Managing Uncertainty in a Business-Rule System

Hassan Aït-Kaci [1] and Philippe Bonnard [2]

[1] IBM Canada Ltd.—hak@ca.ibm.com
[2] IBM France—pbonnard@fr.ibm.com

Abstract. This work investigates effective ways to integrate probabilistic reasoning in an object-oriented Business Rule Management System (BRMS) based on "*Production Rules*" (also known as "*Condition/Action Rules*") and Bayesian Network technology. It is organized as an informal overview of, and motivation for, design features that would be required to make a real industrial BRMS such as IBM WODM JRules capable of handling uncertainty in the conditions and actions of rules. It is presented from the perspective of a user, motivating and illustrating each issue by way of examples. Thus, we need to interpret objects with random-valued attributes, manage class/instance relationships, control probabilistic activation thresholds to rules, specify stateful *vs.*. stateless probability evaluation, adapt working memory modification, when to reset probabilistic variables, and handle Boolean logic with unknown values.[3]

KEYWORDS: Business Rule Management System; Graphical Models; Bayesian Networks; Production Rules; Rule-Based Computing; Probabilistic Computing; Adaptive Computing.

1 Introduction

Motivation—We propose to explore a technology to support probabilistic condition/-action rules. Our objective is to provide a means to support more realistic decision making by allowing rules to adapt to naturally arising variations in data. As importantly, this technology is the key to enable learning decision rules from causal data, sequential data (time series), and empirical decision tables. With such capabilities, a Business Rule Management System (BRMS) such as IBM WODM[4] JRules can make rule-based decision-making greatly more realistic as it would allow managing uncertainty.

The central issue to be addressed is how to make the Rete technology work for such probabilistic rules [2].[5] We believe that all the pieces needed to resolve this issue are now available. Indeed, research and practice in probabilistic inference and generalization technology has made great strides in the past decade [3]. In particular, it is well-known that decision-making in many important Business Rules application areas relies crucially on statistical and/or time series data (*e.g.*, health care, re/insurance, credit rating, financial trading, *etc.*, ...). Finally, being able to deal with uncertainty makes it possible to *learn* rules from data rather than experts, thus rendering decision-making more reliable.[6]

[3] Draft of April 11, 2012. This paper is extracted from [1].
[4] WODM stands for Websphere Operational Decision Management.
[5] Industrial BRMSs have adapted the LISP-based formulation to an object-oriented Rete.
[6] It is to be noted that deriving rules based on decision tables is a frequent request from JRules customers [4].

Relation to other work—There have been previous attempts to mix some form of probabilistic logic with production rules. Indeed, one of the earliest expert systems proposed in the 1980's used a form of odds-abiding rule logic (*viz.*, Mycin [5]). More recently, work such as [6,7] even gives details on how probabilities may be taken into account in the Rete algorithm [2]. Yet, all this work is still far from being standard technology as today's main BRMS vendors still do not support probabilistic reasoning.

Still, the question that must be in the reader's mind is: "*How is this any different from all that has been tried before in the field?*" Indeed, probabilistic Production Rules have been used since some of the earliest AI research.

Mycin [8,5] was one of the original production-rule systems in AI experimenting with enhancing its decision-making power using probabilities. Because its domain of application was medical diagnosis, it was deemed appropriate to draw conclusions from evidence and measures of medical parameters that would estimate a probability weight for the conclusion. However, Mycin's use of probabilities was, contrary to what we propose, not associated to the condition part of C/A rule, but only to its conclusion (a medical diagnosis). For example, "*if the patient has digestive problems, and if this virus is present, and if the patient's body temperature is high, then there is a .85 probability that the patient has gastric flu.*" Such facts asserted with some probabilistic value, when used in conditions of other rules were then combined using the smaller one when conjoined, or the greater one when part of a disjunction. This was not only arbitrary a "logic," but also not at all related nor justified by Bayesian Reasoning logic since probabilistic facts were in no way correlated though Bayesian analysis.

Independently of the important fact that the technologies underlying both C/A-rule systems and probabilistic decision-making have made drastic progress since the time when those systems and other academic prototypes were experimented with, our most pertinent motivation may be explained as follows:

1. both C/A rules (in the form of BRMSs) and Bayesian decision-making, since their earlier manifestation mostly in an academic context, have now reached industrial quality exploitable on the market place, and designed to work in modern popular object-oriented systems (Java, C#, C++, *etc.*, . . .);
2. industrial tools need now to be interfaced, or made to work, seamlessly with other commonly used industrial-quality software development environments;
3. there is no existing industrial-quality combination of the best existing technologies in both fields offering a software that can support a seamless, clean, well-integrated, and efficient probabilistic production-rule decision making environment.

In addition, early probabilistic expert systems ever experimented with have been far from offering as diverse a means to express uncertainty in production rules as what we need to achieve.

Recently, the issue of integrating probabilistic information in the standard database relational model has been addressed. One specific model has emerged: the Probabilistic Relational Model (PRM) [9] and its object-oriented extension OOPRM [10]. In addition to being relevant for decision making under uncertainty, PRMs are a particular instance of Probabilistic-Logical Models, which have also been shown useful for machine learning.

