A Prototype Implementation of BayesOWL

Yi Sun

A paper submitted to the Computer Science and Electrical Engineering

Department

in partial fulfillment of the requirements for the M.S. degree at

University of Maryland Baltimore County

June 2009

Certified by: ______________________________________

Dr. Yun Peng, Advisor

Abstract

This project aims to build a prototype software system for BayesOWL, a probabilistic framework proposed for dealing with uncertainty in Semantic Web (SW) ontologies. It translates a terminological taxonomy of an OWL ontology into a Bayesian Network (BN), integrates probabilistic information about the concept classes and interclass relations into the translated BN, and supports important ontological reasoning tasks as probabilistic reasoning in BN.

The implementation of the BayesOWL framework extends the original framework in two ways. One is encoding probabilistic information in OWL format. The new convention can encode any discrete probabilities in a general form with one or more prior variables and zero or more conditional variables. The other extension is with the input ontologies. In the original BayesOWL framework, only simple ontologies which themselves are terminological taxonomies can be handled. But in this implementation, general OWL ontologies (OWL DL) are supported by taking advantage of existing OWL reasoning tools. These two developments together represent a significant advancement over the original framework.

The prototype system **BayesOWL 1.0** extracts the taxonomy of the named classes from the given OWL ontology and maps this terminological taxonomy into a BN following a set of structural translation rules; it extracts probabilistic uncertainty information from the probability file and incorporates it into the translated BNs using a set of algorithms based on the iterative proportional fitting procedure (IPFP). Finally a resulting BN will be generated as the output.

This prototype system for BayesOWL can be used as a practical tool for ontology engineering tasks such as domain modeling, ontology reasoning and ontology concept mapping.

1 INTRODUCTION

Several approaches have been proposed to realize semantic interoperability among heterogeneous system using OWL ontologies. However, no existing software tools for these approaches can be used by researchers in ontology engineering. In this report, we aim to implement a prototype system of BayesOWL [4, 6], a Bayesian Network based framework proposed for handling uncertainty in the Semantic Web.

1.1 OWL: Web Ontology Language

The Semantic Web (SW) [17] is proposed as a vision for the future of the Web which gives information on the web explicit meanings that can be understood and properly processed by machines. For this purpose, Web Ontology Language (OWL) [18] was proposed and recommended by W3C for defining ontologies that can be used and shared by web contents in describing their semantics. OWL is an extension of RDF [19] based on description logics and goes beyond the basic semantics of RDF Schema. It relies on ontologies to define terms used to model the domain knowledge.

1.2 Why Use Bayesian Network

Since OWL uses crisp description logic to represent a domain, it cannot deal with incomplete, imprecise, or partial knowledge about the domain. But in the real world, uncertainty exists in almost all aspects of areas, including domain modeling, ontology reasoning, concept mapping, etc. So how to do representation and reason under uncertainty is gaining more and more attention. Many recently developed approaches to handle uncertainty SW are based on probability theory [21] because of its mathematical representation language and formal calculus for rational degrees of belief.

Several works [7][8][9][15], including the BayesOWL [4][6], have used Bayesian Networks (BN) [11][12] to model uncertainty in SW. A BN consists of a directed acyclic graph (DAG) and a set of local conditional probability tables (CPTs), one per each random variable. The DAG represents qualitative dependency relationships between random variables while the CPTs quantify the strength of these dependences.

The DAG and CPTs together determine a joint probability distribution (JPD), which can be viewed as a probabilistic knowledge base of the domain. Theoretically, using such a JPD and existing BN inference algorithms, e.g. belief propagation, junction tree, etc., BN can answer any probabilistic queries about the domain [11].

1.3 BayesOWL

BayesOWL [4][6] is a probabilistic framework that augments and supplements OWL for representing and reasoning with uncertainty in SW ontologies based on BN. It consists of a set of translation rules which are used to convert OWL ontology into a BN DAG, a convention of encoding probabilistic information, and a construction mechanism for BN CPTs to integrate available probabilistic information into the JPD of the translated BN. In the resulting BN, each concept class of the given ontology is translated into a concept node, each logical relation is represented as a logic node (L-node for short), and the uncertainty of the inter-class relations are captured by the CPTs. The final BN can be used for ontology engineering tasks, such as domain modeling, ontology reasoning and ontology concept mapping, etc.

The purpose of this project is to implement a software system for BayesOWL, so it can be used as a practical tool by researchers dealing with uncertainty in ontology engineering areas. The prototype implementation of BayesOWL includes a source code package (a .jar file), a graphical user interface, and related documents. In addition, use cases are provided to help others use the software tool and verify the implementation performance.

The rest of this report is organized as follows: Section 2 briefly discusses BayesOWL framework; Section 3 provides detailed description of the extension and implementation of BayesOWL 1.0; Section 4 shows some use cases; and conclusions are given in Section 5.

2 BayesOWL FRAMEWORK

According to the framework, BayesOWL 1.0 has two main functions. One is translating OWL ontologies into BNs while the other one is constructing Conditional

Probability Tables for translated BNs. As can be seen in Figure 2.1 below, translation from a probabilistic OWL ontology to a BN by the prototype system is done in two stages. The first stage is to construct the BN structure (DAG) from the input OWL ontology file and to initialize the conditional probability tables (CPTs) with default values. This is done by *Taxonomy Parser* and *BN structure constructor*. The second stage is to incorporate user provided probabilistic information of concepts and inter-concept relations into the BN CPTs. This is done by *Probability Parser* and *CPT Constructor*.

