelXML format, which is not expected to occur.

4. Specification of probabilistic networks

In this section we describe how to encode probabilistic networks in the ProbModelXML format. The skeleton for a probabilistic network is as follows:

```
<ProbNet type=enumNetworkType >
  <AdditionalConstraints />0..1
  <Comment />0..1
  <DecisionCriteria />0..1
  <Agents />0..1
  <Language />0..1
  <AdditionalProperties />0..1
  <Variables />
  <Links />
  <Potentials />
</ProbNet>
```

4.1. Network properties

In this section we describe some of the general properties of probabilistic networks.

4.1.1. Network type

The tag **ProbNet** is necessarily followed by an attribute type, which indicates the type of network. It is an enumerate, which in the current version of the format can take the following values (see also Table 5, page 59):

1. BayesianNetwork [34],
2. MarkovNetwork [34],
3. InfluenceDiagram [22],
4. LIMID (limited memory influence diagram) [28],

5. DecisionAnalysisNetwork (DAN) [15],
6. DynamicBayesianNetwork [13],
7. SimpleMarkovModel [16],
8. MDP (Markov decision process) [5], including factored MDPs [8, 9],
9. POMDP (partially observable MDP) [3], including factored POMDPs [10] and MOMDPs [33],
10. Dec-POMDP (decentralized POMDP) [6],
11. DynamicLIMID [42, 43].

Decision analysis networks (DANs), which have specific features not available in other models, are treated in Section 5.2.

Dynamic Bayesian networks, simple Markov models, MDPs, POMDPs, and dynamic LIMIDs are temporal models. They have been explained in more detail in Section 2.4. The syntax for encoding them is given in Section 5.1.

4.1.2. Constraints

The main purpose of constraints is to prevent the user from doing illegal operations at the graphical user interface (GUI), such as giving an empty name to a variable or creating a cycle in a Bayesian network. Most software tools implement these constraints by embedding them in the source code of the GUI. However, in OpenMarkov constraints are objects that can be assigned to networks. Appendix A describes the constraints associated to each network type by definition—see Table 5.

Additionally, in OpenMarkov the user can assign to a network other constraints that are not imposed by the network type; they are denoted by "O" (meaning "optional") in Table 5. For example, OnlyFiniteStates is an optional constraint. An algorithm that applies only to Bayesian networks with finite-state variables must make sure that the network satisfies this constraint. The fastest way to check it is to see whether the network has this constraint assigned as an object; if not, the algorithm can ask the constraint to examine the network, which involves a higher computational cost. Another use of constraints may be, for example, to prevent a learning algorithm from adding more than n parents to a node.

The syntax for assigning optional constraints to a network is as follows:

```
<AdditionalConstraints>
  <Constraint name=string>
    <Argument name=string value=string >0..n
  </Constraint>1..n
</AdditionalConstraints>
```

Example 1. *We might use the following constraint to prevent that a learning algorithm assigns more than 5 parents to each node:*

```
<Constraint name="MaxNumParents">
   <Argument name="numParents" value="5" >
</Constraint>
```

4.1.3. Comment

The tag **Comment** denotes a comment in HTML format. The main reasons for writing the comments in HTML rather than in plain text is the possibility of having formatted text and URLs.

In order to avoid conflicts between OpenMarkov's XML syntax and the HTML comment, it is advisable to use the CDATA option of XML languages.

Example 2. *The comment "This* ***network*** *was* created *by Arias & Díez." can be encoded as follows:*

```
<Comment><![CDATA[
   <html>
      <body>
         This <b>network</b> was <i>created</i>
         by Arias &amp;amp; Díez.
      </body>
   </html>
]]></Comment>
```

Please note that the accented character 'í' can be encoded directly, but it would also be possible to use the HTML encoding "í". The ampersand in "Arias & Díez" cannot be encoded directly because it is a reserved character in HTML.

A comment can also be stored as plain text instead of HTML, but it is still necessary to scape the reserved XML characters, such as '<', '>', and '&', as in the following example:

```
<Comment>This network was created by Arias &amp;amp; Díez.</Comment>
```

4.1.4. Decision criteria

The tag **DecisionCriteria** is used in multicriteria decision making. as follows:

```
<DecisionCriteria>
   <Criterion name=string>
      <AdditionalProperties />0..1
   </Criterion>2..n
</DecisionCriteria>
```

Example 3. *In a probabilistic network for cost-effectiveness analysis we may include the following declaration:*

```
<DecisionCriteria>
  <Criterion name="cost" />
  <Criterion name="effectiveness" />
</DecisionCriteria>
```

Then, each utility variable can be associated to a particular criterion, as explained in Section 4.2.

4.1.5. Agents

The tag **Agents** is used for encoding multi-agent systems. Currently, the only type of multi-agent network in ProbModelXML is Dec-POMDP—see Appendix A, Table 5. Its skeleton is:

```
<Agents>
  <Agent name=string>
    <AdditionalProperties />0..1
  </Agent>2..n
</Agents>
```

Example 4. *In a surveillance system we might have two agents:*

```
<Agents>
  <Agent name="video camera" />
  <Agent name="mobile robot" />
</Agents>
```

Then, each decision variable can be assigned to a particular agent—see Section 4.2.

4.1.6. Language

This tag indicates the language in which the names of variables and states are written. It is possible to indicate the name of a variable or a state in a different language by using the tag **AdditionalProperties**, as shown in Example 6.

4.1.7. Additional properties

This tag permits to extend the ProbModelXML format by representing other properties defined by the user.

An additional property can appear in the context of a *ProbNet*, an *Evidence Case*, a *Criterion* (for multicriteria decision making), an *Agent* (in multi-agent models), a *Variable*, a *State* of a variable, a *Potential* and a *Policy*. In all the cases, the skeleton for encoding additional properties is as follows:

```
<AdditionalProperties>
   <Property name=string value=string />1..n
</AdditionalProperties>
```

The attributes defined in each additional property are:

- name: it is a string but if the additional property comes from another system, the name contains a previous "context" separated from the real name with a point. To translate strings from a language to others we can add a localization suffix with two letters.
- value: a string.

Example 5. *In Elvira each variable has a name, which is a string with some restrictions, for example, that it cannot contain spaces. For this reason, it also offers the possibility of assigning a title to each variable, which is a string without those restrictions. However, in ProbModelXML titles are not necessary, because this format does not impose any restriction on the names of variables. When translating a network from Elvira into ProbModelXML, we may be interested in keeping its title, such that we can later translate it back to Elvira. A way to do it is to store it in ProbModelXML as an additional property:*[13]

```
<Property name="elvira.title" value="X ray result"/>
```

Example 6. *As mentioned in Section 4.1.6, each network has a tag* **`Language`** which indicates in what language the names of variables and states are written. Therefore, when displaying the network in that language (for instance, at the GUI), those names are used. However, if the user prefers to visualize the network in a different language, the GUI will use the name assigned as an additional property, if available. For example, in a network using English as the default language, we may have the following definition:

```
<Variable name="Fever" ...>
   <AdditionalProperties>
      <Property name="name.es" value="Fiebre"/>
      <Property name="name.fr" value="Fièvre"/>
      <Property name="name.de" value="Fieber"/>
   </AdditionalProperties>
   ...
```

When displaying the network in English, the name of this variable will appear as "Fever", when in Spanish, as "Fiebre", etc.

[13] An alternative would be to use Elvira's title as ProbModelXML's name, and store Elvira's name as an additional property:

```
<Property name="elvira.name" value="X_ray_result"/>
```

4.2. Variables

The skeleton for encoding a variable is:

```
<Variable name=string type=enumDomainType role=enumNodeRole>
   <Comment />0..1
   <AdditionalProperties />0..1
   <Coordinates x=integer y=integer />0..1
   <Criterion name=string >0..1
   <Agent name=string >0..1
   <AlwaysObserved>0..1
   specification_of_domain
</Variable>
```

The three attributes of a variable—name, type, and role—are mandatory. The name is a string. The type is an enumerate that can take on three values: FiniteStates, Numeric, and Discretized. The role is an enumerate that can take on three values: Chance, Decision, and Utility.

Comment and **AdditionalProperties** have the same structure and meaning as for probabilistic networks (Secs 4.1.3 and 4.1.7 respectively).

The **Coordinates** give the position of the node in the GUI. Following the Java convention, we assume that the origin of coordinates is the upper left corner, as shown in figure 4. Therefore, a network created with a tool that follows the C++ convention (which places the origin of coordinates in the lower left corner), such as Netica, GeNIE, or Hugin, will look upside down when opened with a Java tool, such as Elvira or OpenMarkov.

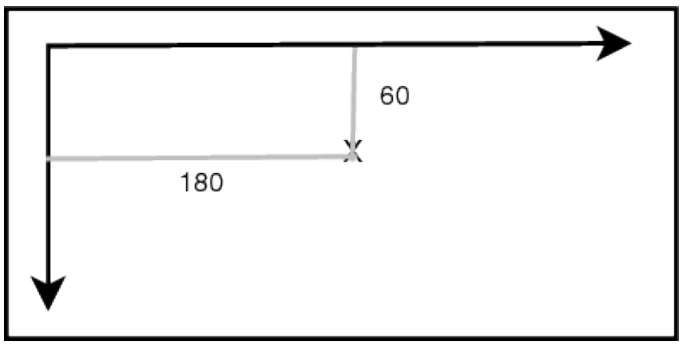

Figure 4: ProbModelXML follows the Java convention of placing the origin of coordinates at the upper left corner.

The **Criterion** tag is present if and only if role="Utility" and the network contains a **DecisionCriteria** tag (Sec. 4.1.4). Obviously, the *string* that denotes the name of the criterion must be one of the criteria defined by the tag **DecisionCriteria** of the network.

Similarly, the **Agent** tag is present if and only if role="Decision" and the network contains an **Agents** tag (Sec. 4.1.5). Obviously, the *string* that denotes the name of the agent must be one of those defined by the **Agents** tag.

The tag **<AlwaysObserved>** is used as a complement of revelation arcs, and therefore it is incompatible with the constraint NoRevelationArcs. By default all types of network have this constraint, except decision analysis networks (see Table 5), which implies that this tag can only be present in this type of network (cf. Sec. 5.2).

The specification of domain for each type of variable is explained in the next subsections.

4.2.1. Domain of a finite-states variable

The domain of a finite-states variable is specified by the tag **<States>**, which encloses a list of states, as follows:

```
<States>
   <State name=string>
      <AdditionalProperties />0..1
   </State>2..n
</States>
```

The tag **AdditionalProperties** plays the same role as in the case of networks and variables.

The order of the states in the ProbModelXML format is relevant, as it will affect the probabilities and utilities assigned to them. We recommend to follow this convention for the most typical domains, which in general implies to order them in increasing severity degrees: "false" < "true"; "absent" < "present"; "no" < "yes"; "negative" < "null" < "positive"; "absent" < "mild" < "moderate" < "severe"; "low" < "medium" < "high".

Example 7. *The domain for a finite-states variable might be as follows:*

```
<States>
   <State name="absent" />
   <State name="present" />
</States>
```

4.2.2. Domain of a numeric variable

The domain of a numeric variable is specified by two thresholds, that define an interval, plus a number that denotes the precision with which its value is measured, as explained below.

```
<Unit>string</Unit>0..1
<Precision>decimalNumber</Precision>
<Thresholds>
   <Threshold value=number belongsTo=enumSide />2
</Thresholds>
```

The tag **<Unit>** specifies the unit of measure which is given by a *string*, for example, "euro" or "second".

The tag **<Precision>** determines the rounding of the numerical values of this variable.

Example 8. ***<Precision>****0.01****</Precision>*** *means that* π *will be rounded down to 3.14; if the precision were 1, 0.1, 0.001, 0.5, 0.25, or 0.05, it would be rounded to 3, 3.1, 3.142, 3.0, 3.25, or 3.15, respectively.*

A threshold separates two intervals. The attribute belongsTo is an enumerate whose possible values are "Left" or "Right". The former indicates that the threshold (its numerical value) belongs to the interval on the left, while the latter means that it belongs to that on the right.

Example 9. *The interval* $[0,1]$ *can be defined as follows:*

```
<Thresholds>
   <Threshold value="0" belongsTo="Right" />
   <Threshold value="1" belongsTo="Left" />
</Thresholds>
```

while the interval $(0,1)$ *would be specified as follows:*

```
<Thresholds>
   <Threshold value="0" belongsTo="Left" />
   <Threshold value="1" belongsTo="Right" />
</Thresholds>
```

Similarly, the interval $(-\infty,+\infty)$ *can be encoded as follows:*

```
<Thresholds>
   <Threshold value="–Infinity" />
   <Threshold value="+Infinity" />
</Thresholds>
```

because when the value of a threshold is $-\infty$ *or* $+\infty$ *the attribute belongsTo does not make sense.*

4.2.3. Domain of a discretized variable

Given that a discretized variable can be viewed as being finite-states and numeric at the same time, its skeleton has the tags of both:

```
<Unit />0..1
<Precision />
<Thresholds>
   <Threshold />3..n
</Thresholds>
<States />2..n
```

A set of i thresholds defines $i-1$ intervals, i.e., $i-1$ states. Given that a finite-state variable must have at least two states (in our opinion, a one-state variable would not make sense), the specification of a discretized variable must contain at least three thresholds ($i \geq 3$). This contrasts with the specification of a numeric variable, which involves only two thresholds because in this case there is only one interval (cf. Section 4.2.2).