As was the case for C/A rules, reconciling the PRM-based decision making paradigm with object-oriented programming has necessitated developing new basic structural and procedural compiling methods [11,10]. However, this work has yet to be tested in an industrial context, as well as the relatively recent development of clever analytical tools and algorithms in uncertainty management in data and decision making [12,13,3]. Indeed, most, if not all, this cited work has been carried out in academia, and not really tested in an industrial context. Decision making under uncertainty has yet to be transferred into "real" business decision assistance products such as IBM's WODM.

Organization of contents—The rest of this paper is organized as follows. Section 2 discusses how our objective summed up above may be articulated into a BRMS such as IBM WODM JRules from the standpoint of a user. Section 2.1 justifies the natural need for managing uncertainty in a BRMS. We essentially model Probabilistic Business Objects following the well-known Probabilistic Relational Model (PRM). Section 2.2 explains the issue of how to handle probabilistic attributes. Section 2.3 shows how to reconcile actual object-oriented structures classes and objects using the PRM. Section 2.4 extends probabilistic control to specify activation thresholds to rules. Section 2.5 discusses how probabilities may be evaluated depending on whether or not the state of the working memory is taken into account. Section 2.6 addresses the general issue of the need to manage operations invoked on uncertain variables. Section 2.7 considers what it means to modify the working memory when it may be used to assert prior probabilities on some probabilistic facts. Section 2.8 explains the pragmatics in a "real" BRMS to support resetting random attributes. Section 2.9 justifies the need for extending Boolean operations with unknown values. Finally, Section 3 concludes with a few perspectives.

2 Probabilistic rule-based decision making

This section explains how the software underlying a BRMS such as IBM WODM JRules may be instrumentalized to accommodate probabilistic reasoning. We describe a set of ways to combine Business Rules and probabilistic data-processing, taking advantage of Bayesian technology [3,14,15,16,17].

Business Rule (BR) engines are usually based on *Production Rules* (PRs), which are Condition/Action rules. The condition part of such a traditional rule uses the classical Boolean connectives (**and**, **or**, **not**). This is sufficient to support decision-making from facts that are certain. However, when either the facts and/or the rules' conditions are uncertain (which is the case in most realistic situations) the traditional model becomes inadequate. Thus, the problem is then to determine how traditional PRs can be made to support decision-making under uncertainty.

We propose to extend the traditional PR-based computation model with a simple and intuitive enhancement of the notion of Production Rule into a Probabilistic Production Rule (PPR). This can be made possible by using Bayesian Network technology. This technology has the advantage of enabling an operationally effective means to account for decision-making under uncertainty, all the while keeping all the benefits of Rete-based pattern-matching [2].

There are two possibilities for PRs to account for uncertainty: (1) see data as probabilistic (*e.g.*, data categorization); (2) see rules as probabilistic (*e.g.*, uncertain deci-

sions).[7] In the first case, we must adapt Rete-style pattern-matching [2] to account for uncertain data. In the second, we need to take into account application of rules of the form: "**if** (*condition*) **then** *action* **with probability** *alpha*." The informal architecture we describe below offers an effective operational means to achieve both.

2.1 Using probabilistic information inside rules

PR systems are usually based on Boolean Logic. A decision is made to fire a rule and take the action it guards whenever a specific Boolean condition application of the rule is satisfied by a set of data. For example, the following PR (expressed in JRules) may be used to plan an appointment for a patient with a specialist when a cancer has been detected.[8]

```
rule routePatientToSpecialist {
  when { patient: Patient ( cancer );
         doctor: Doctor ( isCancerSpecialist();
                          isInSameCountyAs( patient ) );
  }
  then { planAppointement ( patient, doctor );
  }
}
```

Imagine now that, at this level of the diagnostic, the cancer has not been effectively detected although a risk of cancer is somehow estimated, given the characteristics of the patient. If the foregoing rule may be made to take this uncertainty into consideration, it should be modified in such a way as to take care of that risk and refer the patient to a specialist for further testing only when a conditional probability threshold is exceeded (say, `0.3`).

For example:

```
rule referPatientToSpecialist {
  when { patient: Patient
                  ( prob( cancer
                        | age == patient.age;
                          smoker == patient.smoker ) > 0.3 );
         doctor: Doctor ( isCancerSpecialist();
                          isInSameCountyAs( patient ) );
  }
  then { planAppointement ( patient, doctor );
  }
}
```

Then, we can specify in a second rule under what circumstances the patient is to be referred to a general practitioner in cases where the risk is low.

[7] These two possibilities are illustrated on examples below.

[8] The use of the "vertical bar" character "`|`" for "*knowing that*" is more familiar to users dealing with probabilities. Using it inside a "`prob`" expression is not ambiguous. At any rate, the syntax we use in this paper's examples is in no way definite and may be subject to change.

```
rule referPatientToGP {
  when { patient: Patient
                  ( prob( cancer
                        | age == patient.age;
                          smoker == patient.smoker ) < 0.1 );
         doctor: Doctor ( isGP();
                          isInSameCountyAs( patient ) );
  }
  then { planAppointement ( patient, doctor );
  }
}
```

In that case, we have introduced the random variables `cancer`, `age` and `smoker` participating in a probabilistic model of a patient case. A conditional probability operator, `prob`, is introduced to perform the effective evaluation of the risk. What is important here to notice is the mixed use of object data (such as the `isSameTownAs` method), and probabilistic variables (such as `patient`).