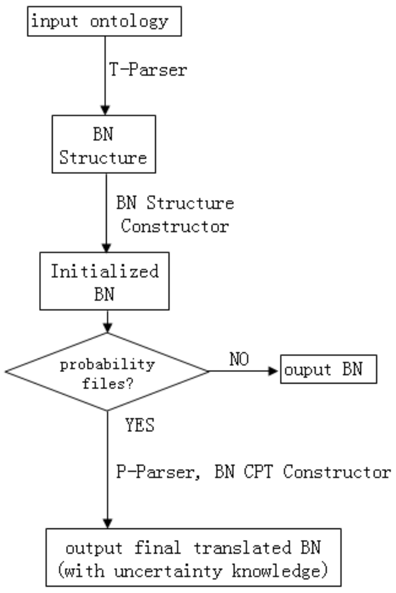


Figure 2.1 BayesOWL 1.0 framework

2.1 Convert Ontology Taxonomy into Bayesian Network

Although SW builds on XML's ability to define customized tagging scheme and RDF's flexible approach to represent data, OWL ontology is not a message format but a knowledge representation: it typically includes descriptions of classes, properties and their instances. Original BayesOWL framework does not deal with properties and individuals. Instead, it focuses on simple, taxonomical ontologies, consisting of concept classes and logical relations between these classes. Six types of logical relations are allowed in OWL by its class axioms and logical constructors:

- rdfs:subClassOf
- owl:equivalentClass

- owl:disjointWith
- owl:unionOf
- owl:intersectionOf
- owl:complementOf

In this prototype implementation, we extended the original framework and went one step further toward modeling the general OWL DL ontologies. In a general ontology, concept classes not only related to each other by the explicitly defined relations, but also implicitly by properties, which are represented as anonymous classes in the RDF graph of the ontology. These anonymous classes are classes without identifiers (ID) or names and they cannot be referenced by other ontologies. Thus when we translate taxonomies into BNs, these anonymous classes will not be translated into variables in BN. On the other hand, we cannot simply drop them because logical relationships between other concept classes may depend on these dummy classes. For example, in “Wine” ontology provided by W3C as a use example and test benchmark [25], there is no explicit definition of relationship between concept classes “RedBurgundy” and “WhiteWine”. However, they are disjoint with each other because they are subclasses of two disjoint anonymous classes “hasColor = #Red” and “hasColor = #White”.

One can find all relations, explicit and implicit, using a full OWL DL reasoner. However, not all of these relations need to be modeled in the BN since some may be derived from each other. To handle these anonymous classes and implicit relationships, we have extended the original structure translation rules as follows.

Let S_{ONT} denote all explicit and implicit relationships between concept classes of the given ontology. We define $S_{DAG} \subseteq S_{ONT}$ as a set of logical relations such that:

1) every L in S_{ONT} is entailed by S_{DAG}; and

2) no L in S_{DAG} is entailed by $S_{DAG} \backslash \{L\}$.

Then, we only need to explicitly model those L in S_{DAG} by subclass arcs and L-nodes in the translated BN. The extended structural translation rules are as follows:

1. **Concept Classes.** Each defined concept class is mapped into a binary variable node, called concept node, in the translated BN.

2. **Logical Relations.** Form S_{DAG} from the given ontology.
3. **Subclasses.** For every subclass relation L_{sub} in S_{DAG}, an arc is set from the superclass to the subclass. A subclass hierarchy of all concept nodes is thus formed.
4. **Logical Nodes.** For every non-subclass relation in S_{DAG}, create an L-node and set its CPT according to the logic of the relation.

It can be seen that the translated DAG preserves the semantics of the given ontology.

In BayesOWL 1.0, *Taxonomy Parser* (T-Parser for short) extracts taxonomies defined in the given ontology, find relationships between concept classes using ontology reasoner tool, and remove redundant relations (relations can be derived from other relations). Details of *T-Parser* will be given in Subsection 3.2.1.

When extraction work is finished, another component, *BN Constructor*, would be provided to construct BN structure. It takes advantage of the translation rules to convert concept classes and relationships mentioned above into a BN DAG. Such a BN will also be initialized by filling its CPTs and set the logic nodes [2][4] to be 'True', so the relationships defined in OWL ontology hold.

2.2 How to Represent Probability Information

In many applications, probabilistic information of concept nodes and inter-concept relations such as prior probabilities for individual concepts and pair-wise conditional probabilities for subclass relations may be available for the given ontology. Before uncertainty information can be used by BayesOWL, it should be encoded into some specific format. Since ontologies are written in OWL, it is natural to make use of OWL syntax when encoding probabilistic information.

Original BayesOWL framework has proposed an extension of OWL to represent probabilistic information [3]. However, it deals with only two types of probabilities: the marginals for individual concepts ($P(c)$) and pair-wise conditionals of subclass relations ($P(c|a)$), where A is a most specific superclass of C. A more general convention is proposed and adopted in this implementation. The extended convention can represent general form of probability with more than one prior and conditional variable (see Subsection 3.2.2 for details).