Example 10. *A variable whose domain is* $(-\infty,+\infty)$ *can be discretized in three intervals:* $(-\infty,0)$, $[0,0]$, *and* $(0,+\infty)$, *as follows:*

```
<Variable name="Balance" type="Discretized"
      role="Chance" order="Increasing" >
   <Coordinates x="100" y="300" />
   <Unit>euro</Unit>
   <Precision>0.01</Precision>
   <Thresholds >
      <Threshold value="–Infinity"/>
      <Threshold value="0.00" belongsTo="Right"/>
      <Threshold value="0.00" belongsTo="Left"/>
      <Threshold value="+Infinity"/>
   </Thresholds>
   <States>
      <State name="negative"/>
      <State name="null"/>
      <State name="positive"/>
   </States>
</Variable>
```

Please note that there are two thresholds having the same numerical value, *but the attribute* belongsTo *is* "Right" *for the first and* "Left" *for the second; the interval delimited by these thresholds contains exactly one value.*

The attribute declaration order="Increasing" means that the ordering of the states is the same as the ordering of the thresholds. This is the most frequent situation. For example, higher temperatures entail higher degrees of fever. However, there are some cases in which lower values of the numerical measurement entail higher degrees of severity; in this case, we use the declaration order="Decreasing".

Example 11. *In cardiology, the smaller the area of a heart valve, the more severe the stenosis:*[14]

```
<Variable name="Mitral stenosis" type="Discretized"
      order="Decreasing" role="Chance" >
   <Coordinates x="563" y="785" />
```

[14]The 2 in "mm^2" is a special character (number 0xb2 in Unicode), not a superindex, because the argument of the **Unit** tag is plain text, not HTML.

```
    <Unit>mm2</Unit>
    <Precision>0.1</Precision>
    <Thresholds>
      <Threshold value="0.0" belongsTo="Left"/>
      <Threshold value="1.0" belongsTo="Left"/>
      <Threshold value="1.5" belongsTo="Left"/>
      <Threshold value="2.0" belongsTo="Left"/>
      <Threshold value="15.0" belongsTo="Left"/>
    </Thresholds>
    <States>
      <State name="no stenosis"/>
      <State name="mild stenosis"/>
      <State name="moderate stenosis"/>
      <State name="severe stenosis"/>
    </States>
  </Variable>
```

The first state, no stenosis, corresponds to the last interval, $(2.0, 15.0]$*, and vice versa. It might seem more coherent that the order of the states were the same as that of the thresholds but this would contravene our recommendation—in Section 4.2.1—that the order of the states should be absent* $<$ *mild* $<$ *moderate* $<$ *severe, which is important when working with canonical models.*

4.3. Links

The structure of the list of links is similar to that of the list of variables; it is enclosed between the tag **Links**. Its skeleton is as follows:

```
<Links>
  <Link directed=boolean >
    <Variable name=string >2
    <Comment />0..1
    <AdditionalProperties />0..1
    <Potential type=enumPotentialType role="Restrictions" >0..1
    <RevelationConditions>0..1
  </Link>0..n
</Links>
```

The declaration directed="true"/"false" means that the link is directed/undirected.

Example 12. *The link* $A \to B$ *can be represented as follows:*

```
<Link directed="true ">
  <Variable name="A" />
  <Variable name="B" />
</Link>
```

The **Potential** with role="Restrictions" is used to indicate that some values of one of the variables are incompatible with some values of the other (see Section 5.2). This tag cannot be present when the network has the constraint NoRestriction. The tag **RevelationConditions** indicates that some values of the first variable reveal the value of the second (see again Section 5.2). This tag is incompatible with the constraint NoRevelationArc. Given that currently all the network types except decision analysis networks (cf. Table 5 on page 59) have these two constraints, these two tags can only be present when the network type is DecisionAnalysisNetwork.

4.4. Potentials

A potential ψ defined on a set of variables $\mathbf{V}$ is a function that assigns a real number to each configuration $\mathbf{v}$ of $\mathbf{X}$. The skeleton of a potential in ProbModelXML is:

```
<Potential type=enumPotentialType role=enumPotentialRole >
   <Comment />0..1
   <AdditionalProperties />0..1
   specification of the potential
</Potential>
```

ProbModelXML can represent several types of potentials, some of them with subtypes. The subtype is indicated by the attribute *type*, that has a different set of values for each potential type. One optional attribute is *function*, whose value is a String with the function name and only exists when *type*=function.

The enumerate *enumPotentialType* can be any of the potentials defined in the next sections. In the case of a dynamic model, it can also be any of the potentials defined in Section 5.1.3.

The role of a potential is used only for top level potentials, not for the subpotentials, i.e., for potentials that make part of another potential. The enumerate *enumPotentialRole* has these possible values:

1. JointProbability: there is a table or function in the potential that determines the probability of each possible configuration of the variables of the potential.

2. ConditionalProbability: there is a table or probability function of one conditioned variable and between zero and several conditioning variables.

3. Utility: there is a table or function that determines the utility of each possible configuration of the variables of the potential.

4. Policy: defines the value of a variable decision that maximizes an objective.

5. Restrictions: indicates that some pairs of values of the variables involved in a link are incompatible, as explained in Section 5.2.

4.4.1. Uniform

It is the most basic potential. It admits any type of variables: finite-states, numerical, discretized, and any combination of them. When the role is ConditionalProbability this potential represents a family of probability distributions, $P(y|\mathbf{x})$, in which the probability of Y is always the same. In particular, if Y is a finite-states variable,

$$\forall \mathbf{x},\ P(y|\mathbf{x}) = \frac{1}{|Y|}\ ,$$

where $|Y|$ is the number of values of Y; if Y is continuous, then

$$\forall \mathbf{x}, \forall y, \forall y',\ p(y|\mathbf{x}) = p(y'|\mathbf{x})$$

and

$$\forall \mathbf{x},\ \int p(y|\mathbf{x}) \cdot dy = 1\ .$$

When the role is jointProbability this potential represents a probability distribution $P(\mathbf{x})$ such that

$$P(\mathbf{x}) = \frac{1}{|\mathbf{X}|}\ ,$$

where $|\mathbf{X}|$ is the number of configurations of the set of variables $\mathbf{X}$.

When the role is Utility, this potential represents a null utility function:

$$\forall \mathbf{x},\ U(\mathbf{x}) = 0\ .$$

The skeleton of a uniform potential is:

```
<Potential type="Uniform" role=enumPotentialRole >
   <UtilityVariable name=string />0..1
   <Variables>
      <Variable name=string>1..n
   </Variables>0..1
</Potential>
```

Please note that this specification contains no numerical information, because it is not necessary.

The reason for having this type of potential in our format is that when editing a new network with a GUI, it is convenient to assign to each chance or utility node a potential that admits any number and types of variables, does not require any parameter, and does not change when we modify the domain of any of the variables on which the potential is defined.

4.4.2. Table

One of the most basic types of potentials is a table, defined on a set of finite-states or discretized variables.[15] Its skeleton is as follows:

```
<Potential type="Table" role=enumPotentialRole >
   <UtilityVariable name=string />0..1
   <Variables>
      <Variable name=string>1..n
   </Variables>0..1
   <Values>decimal_numbers</Values>
   <UncertainValues>
      <Value />1..n
   </UncertainValues>0..1
</Potential>
```

The tag **UtilityVariable** makes sense only if role="Utility". When role="Conditional-Probability", the conditioned variable is the first in the list of **Variables**.

The tag **Values** denotes the numerical values that define the potential. A constant can be represented as a table that does not depend on any variable; in this case, the tag **Values** contains only one decimal number, as we will see in Example 19. The tag **UncertainValues** is used to give more information about each parameter, as explained below.

a) Order of the values

The values enclosed by the tag **Values** or **UncertainValues** increase in accordance with the ordering of the variables within the tag **Variables**. This remark is important because the ordering of the values within each potential is specific of each format.

Example 13. *The utility potential $U(a,b)$ defined in Table 1 can be encoded as follows:*

a	b	Position in table	$U(a,b)$
no	no	0	–5
yes	no	1	–5
no	yes	2	10
yes	yes	3	80

Table 1: Order of the values in the potential $U(a,b)$.

[15] In fact, any potential that involves only finite-states or discretized variables—such as an ICI potential and, in some cases, a tree, a logistic regression, or a Weibull regression—is equivalent to a table. Therefore, an easy way of doing inference is to convert them to explicit tables, although for some of them there are much more efficient ways to do it.

```
<Potential type="Table" role="Utility" >
  <UtilityVariable name="Cost of test"/>
  <Variables>
    <Variable name="A"/>
    <Variable name="B"/>
  </Variables>
  <Values>–5 –5 10 80</Values>
</Potential>
```

Please note that in Table 1 and in the specification of the **`Values`** *of the potential the first variable that changes is the first variable on which the potential depends, and so on.*

Example 14. *The conditional probability $P(c|a,b)$ defined in Table 2 can be encoded as follows:*

c	a	b	Position in table	$P(c\|a,b)$
no	no	no	0	0.9
yes	no	no	1	0.1
no	yes	no	2	0.7
yes	yes	no	3	0.3
no	no	yes	4	0.9
yes	no	yes	5	0.1
no	yes	yes	6	0.2
yes	yes	yes	7	0.8

Table 2: Order of the values in the potential $P(c|a,b)$.

```
<Potential type="Table" role="ConditionalProbability" >
  <Variables>
    <Variable name="C"/>
    <Variable name="A"/>
    <Variable name="B"/>
  </Variables>
  <Values>0.9 0.1 0.7 0.3 0.9 0.1 0.2 0.8</Values>
</Potential>
```

b) Uncertainty about the parameters (values) of the potential

As mentioned above, the uncertainty about the parameters can be expressed in the form of second order probabilities. The skeleton for defining a **`Value`** within the **`UncertainValues`** tag is as follows:

```
<Value distribution=enumDistributionType name=string>arguments</Value>
```

An empty tag, **<Value/>**, means that there is no uncertainty for the corresponding parameter.

The name of the parameter is optional, but it makes more sense to assign a name to a parameter when we intend to perform sensitivity analysis on it than when the numeric value of a parameter is known with precision.

The types of distributions are shown in Table 3. The *Exact* distribution is used when we know with certainty the value of the parameter, v; properly speaking, this is a Dirac delta distribution, but we believe that the name *Exact* is more intuitive for most users. The *Range* distribution means that the probability of the parameter is uniform inside the interval $[a,b]$ and 0 outside. The *Triangular* distribution implies that the probability of the parameter is zero when $v \leq a$ and when $v \geq b$, and reaches its maximum at $v = c$. *Normal*, *LogNormal*, and *Gamma* are the well-known statistical functions. *Gamma-mv*, where "*mv*" stands for "mean and variance", is an alternative parametrization of the Gamma distribution—see Example 15. The "distribution" *Complement* is used to make the probabilities add to 1; its argument, v, is used to distribute the remaining probability proportionally among all the parameters having a *Complement* distribution, as explained in the Examples 16 and 17. The use of the *Dirichlet* distribution is explained in Example 18.

Distribution	Parameters	Conditions for utilities	Conditions for probabilities	Domain	Mean
Exact	v	none	$0 \leq v \leq 1$	$\{v\}$	v
Range	a,b	$a < b$	$0 \leq a < b \leq 1$	$[a,b]$	$\frac{a+b}{2}$
Triangular	a,b,c	$\left\{ \begin{array}{c} a \leq c \leq b \\ a < b \end{array} \right\}$	$\left\{ \begin{array}{c} 0 \leq a \leq c \\ c \leq b \leq 1 \\ a < b \end{array} \right\}$	(a,b)	$\frac{a+b+c}{3}$
Normal	μ,σ	$\sigma > 0$	incompatible	$(-\infty,+\infty)$	μ
LogNormal	μ,σ	$\sigma > 0$	incompatible	$(0,+\infty)$	$e^{\mu+\sigma^2/2}$
Gamma	k,θ	$k > 0, \theta > 0$	incompatible	$[0,+\infty)$	$k\theta$
Gamma	μ,σ	$\mu > 0, \sigma > 0$	incompatible	$[0,+\infty)$	μ
Beta	α,β	meaningless	$\alpha > 0, \beta > 0$	$(0,1)$	$\frac{\alpha}{\alpha+\beta}$
Dirichlet	α	incompatible	$\alpha > 0$	$(0,1)$	$*$
Complement	v	incompatible	$v > 0$	$[0,1]$	$*$

Table 3: Distributions for specifying second order probabilities ($*$ = the mean of this distribution depends not only on its own parameters, but also on the parameters of other distributions).