2.3 Construct Conditional Probability Tables (CPTs)

CPTs for Logic Nodes, which are corresponding relationships defined in OWL ontology, can be completely determined by the logical relation it represents. That is, when its state is set to “True”, the intended logical relation among its parents must hold. For example, if C is the intersection of C_1 and C_2, then the L-Node is “True” if and only if $c c_1 c_2 \vee \overline{c}\, c_1 \overline{c}_2 \vee \overline{c}\, \overline{c}_1 c_2 \vee \overline{c}\, \overline{c}_1 \overline{c}_2$.

Table 2.1 CPT for an intersection L-Node

C	*C*$_1$	*C*$_2$	**Intersection**	
			True	False
True	True	True	1.0	0.0
True	True	False	0.0	1.0
True	False	True	0.0	1.0
True	False	False	0.0	1.0
False	True	True	0.0	1.0
False	True	False	1.0	0.0
False	False	True	1.0	0.0
False	False	False	1.0	0.0

To initialize the CPTs for concept nodes, let π_c be the set of all parent nodes of the concept node C. From the structure translation rules, all nodes in π_c are super-classes of C. Therefore, each entry in $P(c \mid \pi_c)$, the CPT of C, must have value 0 if any of its parents is “False” for that entry. The only other entry in the table is $P(c \mid \pi_c^{\ +})$, in which all parents are “True”. It is the probability of this entry that needs to be determined. If no probabilistic information is available, we choose the following as default:

$$P(c = true \mid \pi_C^+) = P(c = false \mid \pi_C^+) = 0.5 .$$

When probabilistic information is available, this information needs to be incorporated into the CPTs of the concept nodes. For this purpose several algorithms are developed [4][13], all based on a mathematical procedure known as iterative proportional fitting procedure (IPFP) [1][14]. These algorithms take the probabilistic information as constraints and iteratively modify the joint distributions of all variables, using one constraint at a time, until a convergence is reached, in which all the constraints are satisfied by the resulting joint distribution.

The original BayesOWL framework only allows constraints in the forms of marginals for individual concepts ($P(c)$) and pair-wise conditionals of subclass relations ($P(c|a)$). Note that each of these constraints only involves variables within one CPT, and these constraints can be efficiently incorporated by one of the algorithms called SD-IPFP, where D stands for decomposed. In BayesOWL 1.0, we aim to incorporate probability constraints in more general forms. Several issues need to be addressed:

A1) Variables in a constraint may belong to several CPTs;

A2) IPFP does not respect the structure of BN;

A3) CPTs should be set in the general space in which all variables are free while constraints are given in the subspace *LT*, in which all L-nodes are set to "True".

To address these concerns, we adopt another algorithm, E-IPFP, from [13], which is given below:

Let $Q_0(x_i | \pi_i)$ be the initial default CPTs and $R = \{R_i(y)\}$ be the set of constraints.

```
1. $Q_0(x) = \Pi_{i=1}^{n} Q_0(x_i | \pi_i)$;
2. for $k$ = 1 until convergence {
3. $\tilde{Q}_{k-1}(x) = Q_{k-1}(x)$;
4. for each $R_i(y) \in R$ do $\tilde{Q}_k(x) = \tilde{Q}_{k-1}(x) \cdot \frac{R_i(y)}{\tilde{Q}_{k-1}(y | LT)} \alpha_i$;
5. for each concept node $x_j$ extract its CPT $Q_k(x_j | \pi_j)$ from $\tilde{Q}_k(x)$;
6. $Q_k(x) = \Pi_{i=1}^{n} Q_k(x_i | \pi_i)$;
7. $k = k + 1$;}
```

Note that $Q_k(x)$ are the joint distribution of all variables (including both concept nodes and L-nodes), which are modified by iteratively using the constraints at Step 3. This takes into account of A1. A2 is taken into account by Steps 5 and 6 which have the effect of imposing the BN structure as an additional constraint; this constraint is satisfied when the joint distribution $\tilde{Q}_k(x)$ after Step 5 equals $Q_k(x)$ after Step 6. A3 is taken into account of by the denominator in the formula in Step 4 which is the probability of *y* in the subspace of *LT*. Here

$$\alpha_i = \frac{1}{\sum_{xi} Q_{(k-1)}(x_i \mid \pi_{xi}) \cdot R(x_i \mid L_i) / Q_{(k-1)}(x_i \mid L_i, LT)}$$

is the normalization factor.

Also note that, since E-IPFP does not modify any zero value entry in the joint distribution, it only changes those entries $P(c \mid {\pi_c}^+)$ in the CPTs of concept nodes. If all constraints are consistent with each other and with the BN structure, then the procedure will converge; the CPTs obtained at Step 5 define a distribution which satisfies all constraints in *R* with the condition of *LT*.

So, after probability is decoded by *P-Parser*, a component called *CPT Constructor* is used to incorporate probabilistic information into previously generated BN. *CPT Constructor* uses algorithms discussed above to incorporate the probabilistic information into the BN.

As we discussed above, BayesOWL 1.0 contains four components, *T-Parser*, *P-Parser*, *BN Constructor*, and *CPT Constructor*. *T-Parser* extracts taxonomies and relationships defined in OWL ontology file. *BN Constructor* constructs BN structure (the DAG) using the extracted taxonomy. *P-Parser* decodes probabilistic information into constraints which can be used by *CPT Constructor* which is implemented to incorporate the uncertainty information into resulting BNs. In the following section, we will discuss the implementation of these components in detail.

3 PROTOTYPE IMPLEMENTATION

This section discusses implementation details of BayesOWL 1.0. It is organized as follows: Subsection 3.1 gives the architecture of BayesOWL 1.0; Subsection 3.2 discusses the parsers of BayesOWL 1.0; Subsections 3.3 and 3.4 discuss BN Constructor and CPT Constructor separately; Subsection 3.5 lists APIs of BayesOWL 1.0; and Subsection 3.6 gives the GUI of BayesOWL 1.0.

3.1 System Architecture

Figure 3.1 shows the BayesOWL 1.0 system architecture.

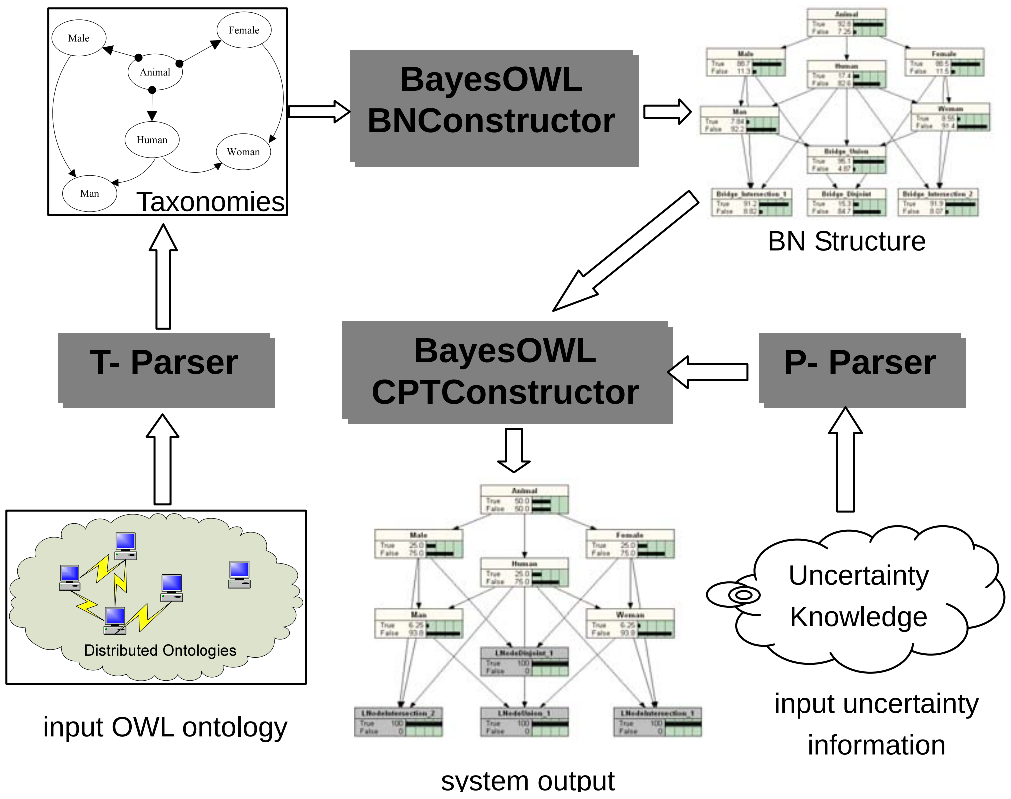


Figure 3.1 BayesOWL 1.0 architecture

As can be seen easily from Figure 3.1, BayesOWL 1.0 works like this:

The *T-Parser* first extracts the terminological taxonomy defined in the given ontology. Output of the *T-Parser* is represented in a specific format which contains items defined in ontology such as ‘class’, ‘subClassOf’, ‘disjointOf’, etc. *BN Constructor* then uses these results and follows the structural translation rules to convert the extracted taxonomy into a BN DAG. If uncertainty knowledge is given, the *CPT Constructor* then uses IPFP-based algorithms (D-IPFP for decomposable cases and E-IPFP for general constraints) given in [13] to construct CPTs for concept nodes.

3.2 BayesOWL 1.0 Parsers

There are two parsers in BayesOWL. One is *T-Parser,* which is used to parse ontology file. The other is *P-Parser*, which is used to parse the file of probabilistic information encoded in OWL format.

3.2.1 T-Parser

As discussed in Subsection 2.1, to extract the taxonomy from the given ontology, we need to deal with the difficult issue of anonymous classes created when parsing OWL DL ontologies. This involves finding all logical relations between concept classes, especially those implicitly defined via these anonymous classes and removing redundant one among them.

There are several OWL reasoners [20], such as Jena [24], FaCT, Pellet [22] etc., developed to parse OWL ontologies. Jena is an open source SW framework using Java. It extracts RDF graphs from ontology files and provides supports for OWL. However, Jena does not include a complete reasoner in its standard distribution for OWL DL entailment. Compared with Jena, FaCT does not have these limitations when reasoning, but it does not provide rule support. In contrast, Pellet is a complete OWL-DL reasoner. It is based on the tableaux algorithms developed for Description Logics and supports the full expressivity of OWL DL including reasoning about nominals. For these reasons we choose Pellet as the middleware to help implement our *T-Parser*. Since Pellet is open source and provides APIs, we can use it directly to parse the given OWL ontology and extract all concept classes.