Example 15 (Gamma distribution)**.** *Let us assume that the cost of a treatment is 0 when no therapy is applied, and 300 € if we apply a certain therapy. The uncertainty about the cost of the therapy may be represented by a Gamma distribution with a mean $\mu = 300$ and a variance $\sigma = 30$. Using the standard representation of the Gamma, in which $\mu = k\theta$ and $\sigma = k\theta^2$, we have $k = 100$ and $\theta = 3$. The utility potential may be encoded as follows:*[16]

[16]It may seem redundant to give explicitly the **Values** of a potential when they can be obtained from the

```
<Potential type="Table" role="Utility" >
   <UtilityVariable name="Cost of treatment"/>
   <Variables>
      <Variable name="Treatment"/>
   </Variables>
   <Values>0 300</Values>
   <UncertainValues>
      <Value distribution="Exact">0</Value>
      <Value distribution="Gamma" name="cost of therapy">100 3</Value>
   </UncertainValues>
</Potential>
```

The same potential can be encoded using the Gamma-mv distribution, which is more intuitive in this case, because its parameters are the mean and the variance:

```
<UncertainValues>
   <Value distribution="Exact">0</Value>
   <Value distribution="Gamma-mv" name="cost of therapy">300 30</Value>
</UncertainValues>
```

The above example refers to a utility potential, in which there is no constraint among the values of the potential. However, in the case of a joint probability potential $P(x)$ there are some constraints on the types and the parameters of the distributions (the second-order parameters), because the probabilities (the first-order parameters) must add up to 1. Similarly, in the case of a conditional probability potential $P(y|\mathbf{x})$ the probabilities corresponding to the same configuration of $\mathbf{x}$ must add to 1. These constraints can be encoded in the following rules:

1. If one of the distributions is *Exact* with $v \neq 0$, *Range*, or *Triangular*, then:

   - all the others must be either *Exact*, *Range*, *Triangular*, or *Complement*;
   - at least one of the others must be *Complement*;
   - the sum of the maxima of all the distributions (different from *Complement*) cannot be greater than 1.

2. If one of the distributions is a *Beta*, then:

   - all the others must be *Exact* with $v = 0$ or *Complement*;

UncertainValues. However, it is not always clear how to obtain the former from the latter; for example, in the case of a triangular distribution, the value might be either the mean of the distribution, $(a+b+c)/3$, or its maximum, c. Another reason for having both tags in the specification of a potential is that the user might wish to have **Values** inconsistent with the mean of the distributions in **UncertainValues**; we found this situation when building a probabilistic model aimed at reproducing the results of a model built by a different research group. A third reason is that there might be a parser that does not know how to obtain the **Values** from the **UncertainValues**; that parser might read the **Values** and ignore the **UncertainValues** tag; in that case, the network might be used for inference, but not for sensitivity analysis.

- at least one of the others must be *Complement*.

3. If one of the distributions is a *Dirichlet*, then:

   - all the others must be *Exact* with $v = 0$ or *Dirichlet*;
   - at least one of the others must also be a *Dirichlet*.

The next examples illustrate the scenarios described by each of these rules, respectively.

Example 16 (Exact, triangular, and complement distributions)**.** *Let us assume that there are four models of cars. We know that 15% of the cars are of the first model and between 0.08 and 0.12 are of the second model; we also know that the number of cars of the third model is twice the number of cars of the fourth. Then the prior probability for the variable "Car model" can be given as follows:*[17]

```
<Potential type="Table" role="jointProbability" >
   <Variables>
      <Variable name="Car model"/>
   </Variables>
   <Values>0.15 0.10 0.25 0.5</Values>
   <UncertainValues>
      <Value distribution="Exact">0.15</Value>
      <Value distribution="Triangular">0.08 0.12 0.10</Value>
      <Value distribution="Complement">1</Value>
      <Value distribution="Complement">2</Value>
   </UncertainValues>
</Potential>
```

When creating a sampled copy of this potential, the first value will always be 0.15. The second value will be sampled from the triangular distribution, and the remaining probability will be assigned proportionally to the parameters of the Complement distributions. For example, if the sampled value obtained for the second parameter is 0.097, the values for the sampled potential will be (0.15, 0.097, 0.251, 0.502).

Example 17 (Beta and complement distributions)**.** *The sensitivity and specificity of a test with respect to a disease can be given as follows:*

```
<Potential type="Table" role="ConditionalProbability" >
   <Variables>
      <Variable name="Result of test"/>
      <Variable name="Disease"/>
   </Variables>
```

[17] A joint probability that depends on only one variable can be understood as a conditional probability with no conditioning variables.

```
    <Values>0.97 0.03 0.05 0.95</Values>
    <UncertainValues>
      <Value distribution="Beta" name="specificity">97 3</Value>
      <Value distribution="Complement">1</Value>
      <Value distribution="Complement">1</Value>
      <Value distribution="Beta" name="sensitivity">95 5</Value>
    </UncertainValues>
  </Potential>
```

Example 18 (Dirichlet distribution)**.** *Given that the Beta distribution is a particular case of the Dirichlet, the potential in Example 17 can also be encoded as follows:*

```
  <Potential type="Table" role="ConditionalProbability" >
    <Variables>
      <Variable name="Result of test"/>
      <Variable name="Disease"/>
    </Variables>
    <Values>0.97 0.03 0.05 0.95</Values>
    <UncertainValues>
      <Value distribution="Dirichlet" name="specificity">97</Value>
      <Value distribution="Dirichlet">3</Value>
      <Value distribution="Dirichlet">5</Value>
      <Value distribution="Dirichlet" name="sensitivity">95</Value>
    </UncertainValues>
  </Potential>
```

This potential encodes two Dirichlet distributions: the first one, with parameters 97 and 3, is used to sample the values of $P(\neg y|\neg x)$ *and* $P(+y|\neg x)$*; the second, with parameters 5 and 95, is used to sample the values of* $P(\neg y|+x)$ *and* $P(+y|+x)$*.*

Please note that, according with the above three rules, it is possible that all the distributions be of type *Complement*. In this case there would be no uncertainty, but this specification might be used to give unnormalized probability values, from which the normalized **Values** would be computed.

4.4.3. Delta

This potential is used to assign a probability distribution to a variable whose value is known with certainty, such that the probability is 0 for all the values except one. Its skeleton is very simple:

```
  <Potential type="Delta" role=enumPotentialRole>
    <Variable name=string/>1
    <State>string</State>0..1
    <StateIndex>non-negative_integer</StateIndex>0..1
    <NumericValue>number</NumericValue>0..1
  </Potential>
```

where role is either jointProbability or ConditionalProbability, and exactly one of the tags **`State`**, **`StateIndex`**, and **`NumericValue`**, must be present in each case. The tags **`State`** and **`StateIndex`** are used for finite-states and discretized variables to denote the state whose probability is 1; the probability for the other states is 0. In this case, the potential represents a Kronecker delta. The tag **`NumericValue`** is used for numerical variables to denote the value on which all the probability density concentrates. In this case, the potential represents a Dirac delta.

This type of potential might be used to assign a prior probability to a variable without parents, but in general it does not make sense to have a "variable" whose value is always the same. The typical use of this type of potential is inside a tree, to indicate, for instance, that if variable X takes on the value x then variable Y takes on the value y—see Example 32 in Section 5.1.3.b.

4.4.4. Tree and ADD

a) Trees

There may be several advantages of using trees instead of tables when the potential has a substructure that repeats itself several times. First, it would be boring and time-consuming to introduce the same parameters again and again in a table. Second, it is easier to maintain a model in which each parameter appears only once. Third, in sensitivity analysis, having a parameter that appears several times in a table is different from having several parameters, even if all of them have the same probability distribution; the difference becomes apparent when sampling the potential. Fourth, a tree may reduce the storage space, both in disk and in working memory. And fifth, inference can be more efficient with trees than with tables [12].

The skeleton of a tree (which is a particular case of an ADD, as we will see below) is very simple:

```
<Potential type="Tree/ADD" role=enumPotentialRole >
   <UtilityVariable name=string />0..1
   <Variables>
      <Variable name=string />1..n
   </Variables>0..1
   <TopVariable name=string />
   <Branches>
      <Branch />1..n
   </Branches>
</Potential>
```

The tag **`Variables`** specifies the set of variables on which the potential is defined; the **`Variables`** must be declared for a top-level tree, but it is not necessary to declare them for embedded trees, as we will see in the examples below. If role="Utility", the **`UtilityVariable`** must be specified, but this variable cannot appear inside the tree. The other **`Variables`** may appear explicitly inside the tree/ADD or not.

The potential must have at least one variable—in addition to the **UtilityVariable** and the conditioned variable—that will be used as the **TopVariable**. If this variable is finite-states or discretized, the skeleton for its branches is:

```
<Branch>
  <States>
    <State name=string />0..n
  </States>
  <Potential />
</Branch>
```

Each state of the **TopVariable** must be assigned to exactly one of the branches of the tree, but a branch may have several states assigned.

If the **TopVariable** is numerical, the skeleton for its branches is:

```
<Branch>
  <Thresholds>
    <Threshold value=number belongsTo=enumSide />2
  </Thresholds>
  <Potential />
</Branch>
```

The thresholds of the branches must cover the domain of the variable (Sec. 4.2.2), without overlapping, in the right order; i.e., the first threshold of the first branch must be the same as the first of the thresholds that define the domain of the variable; the second threshold of the last branch must be the same as the second of the thresholds that define the domain of the variable; and the second threshold of the i-th interval must be the same as the first threshold of the $(i+1)$-th interval.[18]

Using thresholds for a numerical variable that appears within a tree can be interpreted as a discretization of that variable. The difference with a *discretized variable* is that the latter is discretized when defining it (see Sec. 4.2.3), once and forever, while the "discretization" of a *numeric variable* is done within a tree, and therefore may different for each tree involving that variable.

The leaves of a tree are always non-tree potentials, as shown in the following examples.

Example 19. *The utility potential $U(a,b)$ defined by Table 1 (page 25), which is equivalent to the tree in Figure 5, can be represented by a combination of tree and table potentials:*

```
<Potential type="Tree/ADD" role="Utility" >
  <UtilityVariable name="U"/>
```

[18] This ordering of the intervals is required only for numerical variables. In the case of discretized variables, each interval is associated to a state, and the states/intervals can be associated to the branches in any order; it is also possible to have non-consecutive states/intervals in the same branch.

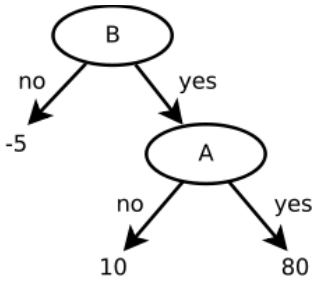

Figure 5: A tree for the potential $U(a,b)$.

```
<Variables>
  <Variable name="A"/>
  <Variable name="B"/>
</Variables>
<TopVariable name="B"/>
<Branches>
  <Branch>
    <States>
      <State name="no"/>
    </States>
    <Potential type="Table">
      <Values>–5</Values>
    </Potential>
  </Branch>
  <Branch>
    <States>
      <State name="yes"/>
    </States>
    <Potential type="Table">
      <Variables>
        <Variable name="A"/>
      </Variables>
      <Values>10 80</Values>
    </Potential>
  </Branch>
</Branches>
</Potential>
```

In this example, the value -5 has been represented as a table that does not depend on any variable. It may look cumbersome to represent a constant as a table, but it has the advantage that this way we can express uncertainty about this value by using the **`UncertainValues`** tag of the table, as explained in Section 4.4.2.b. With respect to the node A, we might have used a subtree with two branches instead of a table, as in Figure 5, but it would occupy more space.

Example 20. *The conditional probability potential $P(c|a,b)$ shown in Table 2 (page 26) can be represented as follows:*

```
<Potential type="Tree/ADD" role="ConditionalProbability" >
  <Variables>
    <Variable name="C"/>
    <Variable name="A"/>
    <Variable name="B"/>
  </Variables>
  <TopVariable name="A"/>
  <Branches>
    <Branch>
      <States>
        <State name="no"/>
      </States>
      <Potential type="Table">
        <Variables>
          <Variable name="C"/>
        </Variables>
        <Values>0.9 0.1</Values>
      </Potential>
    </Branch>
    <Branch>
      <States>
        <State name="yes"/>
      </States>
      <Potential type="Table">
        <Variables>
          <Variable name="C"/>
          <Variable name="B"/>
        </Variables>
        <Values>0.7 0.3 0.2 0.8</Values>
      </Potential>
    </Branch>
  </Branches>
</Potential>
```

In this example, the first lines have declared that the potential represents the conditional probability of C given A and B, therefore each leave in the tree must represent a conditional probability for C.

The format ProbModelXML allows each variable to appear at several levels of the tree, provided that the intervals and the sets of states of the branches are coherent, as shown in the following example.

Example 21. *The potential in Table 4 can be represented by the tree in Figure 6. The root node A has a branch whose associated interval is* $(0,1]$. *The inner node A has two branches; their associated intervals,* $(0,0.5)$ *and* $[0.5,1]$, *do not overlap and their union is* $(0,1]$. *Similarly, the first node B has a branch for the set of states* $\{b_1,b_2\}$. *The second node B has two branches; their associated sets of states,* $\{b_1\}$ *and* $\{b_2\}$, *and their union is* $\{b_1,b_2\}$. *Therefore the intervals and the sets of states for the branches of the tree are coherent.*

$U(a,b)$	$a=0$	$0<a<0.5$	$0.5\leq a<1$
b_2	0	10	30
b_1	0	10	20
b_0	0	0	0

Table 4: This table can be represented by the tree in Figure 6.

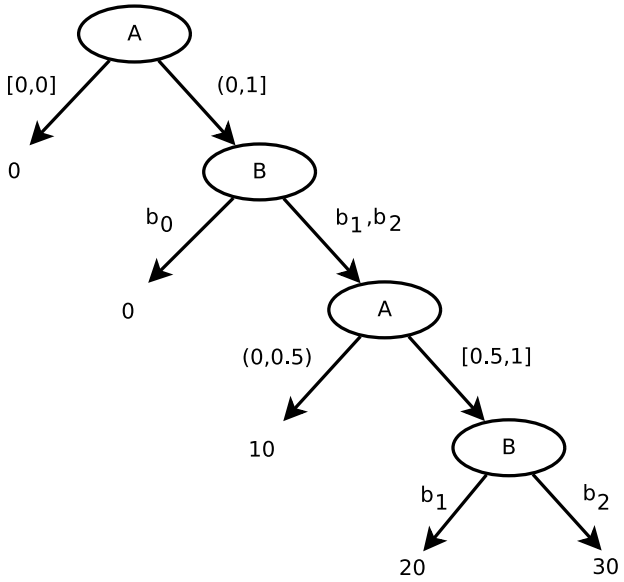

Figure 6: A tree for the potential $U(a,b)$ defined in Table 4.

b) Algebraic decision diagrams

An algebraic decision diagram (ADD) is very similar to a tree, as it is also an acyclic directed graph (ADG). The only difference is that ADDs may contain loops, while trees do not. Therefore, trees are a particular type of ADDs in which no node has more than one parent. When a node in an ADD has more than one parents, the corresponding branch is assigned a label that can be later referenced in another branch.[19]

[19]It might be more intuitive to assign the label to the potential, but in that case we should define an attribute for potentials that would be used only inside tree/ADDs. Therefore, it is more coherent to define labels as a tag

Put formally, in an ADD, if the **TopVariable** is finite-states or discretized, the skeleton for its branches is:

```
<Branch>
   <States>
      <State name=string/>0..n
   </States>
   <Potential />0..1
   <Label>string</Label>0..1
   <Reference>string</Reference>0..1
</Branch>
```

where either **Potential** or **Reference** must be present, and the latter is incompatible with both **Potential** and **Label**. In the case of a numerical **TopVariable**, the skeleton is very similar:

```
<Branch>
   <Thresholds>
      <Threshold value=number belongsTo=enumSide/>2
   </Thresholds>
   <Potential />0..1
   <Label>string</Label>0..1
   <Reference>string</Reference>0..1
</Branch>
```

Therefore, ADDs include trees as a particular case in which it is not necessary to use labels nor references.

Example 22. *The ADD in Figure 7 can be represented by the following code, where the potential –5 has been labeled as "negative utility" in the branch B="no", and there is a reference to that potential in the branch A="no":*

```
<Potential type="Tree" role="Utility" >
   <UtilityVariable name="U" />
   <Variables>
      <Variable name="A"/>
      <Variable name="B"/>
   </Variables>
   <TopVariable name="B"/>
   <Branches>
      <Branch>
```

of branches. This way the tags and attributes of potentials do not depend on whether they are inside or outside a tree/ADD.

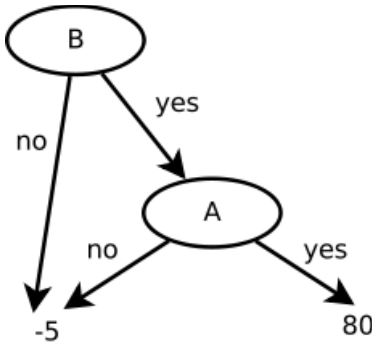

Figure 7: An ADD containing a loop.

```
   <States>
      <State name="no"/>
   </States>
   <Potential type="Table">
      <Values>–5</Values>
   </Potential>
   <Label>negative utility</Label>
</Branch>
<Branch>
   <States>
      <State name="yes"/>
   </States>
   <Potential type="Tree">
      <TopVariable name="A"/>
      <Branches>
         <Branch>
            <States>
               <State name="no"/>
            </States>
            <Reference>negative utility</Reference>
         </Branch>
         <Branch>
            <States>
               <State name="yes"/>
            </States>
            <Potential type="Table">
               <Values>80</Values>
            </Potential>
         </Branch>
      </Branches>
```

```
        </Potential>
      </Branch>
    </Branches>
  </Potential>
```

4.4.5. ICI model

ICI potentials [14] were designed to represent conditional probabilities, such as $P(y|x_1,\ldots,x_n)$, but they can also be used to represent a policy imposed by the user [26]. The term ICI stands for *independence of causal influence*. An ICI potential is composed of several subpotentials, one for each link $X_i \rightarrow Y$. The skeleton of an ICI potential is:

```
<Potential type="ICIModel" role=enumRoleType >
  <Model>enumICIModel</Model>
  <Variables>
    <Variable type=enumVariableType />2..n
  </Variables>
  <Subpotentials>
    <Potential type=enumSubpotentialType />1..n
  </Subpotentials>
</Potential>
```

In this context, *enumRoleType* can only be ConditionalProbability or Policy. In turn, *enumICIModel* can take one of this values: {Or, CausalMax, GeneralizedMax, And, CausalMin, GeneralizedMin, Tuning}; in the future we might include in ProbModelXML other types of ICI models. In the rest of this paper, *enumSubpotentialType* is either Table, Tree/ADD, Delta, or Uniform.

Each subpotential is a potential of two variables: $X_i \rightarrow Y$, except the last subpotential, which represents the leak probability [14] and depends only on Y.

Example 23. *A noisy Or potential for a node Y with two parents, $P(y|x_1,x_2)$, can be defined as a composition of three subpotentials, $P(y|x_1)$, $P(y|x_2)$, and $P(y)$—see [14]:*

```
<Potential type="ICIModel" role="ConditionalProbability" >
  <Model>Or</Model>
  <Variables>
    <Variable name="Y"/>
    <Variable name="X1"/>
    <Variable name="X2"/>
  </Variables>
  <Subpotentials>
    <Potential type="Table" >
      <Variables>
        <Variable name="Y"/>
        <Variable name="X1"/>
```

```
        </Variables>
        <Values>0.3 0.7 0.8 0.2</Values>
      </Potential>
      <Potential type="Table" >
        <Variables>
          <Variable name="Y"/>
          <Variable name="X2"/>
        </Variables>
        <Values>0.9 0.1 0.2 0.8</Values>
      </Potential>
      <Potential type="Table" >
        <Variables>
          <Variable name="Y"/>
        </Variables>
        <Values>0.999 0.001</Values>
      </Potential>
    </Subpotentials>
</Potential>
```

4.4.6. Sum and product

Each potential of type Sum involves $m+1$ numeric variables $\{U, U_1, \ldots, U_m\}$, where U is usually a supervalue node[20] and each U_i is another utility node and a parent of U. If the parents of U_i are $\mathbf{X}_i$, and $\mathbf{X} = \cap_i \mathbf{X}_i$, then

$$U(\mathbf{x}) = \sum_{i=1}^{m} U_i(\mathbf{x}_i) \,. \tag{1}$$

We require that $m \geq 2$ because we must sum at least two utilities. (If m were 1, then $U(\mathbf{x}) = U_1(\mathbf{x}_1)$ and node U would be useless.)

The skeleton of the Sum potential is very simple because it does not involve any numeric parameters:

```
<Potential type="Sum" role="Utility" >
  <UtilityVariable name=string />
  <Variables>
    <Variable name=string>2..n
  </Variables>
</Potential>
```

The definition of the Product potential is analogous.

[20] In the context of influence diagrams, a utility node is said to be *supervalue* if its parents are other utility nodes [41].

4.4.7. Linear combination

This type of potential is used when a *numeric* variable Y depends *deterministically* on a mixed set of variables $\mathbf{X} = \mathbf{D} \cup \mathbf{C}$ (usually these variables will be the parents of node Y in the graph that represents the probabilistic model), where $\mathbf{C} = \{C_1, \ldots, C_m\}$ is a set of m numeric variables and $\mathbf{D} = \{D_1, \ldots, D_n\}$ is a set of n finite-states variables, such that Y is a linear combination of the numeric variables (c_i) with coefficients (β_i) that depend on the finite-states or discretized variables:[21]

$$y = \alpha(\mathbf{d}) + \sum_{i=1}^{m} \beta_i(\mathbf{d}) \cdot c_i \,. \tag{2}$$

The potential Sum is a particular case of LinearCombination in which $n = 0$, $\alpha = 0$, and $\beta_i = 1$ for all i.

The specification of this model is similar to that of an ICI model, in the sense that each subpotential $\beta_i(\mathbf{d})$ is associated with a link, $C_i \to Y$, and the potential $\alpha(\mathbf{d})$ is similar to the leak probability, as it is not associated with any link. As each subpotential depends on a set of finite-states variables, it can be expressed as a table or a tree (or even by an ICI model, but we think that it does not make sense to apply ICI models in this context). The skeleton would be as follows:

```
<Potential type="LinearCombination" role="Utility">
   <UtilityVariable name=string />
   <Variables />1..n
   <Subpotentials>
      <Subpotential type=enumSubpotentialType />2..n
   </Subpotentials>
</Potential>
```

We have restricted the role to Utility because we think that it does not make sense to use this kind of potential to define probability distributions; however, we might relax this restriction in the future. The tag **Variables** encloses first the m numeric variables ($m \geq 1$) and then the n finite-states variables ($n \geq 0$). The tag **Subpotentials** encloses the $m+1$ subpotentials, $\{\alpha(\mathbf{d}), \beta_1(\mathbf{d}), \ldots, \beta_m(\mathbf{d})\}$.[22] As mentioned above, *enumSubpotentialType* is either Table, Tree/ADD, Delta, or Uniform. Each subpotential depends on the finite-states variables $\mathbf{D}$ (or on a subset of them).

Example 24. *In cost-effectiveness analysis, the net health benefit of an intervention [40] is given by*

$$NHB = E - \lambda \cdot C \,, \tag{3}$$

[21] This type of interaction was proposed in [34, sec. 7.2] for networks consisting only of numeric variables. It is also used in DANs [15] when Y is a supervalue node and the set of parents of Y ($\mathbf{X} = Pa(Y)$) consists of some utility nodes—which represent numeric variables, by definition—and some chance and decision nodes representing finite-states variables.

[22] We have assumed that $m \geq 1$ because if there were no numeric variables, then the only subpotential would be $\alpha(\mathbf{d})$, and it would be much more simple to use directly a Table or a Tree/ADD instead of a LinearCombination.

where E is the effectiveness, C is the economic cost, and λ is a parameter that transforms the effectiveness into a monetary scale; clearly, it depends on the wealth of the country. Assuming that the variable Country *takes on two values,* developingCountry *and* developedCountry*, we may have* λ*(*developingCountry*) = 20,000 and* λ*(*developedCountry*) = 30,000. Then, the above equation may lead to the following potential:*

```
<Potential type="LinearCombination" role="Utility">
  <UtilityVariable name="Net health benefit" />
  <Variables>
    <Variable name="Effectiveness" />
    <Variable name="Cost" />
  </Variables>
  <Subpotentials>
    <Potential type="Table">
      <Values>1</Values>
    </Potential>
    <Potential type="Table">
      <Variables>
        <Variable name="Country">
      </Variables>
      <Values>–20000 –30000</Values>
    </Potential>
  </Subpotentials>
</Potential>
```

4.4.8. Logistic regression

A logistic regression potential is used to represent the conditional probability of a boolean variable Y given a set of n conditioning variables $\mathbf{X} = \mathbf{D} \cup \mathbf{C}$. A variable is said to be boolean when its domain is {false, true}, {no, yes}, {absent, present}, or {negative, positive}. We will denote the domain of Y by $\{\neg y, +y\}$. The probability of Y is

$$P(+y|\mathbf{x}) = \left[1 + \exp\left(-\beta_0 - \sum_{i=1}^{n} \beta_i \cdot index(d_i) - \sum_{j=1}^{m} \beta_j \cdot (c_j - b_j)\right)\right]^{-1}, \qquad (4)$$

where n is the number of finite-states variables and m is the number of numerical variables. Each finite-states variable D_i is treated, in the terminology of statistics, as an ordinal categorical variable. In this expression $index(d_i)$ is the index of the value d_i in the domain of D_i. If D_i is a boolean variable, $index(+d_i) = 1$ and $index(\neg d_i) = 0$; therefore, the contribution of $+d_i$ to the sum inside the parenthesis is β_i and that of $\neg d_i$ is 0. Similarly, c_j is the value of the numerical variable C_i, and b_j is the baseline value for variable C_i. Usually, $b_j = 0$.

The skeleton of a potential of this type is:

```
<Potential type="LogisticRegression" role="ConditionalProbability">
  <Variables>
    <Variable name=string base=number>1..n
  </Variables>
  <Coefficients>numbers</Coefficients>
  <CovarianceMatrix>numbers</CovarianceMatrix>0..1
</Potential>
```

Again, the first of the **Variables** is the conditioned variable, Y, and the others are the conditioning ones, $\mathbf{X}$. The coefficient of each variable denotes its β. The attribute base, if present, denotes b_j.