Pellet also offers ontology reasoning and can be used to find both explicit and implicit relationships in S_{ONT} (discussed in Subsection 2.1) of each concept class with all other concept classes. Next, we need to remove those redundant relations before doing BN translation. Otherwise, the BN structure will be a large scale and contains lots of redundant dependant relations. Before giving the redundancy elimination procedure, note that most of the redundant relations in S_{ONT} are stemmed from the "equivalency", "transitivity", and "commutativity" of logical relations. For example,

- if *equivalent*(A, B) is in S_{ONT}, then for any concept C related to A in S_{ONT}, C is also related to B of the same relation in S_{ONT};
- if $A \subseteq B$ and $B \subseteq C$ are in S_{ONT} then $A \subseteq C$ is also in S_{ONT};
- if $A \subseteq B$ and $B \cap C = \varnothing$ are in S_{ONT} then $A \cap C = \varnothing$ is also in S_{ONT};
- if $A = B \cap C$ is in S_{ONT} then $A = C \cap B$ is also in S_{ONT};

To remove redundant relations, we have implemented the following procedure:

1. **Equivalence**:
   1.1 partition the set of all concept classes into equivalence groups, and designate one concept in each group as its representative;
   1.2 replace all equivalence relations in S_{ONT} by these equivalence groups;
   1.3 for all remaining relations in S_{ONT}, replace each concept by the representative of the equivalence group it belongs to;
   1.4 combine relations that are the same or become the same under commutation into one relation;
2. **Disjoint**: remove $A \cap B = \varnothing$ from S_{ONT} if $C \cap D = \varnothing$, $A \subseteq C$ and $B \subseteq D$ are also in S_{ONT};
3. **Subclass**: for each concept class A
   3.1 identify all most specific subsumers of A;
   3.2 remove all $A \subseteq C$ from S_{ONT} if C is not a most specific subsumer of A;

In Section 4, we use the "Wine" ontology [25] to verify our redundancy reduction procedure.

3.2.2 P-Parser

Before we implement *P-Parser*, probabilistic information should be first encoded in specific file as we mentioned in section 2.2. The encoding described in this subsection is a generalization of what we have proposed in [3]. In this encoding convention we treat a probability as a kind of resource, and define several OWL classes to encode probabilities:

- **Class Variable**: its instances denote variables (nodes) in the translated BN. A variable has a property called "hasClass", pointing to the concept class in the original ontology where this variable is mapped from.
- **Class Proposition**: its instances denote variable instantiations. A proposition has two properties: "hasVariable" and "hasState", indicating the variable and the state the proposition is instantiated.
- **Class Probability**: its instances denote individual probabilities. A probability has three properties: "hasProposition" (cardinality >= 1), "hasCondition" (cardinality >= 0) and "hasValue" (cardinality = 1).

Using these classes we can easily define marginal and conditional probabilities without ambiguity. We illustrate this convention by an example that encodes the probability of 'Male' and 'Human' are true given 'Animal' is true, e.g. P(Male, Human | Animal) = 0.511.

Implementation of *P-Parser* is based on the encoding rules, e.g. when markup 'Proposition' is found, an entry of a constraint will be filled by its value. Also the *P-Parser* will check the consistency of probability files. This includes variable consistency (each variable must correspond to a concept class); value consistency (each value should be between 0 and 1, and the sum of values belonging to one constraint should be 1), etc.

```
<owl:Variable rdf:ID="male">
   <hasClass>Male</hasClass>
</owl:Variable>

<owl:Variable rdf:ID="human">
   <hasClass>Human</hasClass>
</owl:Variable>

<owl:Variable rdf:ID="animal">
   <hasClass>Animal</hasClass>
</owl:Variable>

<owl:Proposition rdf:ID="m1">
   <hasVariable>male</hasVariable>
   <hasState>True</hasState>
</owl:Proposition>

<owl:Proposition rdf:ID="h1">
   <hasVariable>human</hasVariable>
   <hasState>True</hasState>
</owl:Proposition>

<owl:Proposition rdf:ID="a1">
   <hasVariable>animal</hasVariable>
   <hasState>True</hasState>
</owl:Proposition>

<owl:Probability rdf:ID="P(m1,h1|a1)">
   <hasPoposition>m1</hasPoposition>
   <hasPoposition>h1</hasPoposition>
   <hasCondition>a1</hasCondition>
   <hasValue>0.511</hasValue>
</owl:Probability>
```

3.3 BN Constructor

Component *BN Constructor* implements the set of structural translation rules given in Subsection 2.1 to build a BN DAG for the taxonomy extracted from the given OWL ontology. It first generates one concept node in the BN for each concept class defined in the ontology using Rule 1. Then, the subclass relationship defined in the OWL ontology will be retrieved and direct subclass will be connected by links from its parents using Rule 2. Finally the *BN Constructor* retrieves all other logical relationships among concept classes, and creates an L-Node for each relationship. The subnets with respect to the L-Nodes and their respective CPTs are also generated in the BN structure according to Rule 3.

After the BN structure is created, the CPTs for concept nodes are initialized as discussed in Subsection 2.3.

3.4 CPT Constructor for Concept Nodes

The remaining work is how to revise BN's initial CPTs of each concept node to integrate given probabilistic information. This is done by the *CPT Constructor*.

Using the IPFP-based algorithms discussed in Section 2.3, *CPT Constructor* first takes the probability constraints generated from the *P-Parser* as inputs and then iteratively modifies the CPTs by E-IPFP (or D-IPFP if each constraint is within one CPT) using those constraints. Finally, when E-IPFP converges, the resulting BN integrating probabilistic information will be provided as the system output.

A tool of BN, Netica, is used when implementing CPT Constructor. Netica is a Bayesian network development software system from Norsys Software Corporation [23]. It provides a set of APIs for BN reasoning and is used in algorithm E-IPFP.