The optional tag **CovarianceMatrix** means that there is a second order probability for the β parameters: a multivariate normal distribution, whose mean vector, $\boldsymbol{\mu}$, is given by the **Coefficients** and whose covariance matrix is

$$\boldsymbol{\Sigma} = \begin{pmatrix} \sigma_{00} & \sigma_{01} & \cdots & \sigma_{0n} \\ \sigma_{10} & \sigma_{11} & \cdots & \sigma_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{n0} & \sigma_{n1} & \cdots & \sigma_{nn} \end{pmatrix} \tag{5}$$

As this matrix is symmetric, we only need to specify $(n+1)\cdot n/2$ of its values, as follows:

```
<CovarianceMatrix>σ00 σ10 σ11 σ20 σ21 σ22 σ30 ... σnn
</CovarianceMatrix>
```

Example 25. *The prevalence of a certain disease Y depends on the patient's sex (X_1) and age (X_2). The domain of X_1 is {female, male} and the age, X_2, is measured in years, about 50. A regression analysis has yielded the following results:*[23]

$$\beta_0 = -4.2 \qquad \beta_1 = 0.78 \qquad \beta_2 = 0.035$$
$$\boldsymbol{\Sigma} = \begin{pmatrix} 0.13 & 0.001 & -0.0003. \\ 0.001 & 0.52 & -0.00006 \\ -0.0003 & -0.00006 & 0.0012 \end{pmatrix}$$

This model can be represented as follows:

```
<Potential type="LogisticRegression" role="ConditionalProbability">
  <Variables>
    <Variable name="Disease Y">
    <Variable name="Sex">
    <Variable name="Age" base="50">
```

[23] The fact that both β_0 and β_1 are positive means that age and being male are risk factors for this disease. The probability that an 80 year old woman has the disease is $[1+\exp(4.2-0.78\cdot 0-0.035\cdot(80-50))]^{-1} = 0.042$.

```
    </Variables>
    <Coefficients>–4.2 0.78 0.035</Coefficients>
    <CovarianceMatrix>0.13 0.001 0.52 –0.0003 –0.00006 0.0012
    </CovarianceMatrix>
  </Potential>
```

4.4.9. Conditional Gaussian

As in the previous case, this type of potential is used when a *numeric* variable Y depends on a mixed set of variables $\mathbf{X} = \mathbf{D} \cup \mathbf{C}$, which in general will be the parents of Y. The main difference is that in this case Y depends *probabilistically* on $\mathbf{X}$, while a linear combination entails deterministic dependence. More specifically, the conditional probability density for Y is a univariate normal distribution:[24]

$$f(y|\mathbf{d},\mathbf{c}) \sim \mathcal{N}(\mu(\mathbf{d},\mathbf{c}),\sigma^2(\mathbf{d}))\,, \tag{6}$$

where

$$\mu(\mathbf{d},\mathbf{c}) = \alpha(\mathbf{d}) + \sum_{i=1}^{m} \beta_i(\mathbf{d}) \cdot c_i\,. \tag{7}$$

The skeleton of a conditional Gaussian is:

```
<Potential type="ConditionalGaussian" role="ConditionalProbability">
  <Variables />1..n
  <Subpotentials>
    <Subpotential type=enumSubpotentialType />3..n
  </Subpotentials>
</Potential>
```

The tag **Variables** must enclose first the conditioned variable Y, then the m numeric variables in $\mathbf{C}$ ($m \geq 0$) and finally the n finite-states variables ($n \geq 0$). The $m+2$ subpotentials are $\{\alpha(\mathbf{d}), \beta_1(\mathbf{d}), \ldots, \beta_m(\mathbf{d}), \sigma^2(\mathbf{d})\}$.

Clearly, the specification of a ConditionalGaussian potential is similar to that of a LinearCombination (cf. Sec. 4.4.7). The main difference is that linear combinations represent utility potentials, while conditional Gaussians represent conditional probabilities. A minor difference is that a conditional Gaussian contains $m+2$ subpotentials instead of $m+1$, because there is one extra potential for representing $\sigma^2(\mathbf{d})$, the variance of Y.

[24] Conditional Gaussian potentials were proposed in [29]. Later, Lauritzen and Frank Jensen [27] used conditional Gaussians of the form $f(\mathbf{y}|\mathbf{d},\mathbf{c})$—a particular case of those in [29]—to build mixed Bayesian networks with the restriction that finite-states nodes could not have numeric parents. The specification proposed here is even more restrictive, in the sense that it assumes that there is only one conditioned variable Y, instead of a set $\mathbf{Y}$. It would be easy to extend the ProbModelXML format to represent more general conditional Gaussian potentials, but currently we do not see the need for it.

4.4.10. Exponential and mixture of exponentials

a) Exponential potentials

Given a mixed set of variables, $\mathbf{X} = \mathbf{D} \cup \mathbf{C}$, as in the above sections, an exponential potential $\phi(\mathbf{x})$ is defined by a subpotential $\phi'(\mathbf{d})$ and a vector of m parameters (one for each numeric variable), such that

$$\phi(\mathbf{x}) = \phi(\mathbf{d}, \mathbf{c}) = \phi'(\mathbf{d}) \cdot \exp\left(\sum_{i=1}^{m} \beta_i \cdot c_i\right) . \tag{8}$$

Accordingly, the skeleton of this type of potential is:

```
<Potential type="Exponential" role=enumPotentialRole >
   <Potential type=enumSubpotentialType />
   <NumericVariables />0..1
   <Coefficients />0..1
</Potential>
```

The inner potential represents $\phi'(\mathbf{d})$, which is defined on the n finite-states variables. The tag **`NumericVariables`** encloses the m numeric variables ($m \geq 1$). The number of **`Coefficients`** must be m.

b) Mixtures of exponentials

A MixtureOfExponentials potential is given by the sum of several exponential potentials:

$$\phi(\mathbf{x}) = \sum_{r} \phi_r(\mathbf{x}) . \tag{9}$$

Its skeleton is very simple:

```
<Potential type="MixtureOfExponentials" role=enumPotentialRole >
   <Variables>
   <Subpotentials>
      <Potential type="Exponential" />1..n
   </Subpotentials>
</Potential>
```

For coherence with the rules for other types of potentials, the tag **`Variables`** must enclose first the numeric variables and then the finite-states variables.

c) Mixtures of truncated exponentials

Given $\mathbf{X} = \mathbf{D} \cup \mathbf{C}$, where $\mathbf{D}$ contains n finite-states variables and $\mathbf{C}$ contains m numeric variables, a mixture of truncated exponentials (MTE potential) is defined by partitioning $\mathbb{R}^m$ into a set of

hypercubes and assigning a mixture of exponentials to each one.[25] When an MTE represents the joint probability for the variables of $\mathbf{X}$ or the conditional probability of one variable in $\mathbf{X}$ (either numeric or finite-states) given the others, it must be always non-negative; in principle, it should be normalized, but it is also possible to work with unnormalized potentials, under the assumption that there is an implicit normalization constant that is not relevant.

In ProbModelXML, a mixture of truncated exponentials can be encoded as a tree whose leaf nodes are all MixtureOfExponential potentials.

Example 26. *The MTE in Figure 8, which represents a utility function of two finite-states variables and two numeric variables, $U(d_0,d_1,c_0,c_1)$, can be encoded as follows. Please note that the sub-tree for D1 in Figure 8 has not been encoded in ProbModelXML as a tree containing two exponentials, but as a single exponential, $\phi'(d_1)\cdot\exp(0.25\,c_1)$, where $\phi'(d_1)$ is a Table potential such that $\phi'(\text{D1=no}) = 10$ and $\phi'(\text{D1=yes}) = 5$.*

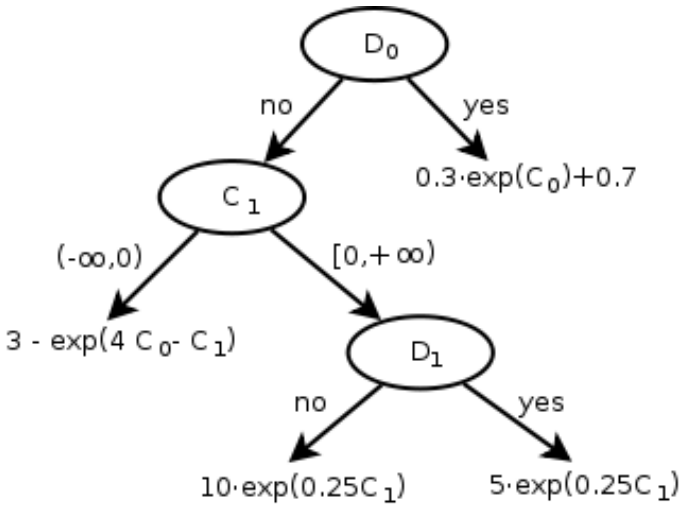

Figure 8: A mixture of truncated exponentials.

[25]The definition of MTE proposed in this section is somewhat more general than that proposed by Moral et al. [31], because in their definition each exponential has the form

$$\phi(\mathbf{x}) = \phi(\mathbf{d},\mathbf{c}) = \underbrace{a\cdot\exp\left(\sum_{j=1}^{n} b_j\cdot d_j\right)}_{\phi'(\mathbf{d})}\cdot\exp\left(\sum_{i=1}^{m}\beta_i\cdot c_i\phi'(\right) \tag{10}$$

while in ours there are more degrees of freedom in the choice of $\phi'(\mathbf{d})$. The constant term a_0 in their definition may correspond to an exponential in which all the β_i's are zero. Their definition does not only partition the joint domain of the numeric variables, $\mathbb{R}^m$, but also the joint domain of the finite-states variables; in our definition that partitioning can be represented by means a tree, as explained below.

In this sense, our proposal is closer to that of Koller et al. [25], in which the probability of a finite-states variable conditioned on a set of numeric parents is given by a mixture of expontials, each one being similar to that in Equation 8.

```
<Potential type="Tree/ADD" role="Utility" >
  <UtilityVariable name="U1" >
  <Variables>
    <Variable name="D0" />
    <Variable name="D1" />
    <Variable name="C0" />
    <Variable name="C1" />
  </Variables>
  <TopVariable name="D0" />
  <Branches>
    <Branch>
      <States>
        <State name="no" >
      </States>
      <Potential type="Tree/ADD" >
        <TopVariable name="C1" />
        <Branches>
          <Branch>
            <Thresholds>
              <Threshold value="–Infinity" >
              <Threshold value="0" belongsTo="Left" >
            </Thresholds>
            <Potential type="MixtureOfExponentials" >
              <Variables>
                <Variable name="C0"/>
                <Variable name="C1"/>
              </Variables>
              <Subpotentials>
                <Potential type="Exponential" />
                  <Potential type="Table">
                    <Values>3</Values>
                  </Potential>
                </Potential>
                <Potential type="Exponential" />
                  <Potential type="Table">
                    <Values>–1</Values>
                  </Potential>
                  <NumericVariables>
                    <Variable name="C0"/>
                    <Variable name="C1"/>
                  </NumericVariables>
                  <Coefficients>4  –1</Coefficients>
                </Potential>
```

```
            <Subpotentials>
          </Potential>
        </Branch>
        <Branch>
          <Thresholds>
            <Threshold value="0" belongsTo="Left">
            <Threshold value="+Infinity" >
          </Thresholds>
          <Potential type="MixtureOfExponentials">
            <Variables>
              <Variable name="C1"/>
              <Variable name="D1"/>
            </Variables>
            <Subpotentials>
              <Potential type="Exponential" />
                <Potential type="Table" >
                  <Variables>
                    <Variable name="D1" />
                  </Variables>
                  <Values>10 5</Values>
                </Potential>
                <NumericVariables>
                  <Variable name="C1"/>
                </NumericVariables>
                <Coefficients>0.25</Coefficients>
              </Potential>
            </Subpotentials>
          </Potential>
        </Branch>
      </Branches>
    </Potential>
  </Branch>
  <Branch>
    <States>
      <State name="yes">
    </States>
    <Potential type="MixtureOfExponentials">
      <Variables>
        <Variable name="C0"/>
      </Variables>
      <Subpotentials>
        <Potential type="Exponential" >
          <Potential type="Table">
```

```
            <Values>0.3</Values>
          </Potential>
          <NumericVariables>
            <Variable name="C0"/>
          </NumericVariables>
          <Coefficients>1</Coefficients>
        </Potential>
        <Potential type="Exponential" >
          <Potential type="Table">
            <Values>0.7</Values>
          </Potential>
        </Potential>
      <Subpotentials>
    </Potential>
  </Branch>
 </Branches>
</Potential>
```

5. Special types of networks

5.1. Dynamic models

The representation of dynamic models in their compact form (see Section 2.4) is not very different from that of other types of models. In this section we explain only those aspects in which they differ.

5.1.1. Network tags

In addition to the tags defined in Section 4, our format offers the following tags for a network that represents a dynamic model:

```
<TimeUnit>string</TimeUnit>0..1
<CycleLength>number</CycleLength>0..1
<CoordinatesShift>number number</CoordinatesShift>0..1
```

The meaning of **TimeUnit** and **CycleLength** is obvious. The default value for **TimeUnit** is the empty string. The default value for **CycleLength** is 1, i.e., one time unit.

Example 27. *The specification*

```
<TimeUnit>year</TimeUnit>
<CycleLength>0.5</CycleLength>
```

is equivalent to

```
<TimeUnit>month</TimeUnit>
<CycleLength>6</CycleLength>
```

The tag **<CoordinatesShift>** is used to maintain the relative positions of variables in different time slices.

Example 28. *The tag* **<CoordinatesShift>**400 10**</CoordinatesShift>** *means that if the node "X [t]" has coordinates* $(80,60)$*, then "X [*$t+1$*]" has coordinates* $(480,70)$*, unless there is a tag in the declaration of "X [*$t+1$*]" that overrides this value.*

5.1.2. Dynamic variables

Temporal variables are recognized because they have an attribute indicating the time slice that contains the variable:[26]

<Variable name=*string* timeSlice=*non-negative_integer* ... **>**

When the timeSlice is greater than 0 (for example, **<Variable** name="Disease" timeSlice="1" **>**), it is not necessary to specify the domain of the variable, because it is necessarily the same as that of the corresponding variable at the first time slice (in our example, **<Variable** name="Disease" timeSlice="0" **>**).

The attribute timeSlice is also used when denoting variables in links and potentials.

Example 29. *The decision about therapy in the current time slice affects the probability of the disease in the next one. This can be encoded as follows:*

```
<Link directed="true ">
   <Variable name="Dec:Therapy" timeSlice="0" />
   <Variable name="Disease" timeSlice="1" />
</Link>
```

5.1.3. Potentials for dynamic models

Temporal models can use all the potentials defined in Section 4.4, as well as the following potentials defined specifically for temporal relations.

a) Same as previous

The SameAsPrevious type means that a potential defined on a set of variables is of the same type and has the same **Values** as the potential defined on the corresponding variables of the previous time slice. Its skeleton is:

[26] In OpenMarkov, the time slice of each variable is indicated in square brackets. For example, the variable **<Variable** name="Disease" timeSlice="0"**>** is depicted as "Disease [0]" in Figure 2.

```
<Potential type="SameAsPrevious" role=enumPotentialRole >
   <Variables>
      <Variable name=string timeSlice=non-negative_integer>1..n
   </Variables>
</Potential>
```

Example 30. *The following declaration means that the conditional probability $P(Symptom\ [1]\,|\,Disease\ [1])$ for the model in Figure 2 is the same as $P(Symptom\ [0]\,|\,Disease\ [0])$:*

```
<Potential type="SameAsPrevious" role="ConditionalProbability">
   <Variables>
      <Variable name="Symptom" timeSlice="1">
      <Variable name="Disease" timeSlice="1">
   </Variables>
</Potential>
```

Please note that in the concise model depicted in Figure 3 we do not need a **SameAsPrevious** potential for "Symptom [1]" because this variable does not appear explicitly, but there are other cases in which this type of potential is necessary, or at least very useful. Also note that the variables in a potential of this type may belong to different time slices.

b) Cycle length shift

This type of potential is used to increase the value of a variable that represents age, in a broad sense, such as the age of a person or the "age" of a prosthesis (the time elapsed since it was implanted).

```
<Potential type="CycleLengthShift">
   <Variables>
      <Variable name=string timeSlice=non-negative_integer>2
   </Variables>
</Potential>
```

The two **Variables** must be temporal variables, sharing the same name, and the time index for the second must be one unit greater than that of the first. This type of potential does not require any numeric argument.

Example 31. *The value of variable "Age [t]" is that of "Age [$t-1$]" plus the cycle length:*

```
<Potential type="CycleLengthShift">
   <Variables>
      <Variable name="Age" timeSlice="0">
      <Variable name="Age" timeSlice="1">
   </Variables>
</Potential>
```

Example 32. *The following tree indicates that the age of a prosthesis increases with the passing of time (first branch), but if the prosthesis is replaced, its age is reset to 0 (second branch):*

```
<Potential type="Tree/ADD" role="ConditionalProbability"/>
  <Variables>
    <Variable name="Prosthesis age" timeSlice="1"/>
    <Variable name="Prosthesis age" timeSlice="0"/>
    <Variable name="Prosthesis replacement" timeSlice="1"/>
  </Variables>
  <TopVariable name="Prosthesis replacement" timeSlice="1"/>
  <Branches>
    <Branch>
      <States>
        <State name="no"/>
      </States>
      <Potential type="CycleLengthShift"/>
        <Variable name="Prosthesis age" timeSlice="0">
        <Variable name="Prosthesis age" timeSlice="1">
      </Potential>
    </Branch>
    <Branch>
      <States>
        <State name="yes"/>
      </States>
      <Potential type="Delta">
        <Variable name="Prosthesis age" timeSlice="1">
        <NumericValue>0</NumericValue>
      </Potential>
    </Branch>
  </Branches>
</Potential>
```

c) Weibull distribution

The Weibull distribution [44, 45] is used mainly in survival analysis. Let variable "$Y\ [t]$" represent the occurrence of a hazardous event between time $t-1$ and t. The probability of this event may depend on a set of variables $\mathbf{X}$, which represent the parents of "$Y\ [t]$" in the network: $\mathbf{X} = Pa(Y\ [t])$. For example, the probability of death may depend on the sex and age of the patient and on the presence of some diseases. Similarly, the probability of a prosthesis failure may depend on its age (how long ago it was implanted), the model (some prostheses have better quality than others), etc. [11].

If we assume that the probability that the hazardous event happens at time t—provided that it has not happened earlier and the configuration of the conditioning variables $\mathbf{X}$ at time t is $\mathbf{x}$—the

probability density function is:

$$f(t|\mathbf{x}) = \begin{cases} k\lambda(\mathbf{x})^{-k} t^{k-1} \exp(-\lambda(\mathbf{x})^{-k} t^k) & \text{if } t \geq 0 \\ 0 & \text{if } t < 0\,, \end{cases} \tag{11}$$

where the *shape parameter* k is a scalar, $k > 0$, and the so-called *scale parameter* λ depends on the variables in $\mathbf{X}$:

$$\lambda(\mathbf{x}) = \exp\left(\beta_0 + \sum_{i=1}^{n} \beta_i \cdot index(d_i) + \sum_{j=1}^{m} \beta_j \cdot (c_j - b_j)\right)\,. \tag{12}$$

The terms involved in this expression are the same as those in Equation 4. Therefore, the probability of the occurrence of a hazardous event between time $t - \delta$ and t is

$$P(\mathrm{Y[t]} = \mathrm{yes}|\mathbf{x}) = 1 - \exp\left\{[(t-\delta)^k - t^k]/\lambda(\mathbf{x})^{-k}\right\}\,, \tag{13}$$

where δ is the cycle length (see [11] for a derivation of this equation).

The exponential distribution is a particular case of the Weibull in which $k = 1$, which implies that

$$f(t|\mathbf{x}) = \begin{cases} \lambda(\mathbf{x})^{-1} \exp(-t/\lambda(\mathbf{x})) & \text{if } t \geq 0 \\ 0 & \text{if } t < 0\,, \end{cases} \tag{14}$$

and, consequently, the probability of a hazardous event is a constant:

$$P(\mathrm{Y[t]} = \mathrm{yes}|\mathbf{x}) = 1 - \exp\{-\delta/\lambda(\mathbf{x})\}\,. \tag{15}$$

The specification of this type of model in ProbModelXML can be done with the following skeleton, where the **Coefficients** are listed in the order $\{k, \beta_0, \beta_1, \ldots, \beta_n\}$:

```
<Potential type="WeibullDistribution" role="ConditionalProbability">
  <TimeVariable name=string >0..1
  <Variables>
    <Variable name=string base=number>1..n
  </Variables>
  <Coefficients>numbers</Coefficients>
  <CovarianceMatrix>numbers</CovarianceMatrix>0..1
</Potential>
```

The similarity with the logistic regression potential (cf. Sec. 4.4.8) is obvious.

The **TimeVariable** plays the role of t in Equations 11 and 13. For example, when modeling a recurrent disease, t may be the time elapsed since the previous episode. In this case, we may have

```
<TimeVariable name="Prosthesis age" timeSlice="1">
```

This variable must update its value at each time slice using the CycleLengthShift potential, as explained with an example in the previous section.

When no **TimeVariable** is specified, we assume that t is $i \cdot \delta$, where i is the index of the current time slice and δ is the cycle length.

The first of the **Variables** represents the occurrence of the event, i.e., "$Y[t]$", while the others represent the conditioning variables, $\mathbf{X}$ (cf. Eq. 13).

The tags **Coefficients** and **CovarianceMatrix** play the same role as in the case of a logistic regression (see Sec. 4.4.8), but then the order of the parameters was $\{\beta_0, \beta_1, \ldots, \beta_n\}$ and now it is $\{k, \alpha, \beta_1, \ldots, \beta_n\}$.

5.2. Decision analysis networks (DANs)

Given that influence diagrams are not appropriate for representing asymmetric decision problems, several alternative formalisms have been proposed in the last years [7, 15]. Decision analysis networks (DANs) [15], which are one of those types of models, can be encoded in ProbModelXML by using specific tags.

One of the features of DANs are restrictions. A restriction (x, y) associated to the link $X \to Y$ means that the values x and y are incompatible [15]. In ProbModelXML, the restrictions associated to a link can be represented by a potential ψ defined on $X \times Y$, such that $\psi(x, y) = 0$ means that there is a restriction (x, y).

Example 33. *Let us assume that a DAN contains a link $X \to Y$, where the domain of X is $\{\neg x, +x\}$ and that of Y is $\{\neg y, +y\}$. In the following fragment:*

```
<Link directed="true ">
   <Variable name="X" />
   <Variable name="Y" />
   <Potential type="Table" role="Restrictions" >
      <Variables>
         <Variable name="X"/>
         <Variable name="Y"/>
      </Variables>
      <Values>1 1 1 0</Values>
   </Potential>
</Link>
```

the value 0 denotes a restriction $(+x, +y)$, meaning that these two values are incompatible.[27]

[27] We might have chosen a more compact way of representing restrictions, instead of specifying the variables twice. However, potentials allow us to represent restrictions between all the types of variables (finite-states, discretized, numeric, and any pair of them) and several types of potentials (Table, Tree, Delta...) without introducing a new syntax for each particular case.

DANs have three specific ways to denote the flow of information [15]. The first one consists in declaring a chance variable is always observed by using the **AlwaysObserved** tag introduced in Section 4.2. This means that such variable can be observed without the need to perform any action.

Example 34. *In medicine symptoms are considered as "always observed" variables, because it is not necessary to make any test to detect them:*

```
<Variable name="Symptom" ...>
   <AlwaysObserved>
   ...
```

The second way of representing the flow of information is to use revelation arcs [15], which mean that certain values of a variable X reveal the value of variable Y and the evidence about Y can be used when making subsequent decisions. This property is represented by the tag **RevelationCondition** that may appear within the specification of the link $X \rightarrow Y$ (Sec. 4.3). If X is finite-states variable or discretized, then the syntax is:

```
<RevelationConditions>
   <State name=string />1..n
</RevelationConditions>
```

Example 35. *If variable X represents the decision of whether performing a test and Y represents the result of the test, the value X = "do test" reveals the value of Y, which can be either "positive" or "negative", but D = "do not test" does not reveal the value of Y. The link $X \rightarrow Y$ can be encoded as follows:*

```
<Link directed="true ">
   <Variable name="Dec:Test" />
   <Variable name="Result of test" />
   <RevelationConditions>
      <State name="do test"/>
   </RevelationConditions>
</Link>
```

If X is numeric, the syntax is (thresholds are explained in Sec. 4.2.2):

```
<RevelationConditions>
   <Threshold value=number belongsTo=enumSide />2..2n
</RevelationConditions>
```

where the number of thresholds must be even, and each pair defines an interval of values of X.

The third way of representing the flow of information is to divide the DAN into several stages, but we have not implemented it yet because we have found no real-world problem requiring it [15].

6. Additional information

In this section we describe three types of information that can be used when doing inference with a probabilistic network. Each of these types of information can be stored in the same file as the network or in a different file, as explained in Section 3.

6.1. Inference options

6.1.1. General inference options

Inference options are used to indicate how to evaluate a probabilistic network. The skeleton for the inference options is as follows:

```
<InferenceOptions>
  <EliminationOrder>0..n
  <Algorithm name=string >
    <EliminationOrder>0..n
    <Argument name=string value=string />0..n
  </Algorithm>0..n
</InferenceOptions>
```

The elimination order given outside the tag **Algorithm** is common to all the algorithms that wish to use it; this order can be overridden for each particular algorithm. The skeleton of the elimination order is:

```
<EliminationOrder>
  <Variable name=string >1..n
</EliminationOrder>
```

6.1.2. Inference options for dynamic models

There are also specific options for evaluating dynamic models.

```
<Horizon>number</Horizon>0..1
<DiscountRate numTimeUnits=number>string</DiscountRate>0..1
```

The optional attribute numTimeUnits inside the tag **<DiscountRate>** means the period of time for which the discount applies. If numTimeUnits is omitted, the discount applies to a cycle.