3.5 BayesOWL 1.0 APIs

Figure 3.2 shows the list of BayesOWL 1.0 APIs, which are contained in several Java packages. The package "commonDefine" contains classes defining data structure such as joint probability distribution, etc. The package "commonMethod" contains a list of operations for the defined data structures. IPFP based algorithms are packed in the package "coreAlgorithms". Both *T-Parser* and *P-Parser* are defined in the package "parser". The package "constructor" consists of *BN structure constructor* and *CPT constructor*. Finally, the package "GUI" implements the system's Graphical User Interface. All of these packages work together to complete the Ontology to BN translation. Each of these packages can also be used separately.

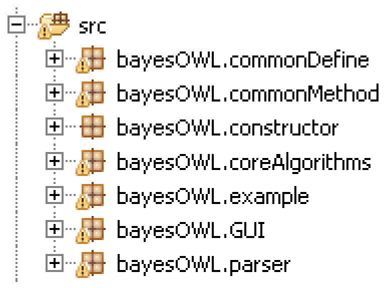

src
bayesOWL.commonDefine
bayesOWL.commonMethod
bayesOWL.constructor
bayesOWL.coreAlgorithms
bayesOWL.example
bayesOWL.GUI
bayesOWL.parser

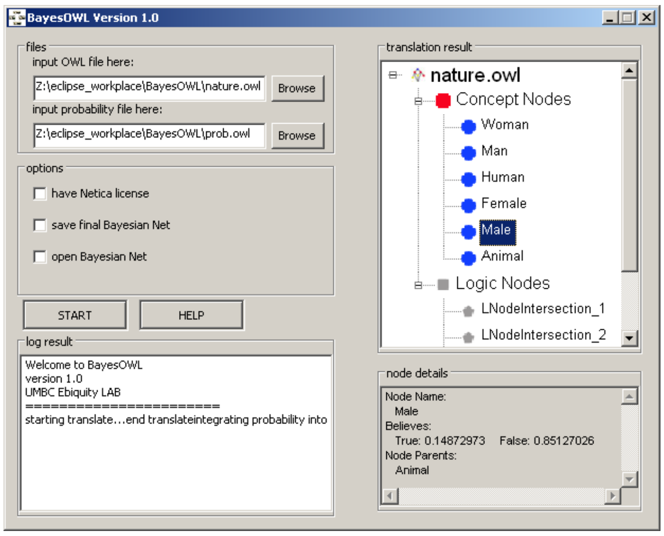
Figure 3.2 BayesOWL API

3.6 BayesOWL GUI

The system GUI is given in Figure 3.3. The layout is divided into several areas:

- **File input**: used to input OWL ontology files and probability files;
- **Options**: designed for optional operations such as requesting Netica license for large BN, the location that the resulting BN is to be saved, and whether the user want to open and view the resulting BN when it is generated;
- **Log area**: used to show the running status;
- **Resulting BN**: shows the list of concept nodes and the list of L-Nodes of the translated BN; and
- **Node detail**: gives detailed information of a node selected in the Resulting BN Area, including its prior beliefs and its parents.

The BayesOWL GUI is executable. After the input ontology and probability files are specified, the "start" button starts the translation, the resulting BN will be generated and saved, and the network structure is shown in the translation result area.



```
input OWL file here:
Z:\eclipse_workplace\BayesOWL\nature.owl
input probability file here:
Z:\eclipse_workplace\BayesOWL\prob.owl

options
☐ have Netica license
☐ save final Bayesian Net
☐ open Bayesian Net

log result
Welcome to BayesOWL
version 1.0
UMBC Ebiquity LAB
========================
starting translate...end translateintegrating probability into

translation result
nature.owl
  Concept Nodes
    Woman
    Man
    Human
    Female
    Male
    Animal
  Logic Nodes
    LNodeIntersection_1
    LNodeIntersection_2

node details
Node Name:
  Male
Believes:
  True: 0.14872973    False: 0.85127026
Node Parents:
  Animal
```

Figure 3.3 GUI of BayesOWL 1.0

4 USE CASE AND DISCUSSIONS

We demonstrate the use of BayesOWL 1.0 by a simple example ontology called "nature", taken from [4]. This ontology defines the following six concept classes and several logical relations among these concepts:

- "Animal" is a primitive concept class;

- “Male”, “Female” and “Human” are subclasses of “Animal”;
- “Man” and “Woman” are two subclasses of “Human”;
- “Male” and “Female” are disjoint with each other;
- “Man” is an intersection of “Human” and “Male”;
- “Woman” is an intersection of “Human” and “Female”;
- “Human” is the union of “Man” and “Woman”.

Figure 4.1 gives the BN structure translated from this ontology. It contains six concept nodes, one per each concept class, together with the directed links for the defined subclass relations. The BN also contains four L-Nodes for the four defined logical relations, together with the proper links as dictated by the structure translation rules for these logical relations. All nodes’ CPTs are initialized using rules discussed in Section 2.3.

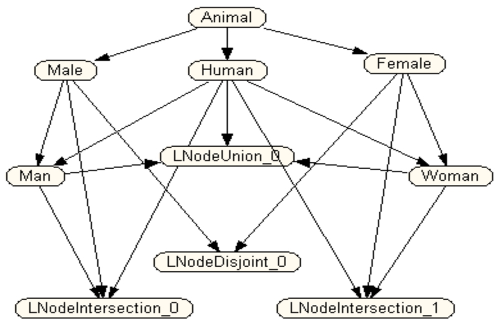


Figure 4.1 Translated BN structure from the “nature” ontology

We have provided a set of probabilistic constraints as follows:

- P(Animal) = 0.56
- P(Male,Human|Animal)= 0.51
- P(Female,Human|Animal)= 0.26
- P(Man|Animal,Human)= 0.66
- P(Woman|Animal,Human)= 0.34

As can be seen from these constraints, probabilities are more general than we have used in original framework as each involves multiple variables in different CPTs.

After running BayesOWL 1.0, the translated BN and its final CPTs are given in

Figures 4.2 and 4.3 below. It can be seen that, after all logic nodes are set to "True", the network is consistent with all the probability constraints.

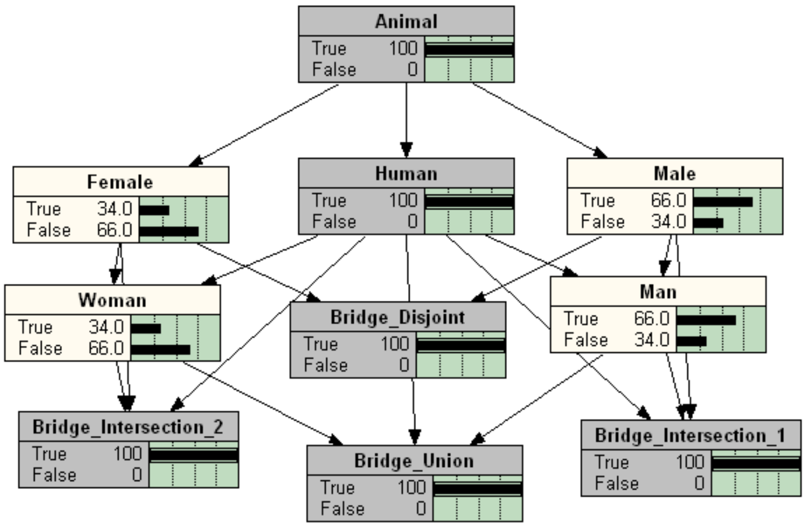
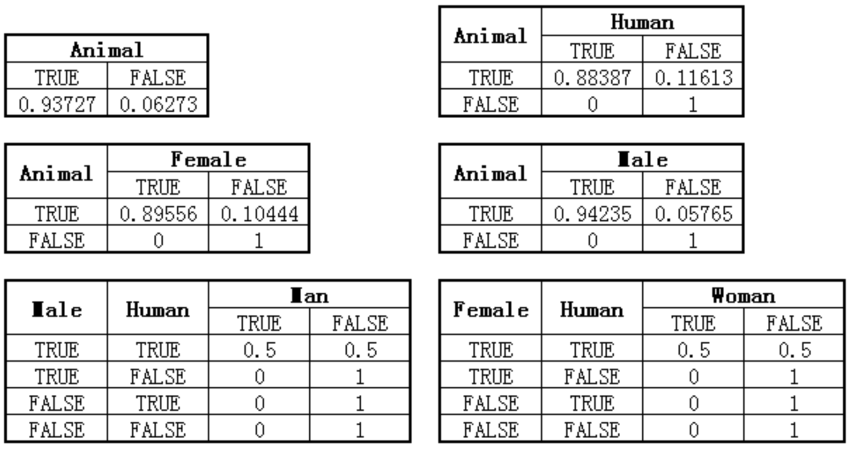

Animal True 100 False 0
Female True 34.0 False 66.0
Human True 100 False 0
Male True 66.0 False 34.0
Woman True 34.0 False 66.0
Bridge_Disjoint True 100 False 0
Man True 66.0 False 34.0
Bridge_Intersection_2 True 100 False 0
Bridge_Union True 100 False 0
Bridge_Intersection_1 True 100 False 0

Figure 4.2 Translated BN of the "nature" ontology



Animal	
TRUE	FALSE
0.93727	0.06273

Animal	Human	
	TRUE	FALSE
TRUE	0.88387	0.11613
FALSE	0	1

Animal	Female	
	TRUE	FALSE
TRUE	0.89556	0.10444
FALSE	0	1

Animal	Male	
	TRUE	FALSE
TRUE	0.94235	0.05765
FALSE	0	1

Male	Human	Man	
		TRUE	FALSE
TRUE	TRUE	0.5	0.5
TRUE	FALSE	0	1
FALSE	TRUE	0	1
FALSE	FALSE	0	1

Female	Human	Woman	
		TRUE	FALSE
TRUE	TRUE	0.5	0.5
TRUE	FALSE	0	1
FALSE	TRUE	0	1
FALSE	FALSE	0	1

Figure 4.3 Final CPTs of BN for "nature" ontology

We have also applied BayesOwl 1.0 on the "Wine" ontology [25], a domain ontology built by W3C for semantic web. The result is as follows:

Table 4.1 Number of relations in wine ontology

	Explicitly Defined	Derived by Pellet	After Redundant Reduction
No. of Relations	99	6641	**388**

As given in Table 4.1, the number of logical relations explicitly defined between concept classes in the ontology file is 99. This number is much smaller than the total number of relations derived by Pellet reasoner which is 6641. After applying our redundant relation reduction procedure, the number is reduced to 388, which is less than 6% of all derivable logical relations. Each of these relations is represented by an L-node in the translated BN. The total number of the concept classes in wine ontology, including those in the “Food” ontology Wine imports, is 126, and the total number of the nodes (the concept nodes plus the L-nodes) in resulting BN is 342.