Example 36. *If the cycle length of the model is one year (or 12 months), a discount of 5% per annum can be expressed as*

```
<DiscountRate>0.05</DiscountRate>
```

If ***TimeUnit*** *is month, it can be expressed as*

```
<DiscountRate numTimeUnits="12">0.05</DiscountRate>
```

6.2. Evidence

A **finding** is the assignment of a value to a variable. A set of findings is an **evidence case**. Therefore, the skeleton for evidence is:

```
<Evidence>
  <EvidenceCase>
    <Finding variable=string
      state=string stateIndex=integer numericValue=number />0..n
  </EvidenceCase>1..n
</Evidence>
```

The attribute variable is compulsory. If the variable is of type finite states, then either state or stateIndex must be present. If the variable is numeric, then numericValue must be present. If the variable is discretized, one of the three attributes—and only one—must be present.

6.3. Policies and strategies

A **policy** for a decision D given a set of informational predecessors $\mathbf{X}$ (the variables known by the decision maker) can be expressed as a probability distribution on the domain of D for each configuration of $\mathbf{X}$: $P(d|\mathbf{x})$. Therefore, a policy is very similar to a conditional probability potential. In particular, a **deterministic policy** can be given by a delta potential (cf. sec. 4.4.3) for each configuration of $\mathbf{X}$.

The evaluation of a probabilistic network that includes decisions returns a policy for each decision. There may also be policies imposed by the user. A set of policies constitutes a **strategy**. Strategies can be stored with the purpose of doing inference on a probabilistic network without the need to find the optimal strategy in each particular situation. The skeleton of a strategy is very simple:

```
<Strategy>
  <Potential type=enumPotentialType role="Policy" >1..n
</Strategy>
```

The specification of a policy potential is very similar to that of a conditional probability potential, with the decision D playing the role of the conditioned variable.

7. Discussion

7.1. About parsers and writers

The main advantage of the format proposed in this paper is its expressive power: ProbModelXML is able to represent more types of networks and potentials than any other format, and offers facilities for encoding properties that are not explicitly included in the specification of the format.

Another advantage of ProbModelXML is the use of an XML syntax. Conceptually this is not a significant difference with respect to other formats, but in practice this characteristic is relevant mainly because **XML is much easier to parse** than other types of syntaxes: there exist many tools for parsing XML from several programming languages (Java, C++, etc.), with the corresponding utilities for writing XML files, as well as many other tools for specifying XML formats and for validating them (DTD, XML Schema, Relax NG, ISO DSDL...).

At first sight, this expressive power might be disheartening for a programmer interested in using this format, due to the difficulty of building a parser that implements all its features. However, we do not intend that each parser of ProbModelXML implements every tag and attribute of the format. In fact, OpenMarkov, does not have yet all the data structures defined in the specification of this format; consequently its parser and its writer are not yet able to deal with all the types of potentials that ProbModelXML can encode. More generally, a writer of ProbModelXML included in a software package does not need to be able to generate all the properties of ProbModelXML, but only those corresponding to the data structures of that package. Similarly, a parser may limit itself to a subset of ProbModelXML; it may ignore the tags and attributes that it does understand and throw an error message when an attribute recognized by the parser has an unexpected value, as explained in Section 3.2. This possibility of ignoring tags and attributes is facilitated by the XML syntax of the format proposed in this paper.

7.2. Future work

The skeletons used in this paper to describe the syntax of ProbModelXML are very intuitive, but due to the limited expressive power of skeletons, they are by themselves insufficient to specify the format: they must be complemented with the explanations given in the text. One of the first tasks that we will undertake when ProbModelXML reaches a relatively stable situation is to create an XML schema, either an XML Schema Document (XSD) or a RELAX NG schema.

We will also extend ProbModelXML to cover submodels (as in GENIE), new types of potentials (such as mixtures of polynomials [36, 37]) and new types of networks, such as object-oriented Bayesian networks [47] and probabilistic relational models [24, 46].

8. Conclusion

In this paper we have described the main features of a new XML format for encoding probabilistic graphical models (PGMs). One of its advantages is the possibility of representing several types of models, such as Bayesian networks, Markov networks, influence diagrams, LIMIDs, as well as dynamic models: dynamic Bayesian networks, MDPs, POMDP, and DLIMIDs, and it is easy to add new types of models by combining the existing constraints or by defining new ones. Another advantage is the possibility of encoding user-defined properties without modifying the specification of the format, by placing them under the **`AdditionalProperties`** tag. In contrast, the formats proposed previously can represent either Bayesian networks (sometimes in conjunction with influence diagrams) or POMDPs—with the exception of Netica's DNET, which admits both types of models—and none of them can encode user-defined properties. An advan-

tage of our format with respect to others, such as DNET and Elvira, is its XML syntax, which permits to use the utilities available for generating parsers and writers in different languages: Java, C++, etc.

For these reasons, we believe that ProbModelXML may be very useful as an interchange format for PGMs. Clearly, OpenMarkovXML is a very rich format, which makes it difficult to implement all its features. However, each software package can implement only the subset of features required for its purpose. It suffices to throw an error message when the parser encounters such a feature. Even our software tool OpenMarkov is currently unable to cope with several features of the format, but we have decided to include them in the format to satisfy the needs of other research groups. For this reason, we believe that the format presented in this paper can be very useful for interchanging several types of PGMs between different software tools and research groups.

Acknowledgments

This work has been supported by grants TIN2006-11152 and TIN2009-09158 of the Spanish Ministry of Science and Technology, by FONCICYT grant 85195, and by the OptiFox project 262266 (7th Frame Programme of the European Union).

Appendix A Constraints used in OpenMarkov

Table 5 lists the constraints implemented in the current version of OpenMarkov and whether each one of the is associated with a particular network type; network types are defined in Section 4.1.1 and constraints in Section 4.1.2. The meaning of each constraint is given below. Please note that influence diagrams and LIMIDs have the same constraints, but their semantic is different, and consequently they should be evaluated with different algorithms. The same occurs for MDPs and POMDPs.

A.1 Constraints about nodes and variables

NoEmptyName: The name of a variable must be a non-empty string.

DistinctVariableNames: The name of the variables must be different.

OnlyFiniteStateVariables: All the chance and decision variables must be of finite-state or discretized (see Sec. 2.3). However, this restriction does not affect utility variables, which in ProbModelXMLare assumed to be always numerical. This constraint it is used by many inference algorithms.

OnlyNumericVariables: All variables must be purely numeric. Discretized variables are not admitted, because they are treated as finite-state variables.

OnlyChanceNodes: All nodes must have the role Chance. Decision and Utility nodes are not allowed. This constraint is used, for instance, to define Bayesian networks.

	BayesianNetwork	MarkovNetwork	InfluenceDiagram	LIMID	DecisionAnalysisNetwork	DynamicBayesianNetwork	SimpleMarkovModel	MDP	POMDP	Dec-POMDP	DynamicLIMID
NoEmptyName	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
DistinctVariableNames	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
OnlyFiniteStatesVariables	O	O	O	O	O	O	O	O	O	O	O
OnlyNumericVariables	O	O	O	O	O	O	O	O	O	O	O
OnlyChanceNodes	**Y**	**Y**	N	N	N	**Y**	N	N	N	N	N
OnlyOneUtilityNode	–	–	O	O	O	–	O	O	O	O	O
OnlyAtemporalVariables	Y	Y	Y	Y	Y	**N**	**N**	**N**	**N**	**N**	**N**
OnlyTemporalVariables	N	N	N	N	N	**Y**	N	**Y**	**Y**	**Y**	**Y**
OnlyOneAgent	Y	Y	Y	Y	Y	Y	Y	Y	Y	**N**	Y
DistinctLinks	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
NoMultipleLinks	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
OnlyDirectedLinks	Y	**N**	Y	Y	Y	Y	Y	Y	Y	Y	Y
OnlyUndirectedLinks	N	**Y**	N	N	N	N	N	N	N	N	N
NoRestriction	Y	Y	Y	Y	**N**	Y	Y	Y	Y	Y	Y
NoRevelationArc	Y	Y	Y	Y	**N**	Y	Y	Y	Y	Y	Y
NoSelfLoop	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
NoCycle	Y	**O**	Y	Y	Y	Y	Y	Y	Y	Y	Y
NoClosedPath	O	O	O	O	O	O	O	O	O	O	O
MaxNumParents	O	O	O	O	O	O	O	O	O	O	O
NoUtilityParent	–	–	O	O	O	–	O	O	O	O	O
NoSupervalueNode	–	–	O	O	O	–	O	O	O	O	O
NoMixedParents	–	–	O	O	O	–	O	O	O	O	O
NoBackwardLink	–	–	–	–	–	Y	Y	Y	Y	Y	Y

Table 5: Constraints used in the OpenMarkov tool. The letter in each cell indicates whether a constraint is associated with a network type: Y = yes, N = no, O = optionally. A dash means that a constraint does not make sense because of the presence of another constraint (see the text for a more detailed explanation). Each constraint has a default behavior; a bold letter in this table means that the default behavior has been overriden for a particular type of network.

OnlyOneUtilityNode: The network contains at most one atemporal utility node and, in the case of dynamic models, at most one utility node per time slice. This constraint does not make sense when OnlyChanceNodes is present.

OnlyAtemporalVariables: The model contains no temporal variable.

OnlyTemporalVariables: Used by most dynamic models (Secs. 2.4 and 5.1). In the current version of ProbModelXML, the only type of dynamic model that does not have this constraint is SimpleMarkovModel , which accept both temporal and atemporal variables.

OnlyOneAgent: All the variables in the network correspond to the same agent. Obviously, multi-agent models do not have this constraint. Currently the only multi-agent type of model in ProbModelXMLare Dec-POMDPs.

A.2 Constraints about links (structure of the graph)

There are several constraints relative to the structure of the graph. Some of them are self-explanatory given the definitions in Section 2.2.

DistinctLinks: The network cannot have two equal links. For example, the network cannot have two links $X \rightarrow Y$. A link $X \rightarrow Y$ is different from $Y \rightarrow X$ and X — Y, but the latter is considered to be the same as Y — X (see Sec. 2.2).

NoMultipleLinks: This constraint makes the undirected link $A - B$ incompatible with both $A \rightarrow B$ and $B \rightarrow A$; however $A \rightarrow B$ would be compatible with $B \rightarrow A$. This restriction will be used, for instance, when building chain graphs, but they are not included in ProbModelXML yet.

Figure 9: The constraint *DistinctLinks* permits this graph, but *NoMultipleLinks* does not.

OnlyDirectedLinks: This constrain is used by most PGMs, such as Bayesian networks, influence diagrams, etc.

OnlyUndirectedLinks: Currently this constraint is used only by Markov networks. It is incompatible with OnlyDirectedLinks and vice versa.

NoRestriction: A constraint (x,y) associated with an arc $X \rightarrow Y$ means that the values x and y are incompatible [15]; for example, the decision of discharging a patient now (x) prevents the performance of a test a few hours later (y). Currently all the network types have this constraint, except decision analysis networks (see Sec. 5.2), but in the future we might include in ProbModelXML other types of networks free from this constraint, such as influence diagrams with (link) constraints [38].

NoRevelationArc: Revelation arcs are used by DANs to indicate that a variable reveals the value of another variable [15]; for example, the decision of performing a test "reveals" the result of the test (see Sec. 5.2). A network that admits revelation arcs can also indicate that some variables are always observed (Sec. 5.2). Currently DANs are the only network type that admits revelation arcs and always-observed variables.

NoSelfLoop: Self-loops were defined in Sec. 2.2 as a link connecting a node to itself. None of the PGMs defined currently in ProbModelXMLcan contain self-loops. However, when representing dynamic models in Netica and GeNIE, temporal nodes have self-loops indicating that the value of a variable at a certain time influences the value of the same variable in the future.

NoCycle: It is used by most PGMs, such as Bayesian networks, influence diagrams, etc. It permits the presence of loops.

NoClosedPath: It forbids both cycles and loops (see Sec. 2.2 for the definitions). In the case of undirected graphs, this constraint implies that the graph of the model is a tree. In the case of directed graphs, it implies that the graph is polytree.

MaxNumParents: It limits the number of parents that a node may have, as specified by its argument.

NoUtilityParent: This constraint forbids that a chance or decision node has a utility node as a parent. However, it does not forbid that a utility node be a parent of another utility node (the latter would be called *supervalue* node [41]). This constraint does not make sense when OnlyChanceNodes is present.

NoSuperValueNode: A supervalue node is a utility node such that some of its parents are other utility nodes. This constraint implies that all the parents of a utility node must play the role chance or decision. It does not make sense when OnlyChanceNodes is present.

NoMixedParents: This restriction establishes that all the parents of a utility node belong to only one of these two sets of parents: (1) chance and decision nodes, or (2) utility nodes. This constraint does not make sense when OnlyChanceNodes is present.

The standard definitions of most decision models (influence diagrams, LIMIDs, MDPs, etc.) reject the possibility of supervalue nodes and mixed parents. The only exceptions are decision analysis networks and dynamic LIMIDs. Given that ProbModelXMLadmits super-value nodes and mixed parents in all decision models, most algorithms should check that the constraints NoSuperValueNode and NoMixedParents are satisfied before trying to evaluate any of those models (cf. Sec. 4.1.2).

NoBackwardLinks: In temporal models, it forbids links to the past. This constraint does not make sense when OnlyAtemporalVariables is present.