5 CONCLUSIONS

In this project, a prototype system BayesOWL 1.0 is developed to implement a probabilistic framework for uncertainty reasoning in semantic web ontologies based on Bayesian networks. The implementation extended the original framework from simple terminological taxonomy ontologies to general OWL DL ontologies by using existing OWL reasoner tool to deal with logical relations implicitly defined via anonymous classes. A more general convention is proposed and adopted to encode probabilistic information as general marginals and conditionals. These two extensions are significant advancement of the original framework.

BayesOWL 1.0 provides a set of APIs, including algorithms for structural translation from OWL ontologies to BN DAG and for incorporating probabilistic constraints into the BN CPTs. A graphical user interface is also implemented to facilitate the use of the system. Experiments show that the system runs well on OWL taxonomies of different size.

As a software tool, the BayesOWL system can be used by researchers and practitioners on ontology engineering such as domain modeling, ontology reasoning and ontology concept mapping.

Further improvement of this system includes extending the framework from consistent probabilistic constraints to inconsistent constraints. Several proposals have been made to modify a joint distribution with inconsistent constraints [14][16]. How to learn uncertainty constraints from existing resources automatically also needs to be investigated. Finally, the framework and the implementation will be generalized to include the properties and individuals defined in the OWL ontologies as well.

REFERENCES

[1] Deming WE, Stephan FF (1940) On a Least Square Adjustment of a Sampled Frequency Table when the Expected Marginal Totals are Known. Ann. Math. Statist. 11:427-444

[2] Ding, Z., Peng, Y. et al.: "A Bayesian Approach to Uncertainty Modeling in OWL Ontology", *Proceedings of the International Conference on Advances in Intelligent Systems Theory and Applications*, November 2004, Luxembourg.

[3] Ding, Z. and Peng, Y: "A probabilistic extension to the web ontology language OWL", *Thirty-Seventh Hawaii International Conference on System Sciences (HICSS-37)*, Big Island, Hawaii, Jan. 5 – 8, 2004.

[4] Ding, Z., "BayesOWL: Uncertainty Modeling in Semantic Web Ontologies", PhD thesis, 2005.

[5] Ding, Z., Peng, Y., Pan, R., and Yu, Y: "A Bayesian Methodology Towards Automatic Ontology Mapping", *AAAI-05 Workshop on Contexts and Ontologies: Theory, Practice and Applications (C&O-2005)*, Pittsburgh, PA, July 9, 2005.

[6] Ding, Z., Peng, Y. and Pan, R: "BayesOWL: Uncertainty Modeling in Semantic Web Ontologies", in *Soft Computing in Ontologies and Semantic Web*, Springer-Verlag, March 2006.

[7] Fukushige Y: "Representing Probabilistic Knowledge in the Semantic Web", position paper in *The W3C Workshop on Semantic Web for Life Sciences*, Cambridge, MA, USA, 2004.

[8] Holi M, HyvÄonen E.: "Probabilistic Information Retrieval based on Conceptual Overlap in Semantic Web Ontologies", in *Proceedings of the 11th Finnish AI Conference, Web Intelligence*, Vol. 2. Finnish AI Society, Finland, 2004.

[9] Koller D, Levy A, Pfeffer A.: "P-CLASSIC: A Tractable Probabilistic Description Logic", in *Proceedings of AAAI-97*, 390-397, 1997.

[10] Pan, R., Ding, Z., Yu, Y. and Peng, Y.: "A Bayesian Network Approach to Ontology Mapping", in *Proceedings of the Fourth International Semantic Web Conference (ISWC 2005)*, Galway, Ireland, Nov. 6-10, 2005.

[11] Pearl J.: *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufman, San Mateo, CA, 1988.

[12] Pearl J, Fusion, *Propagation and Structuring in Belief Networks*. Artificial Intelligence 29:241-248, 1986.

[13] Peng, Y. and Ding, Z.: "Modifying Bayesian Networks by Probability Constraints", in *Proceedings of 21st Conference on Uncertainty in Artificial Intelligence (UAI-2005)*, Edinburgh, Scotland, July 26-29, 2005.

[14] Vomlel J.: *Methods of Probabilistic Knowledge Integration*. PhD Thesis, Department of Cybernetics, Faculty of Electrical Engineering, Czech Technical University, 1999.

[15] Yelland PM.: *Market Analysis Using Combination of Bayesian Networks and Description Logics*, Sun Microsystems Technical Report TR-99-78, 1999.

[16] Zhang, S. and Peng, Y: “An Efficient Method for Probabilistic Knowledge Integration”, in *Proceedings of The 20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI-2008)*, Dayton, Ohio, Nov. 3-5, 2008.
[17] http://en.wikipedia.org/wiki/Semantic_Web
[18] http://www.w3.org/TR/owl-features/
[19] http://www.w3.org/RDF/
[20] http://en.wikipedia.org/wiki/Semantic_reasoner
[21] http://www.w3.org/2005/Incubator/urw3/XGR-urw3-20080331/
[22] http://clarkparsia.com/pellet
[23] http://www.norsys.com/
[24] http://jena.sourceforge.net/
[25] http://www.w3.org/TR/2003/CR-owl-guide-20030818/wine