Appendix B Changelog

Changes introduced in version 0.2.0

- We have introduced new inference options (Sec. 6.1) and the syntax for encoding policies (Sec. 6.3).
- The arguments of the constraints are now encoded as "**<Argument** name=*string* value=*string* **>**" instead of "**<Argument>***string***</Argument>**" (Sec. 4.1.2).
- There is a new potential, Sum (Sec. 4.4.6).
- Link constraints (Sec. 5.2) have been renamed as *restrictions* to avoid confusions with network constraints. Consequently, the role LinkConstraints (Sec. 4.4) has been renamed Restrictions
- There is a new constraint, NoRestriction (Sec. A.2).
- The tag **NumericVariables** has been replaced with **Variables** in the potentials LinearCombination and ConditionalGaussian, but it is kept in Exponential. The tags **Variables** and **Subpotentials** have been added to MixtureOfExponentials (Sec. 4.4.10.b).
- The tag **SpontaneouslyObserved** has been renamed as **AlwaysObserved** (Sec. 5.2).
- The tag **Label**, that was used in links, has been removed. Consequently, the constraint OnlyUnlabeledLinks has been removed as well.
- In the previous version, a **Potential** might have the attributes label and ref, used to encode ADDs containing loops. These attributes have been replaced by the new tags **Label** and **Reference** used inside the branches of ADDs (Sec. 4.4.4.b).

Changes introduced in version 0.1.9

This section describes the changes introduced with respect to previous versions of the format (openmarkovXML-draft-110203.pdf, released on February 3, 2011).

- The name of the format has been changed from OpenMarkovXML to ProbModelXML to detach it from any particular software tool—see Footnote 2.
- We have added the network tags **DecisionCriteria** and **Agents**, used for multicriteria decision making (Sec. 4.1.4) and for multi-agent models (Sec. 4.1.5), respectively. Correspondingly, we have introduced the attributes criterion and agent in the specification of variables (Sec. 4.2).
- The specification of the coordinates of a variable (Sec. 4.2) has changed from **<Coordinates>***integer integer***</Coordinates>** to **<Coordinates** x=*integer* y=*integer* **/>**.

- In the specification of the domain of a numeric variable (Sec. 4.2.2), we now write **<Intervals>** instead of **<Thresholds>**, to make it coherent with the case of discretized variables (Sec. 4.2.3).

- The new specification of links does not use the old attributes var1 and var2, but two **Variable** tags (Sec. 4.3).

- We have replaced the tag **ExtendedValues** with **UncertainValues**. We have simplified the specification of second order probability distributions for the parameters of tables (Sec. 4.4.2.b).

- Now tree/ADD potentials include a **Variables** tag that specifies the set of variables on which the potential is defined. However, it may happen that some of these variables do not appear inside the tree/ADD, as explained in Section 4.4.4.

  Now the **TopVariable** is compulsory for each tree; in the previous version, it was allowed to omit this tag when the tree depends on only one variable.

  Similarly, now each branch must have a **States** or **Thresholds** tag; in the previous version, these tags could be omitted in the last branch.

  These changes, aimed at making all the information explicit, facilitate the parsing of tree/ADDs.

- We have introduced the attribute timeSlice for dynamic variables (Sec. 5.1.2). Now we write **<Variable** name="Disease" timeSlice="0"**>** instead of **<Variable** name="Disease [0]"**>**, which simplifies the parsing of dynamic variables.

- We have added the following potentials: Uniform (Sec. 4.4.1), Product (Sec. 4.4.6), SameAsPrevious (Sec. 5.1.3.a).

- We have introduced specific options for decision analysis networks (DANs, Sec. 5.2):

  - a new network type (Sec. 4.1.1),
  - a new tag for variables, **SpontaneouslyObserved** (Sec. 4.2), renamed as **AlwaysObserved** in version 0.2.0.
  - a new tag for links, **RevelationConditions** (Sec. 4.3),
  - a new role for potentials, LinkConstraints (Sec. 4.4), renamed as Restrictions in version 0.2.0; a potential having this role can be attached to a link to declare some restrictions (Sec. 4.3), and
  - a new constraint, NoRevelationArc, in Section A.1 and Table 5.

- Appendix A, which describes the constraints used in OpenMarkov, has been completely revised, especially Table 5.

References

[1] M. Arias. *Carmen: Una Herramienta de Software Libre para Modelos Gráficos Probabilistas*. PhD thesis, UNED, Madrid, 2009. In Spanish. 4

[2] G. Arroyo-Figueroa and L. E. Sucar. A temporal Bayesian network for diagnosis and prediction. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI'99)*, pages 13–20, Stockholm, Sweden, 1999. Morgan Kaufmann, San Francisco, CA. 3

[3] K. J. Åström. Optimal control of Markov decision processes with incomplete state estimation. *Journal of Mathematical Analysis and Applications*, 10:174–205, 1965. 3, 14

[4] L. E. Baum and T. Petrie. Statistical inference for probabilistic functions of finite state Markov chains. *Annals of Mathematical Statistics*, 37, 1966. 3

[5] R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957. 3, 10, 14

[6] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, 27:819–840, 2002. 9, 14

[7] C. Bielza, M. Gómez, and P. P. Shenoy. A review of representation issues and modelling challenges with influence diagrams. *Omega*, 39:227–241, 2011. 53

[8] C. Boutilier, R. Dearden, and M. Goldszmidt. Exploiting structure in policy construction. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI–95)*, pages 1104–1111, Montreal, Canada, 1995. 3, 9, 14

[9] C. Boutilier, R. Dearden, and M. Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1-2):49–107, 2000. 3, 9, 10, 14

[10] C. Boutilier and D. Poole. Computing optimal policies for partially observable decision processes using compact representations. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96*, pages 1168–1175, Portland, OR, 1996. AAAI Press. 3, 9, 10, 14

[11] A. Briggs, K. Claxton, and M. Sculpher. *Decision Modelling for Health Economic Evaluation*. Oxford University Press, New York, 2006. 51, 52

[12] A. Cano, S. Moral, and A. Salmerón. Penniless propagation in join trees. *International Journal of Intelligent Systems*, 15:1027–1059, 2000. 31

[13] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5:142–150, 1989. 3, 9, 14

[14] F. J. Díez and M. J. Druzdzel. Canonical probabilistic models for knowledge engineering. Technical Report CISIAD-06-01, UNED, Madrid, Spain, 2006. 5, 38

[15] F. J. Díez and M. Luque. Representing decision problems with Decision Analysis Networks. Technical Report CISIAD-10-01, UNED, Madrid, Spain, 2010. 3, 10, 14, 40, 53, 54, 60, 61

[16] F. J. Díez, M. A. Palacios, and M. Arias. MDPs in medicine: opportunities and challenges. In *Decision Making in Partially Observable, Uncertain Worlds: Exploring Insights from Multiple Communities (IJCAI Workshop)*, Barcelona (Spain), 2011. 9, 14

[17] F. J. Díez and M. A. J. van Gerven. Dynamic LIMIDs. In L. E. Sucar, J. Hoey, and E. Morales, editors, *Decision Theory Models for Applications in Artificial Intelligence: Concepts and Solutions*, pages 164–189. IGI Global, Hershey, PA, 2011. 3, 9, 10

[18] The Elvira Consortium. Elvira: An environment for creating and using probabilistic graphical models. In J. A. Gámez and A. Salmerón, editors, *Proceedings of the First European Workshop on Probabilistic Graphical Models (PGM'02)*, pages 1–11, Cuenca, Spain, 2002. 5

[19] S. F. Galán, G. Arroyo-Figueroa, F. J. Díez, and L. E. Sucar. Comparison of two types of event Bayesian networks: A case study. *Applied Artificial Intelligence*, 21:185–209, 2007. 3

[20] S. F. Galán and F. J. Díez. Networks of probabilistic events in discrete time. *International Journal of Approximate Reasoning*, 30:181–202, 2002. 3

[21] J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier. SPUDD: Stochastic planning using decision diagrams. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI'99)*, pages 279–288, Stockholm, Sweden, 1999. Morgan Kaufmann, San Francisco, CA. 6

[22] R. A. Howard and J. E. Matheson. Influence diagrams. In R. A. Howard and J. E. Matheson, editors, *Readings on the Principles and Applications of Decision Analysis*, pages 719–762. Strategic Decisions Group, Menlo Park, CA, 1984. Reprinted as [23]. 3, 10, 13

[23] R. A. Howard and J. E. Matheson. Influence diagrams. *Decision Analysis*, 2:127–143, 2005. 65

[24] M. Jaeger. Relational Bayesian networks. In *Proceedings of the Thirteenth Conference in Artificial Intelligence (UAI-97)*, page 266273, San Francisco, CA, 1997. Morgan Kaufmann. 57

[25] D. Koller, U. Lerner, and D. Anguelov. A general algorithm for approximate inference and its application to hybrid Bayes nets. In *Proceedings of the Fifteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI'99)*, pages 324–333, San Francisco, CA, 1999. Morgan Kaufmann. 45

[26] C. Lacave, M. Luque, and F. J. Díez. Explanation of Bayesian networks and influence diagrams in Elvira. *IEEE Transactions on Systems, Man and Cybernetics—Part B: Cybernetics*, 37:952–965, 2007. 12, 38

[27] S. L. Lauritzen and F. Jensen. Stable local computation with conditional Gaussian distributions. *Statistics and Computing*, 11:191–203, 2001. 43

[28] S. L. Lauritzen and D. Nilsson. Representing and solving decision problems with limited information. *Management Science*, 47:1235–1251, 2001. 13

[29] S. L. Lauritzen and N. Wermuth. Graphical models for associations between variables, some of which are qualitative and some quantitative. *The Annals of Statistics*, 17:31–57, 1989. 43

[30] A. A. Markov. Rasprostranenie zakona bol'shih chisel na velichiny, zavisyaschie drug ot druga. *Izvestiya Fiziko-matematicheskogo obschestva pri Kazanskom universitete*, 2-ya seriya, 15:135–156, 1906. 3

[31] S. Moral, R. Rumí, and A. Salmerón. Mixtures of truncated exponentials in hybrid Bayesian networks. *Lecture Notes in Artificial Intelligence*, 2143:156–167, 2001. 45

[32] K. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, Computer Science Division, University of California, Berkeley, 2002. 3, 9

[33] S. C.W. Ong, S. W. Png, D. Hsu, and W. S. Lee. POMDPs for robotic tasks with mixed observability. In *Proceedings of Robotics: Science and Systems V*, Seattle, WA, 2009. 7, 14

[34] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA, 1988. 3, 7, 13, 40

[35] P. Poupart. *Exploiting Structure to Efficiently Solve Large Scale Partially Observable Markov Decision Processes*. PhD thesis, Dept. of Computer Science, University of Toronto, Canada, 2005. 6

[36] P. P. Shenoy. A re-definition of mixtures of polynomials for inference in hybrid Bayesian networks. In W. Liu, editor, *Proceedings of the 11th European conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU'11)*, pages 98–109. Springer, Heidelberg, 2011. 57

[37] P. P. Shenoy and J. C. West. Inference in hybrid Bayesian networks using mixtures of polynomials. *International Journal of Approximate Reasoning*, 52:641–657, 2010. 57

[38] J. E. Smith, S. Holtzman, and J. E. Matheson. Structuring conditional relationships in influence diagrams. *Operations Research*, 41:280–297, 1993. 60

[39] M. T. J. Spaan and N. Vlassis. Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, 24:195–220, 2005. 6

[40] A. A. Stinnett and J. Mullahy. Net health benefit: A new framework for the analysis of uncertainty in cost-effectiveness analysis. *Medical Decision Making*, 18:S68–S80, 1998. 40

[41] J. A. Tatman and R. D. Shachter. Dynamic programming and influence diagrams. *IEEE Transactions on Systems, Man, and Cybernetics*, 20:365–379, 1990. 39, 61

[42] M. A. J. van Gerven and F. J. Díez. Selecting strategies for infinite-horizon dynamic LIMIDs. In M. Studený and J. Vomlel, editors, *Proceedings of the Third European Workshop on Probabilistic Graphical Models (PGM'06)*, pages 131–138, Prague, Czech Republic, 2006. 14

[43] M. A. J. van Gerven, F. J. Díez, B. G. Taal, and P. J. F. Lucas. Selecting treatment strategies with dynamic limited-memory influence diagrams. *Artificial Intelligence in Medicine*, 40:171–186, 2007. 3, 9, 10, 14

[44] W. Weibull. A statistical theory of the strength of materials. *Ingenjörsvetenskapsakademiens Handlingar*, 151:1–45, 1939. 51

[45] W. Weibull. A statistical distibution function of wide applicablity. *ASME Journal of Applied Mechanics, Transactions of the American Society of Mechanical Engineers*, pages 293–297, September 1951. 51

[46] D. Koller y A. Pfeffer. Probabilistic frame-based systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98*, pages 580–587, Madison, WI, 1996. 57

[47] D. Koller y A. Pfeffer. Object-oriented Bayesian networks. In *Proceedings of the Thirteenth Conference in Artificial Intelligence (UAI-97)*, pages 302–313, San Francisco, CA, 1997. Morgan Kaufmann. 57