2.2 Mapping random variables into classe attributes

An improvement of the previous version maps directly the random variables of the model into the object model. In so doing, we follow the Probabilistic Relational Model (PRM) extension of Bayesian Networks [18]. Thus, the patient routing rule is then rewritten into something easier to read:

```
rule referPatientToSpecialist {
  when { patient: Patient
                  ( prob( cancer | age; smoker ) > 0.3);
         doctor: Doctor ( isCancerSpecialist();
                          isInSameCountyAs( patient ) );
  }
  then { planAppointement ( patient, doctor );
  }
}
```

Observe that now, `cancer`, `age` and `smoker` are no longer *variables* but *attributes* of the `Patient` class. The PRM mapping associates to them a probabilistic model with conditional distributions [18].[9] Note that those attributes are not exactly instance attributes as in a regular object model. Note that those attributes are no longer simple instance attributes as in a regular object model. Indeed, they may *not carry an actual value at run-time*—in other words, their value may simply be *unknown* (*i.e.*, $\perp$ formally).

Another advantage is the capability of probabilistic network generation. Each instance of the `Patient` class is linked to a distinct network, where each random variable is mapped into an instance attribute. Generating a new `patient` network requires only instantiating a new `Patient` object.

[9] See Figures 1, 2, and 3 for examples of such class and influence diagrams.

2.3 Using relation between classes and instances

The PRM goes beyond the simple attribute mapping presented in the previous section. Indeed, by completing the mapping between classes and relations, it enables defining probabilistic network overlapping several instances. In such models, probabilistic dependencies between attributes of different classes and instances are possible. Consequently, the PRM takes advantage of the object-oriented model for representing a set of classes with possible navigation between them. While this makes it easy to generate complex probabilistic networks, it also complicates evaluation of their probabilities [19,20].

In our case study, we introduce in the model a set of complementary analyses prescribed to a patient. The new strategy is to refer a patient to a specialist if the risk is deemed sufficiently high knowing the results of a gamma analysis performed on the patient.

```
rule referPatientToSpecialist {
  when { patient: Patient ( );
         analysis: CEAAnalysis ( ) in patient.analysisList;
         evaluate ( prob( cancer
                        | age; smoker; analysis.value ) > 0.3 )
         doctor: Doctor ( isCancerSpecialist();
                          isInSameCountyAs( patient ) );
  }
  then { planAppointement ( patient, doctor );
  }
}
```

In this rule, the random variable `analysis.value` denotes an attribute of the class `CEAAnalysis`.[10] It is connected to a `Patient`'s instance and its probabilisitc attributes by navigating through the `Patient.analysisList` reference from the patient instance.

Furthermore, this reference concept brings the opportunity to define complex aggregation variables over a collection (Figure 1 illustrates this for our example.) In fact, the PRM makes use of the most common aggregation funtions: `exists`, `forall`, `mean`, `min`, `max`, *etc.*, ... These functions can thus be used in the model for defining new probabilisitic variables, or directly in the rules by ways of conditions involving aggregations [19,21]. Figures 2 and 3 illustrate influence diagrams for our example, with and without aggregation—where the influence of the random variables A_n.value is aggregated using "$\bigoplus$" as a generic aggregation symbol standing for any aggregation functions such as listed above.

2.4 Adding probabilistic activation thresholds to rules

Note that the probabilistic operator `prob` can be used explictly in a rule's condition as shown the in previous section. We now propose to remove it from the rules for simplicity's sake. To achieve that, we introduce a new type of rule: a *probabilisitic production*

[10] "CEA" stands for *"CarcinoEmbryonic Antigen."*

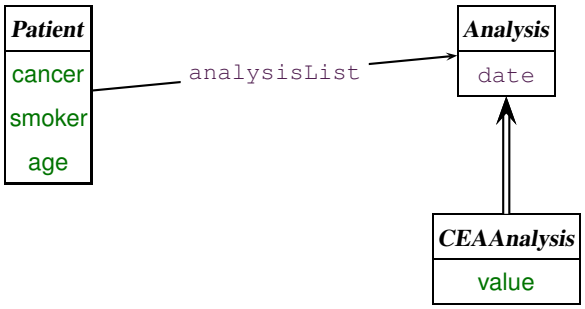

Fig. 1. Example of Class Navigation Diagram

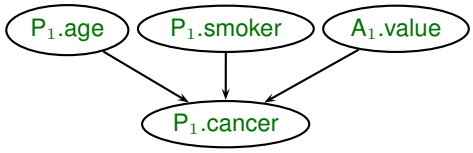

Fig. 2. Example of Bayesian Influence

rule. This sort of rule is simply obtained as a production rule with an additional probabilistic activation threshold. Such a rule's probability threshold helps to determine when a PPR is eligible (*i.e.*, potentially fireable). By definition, a PPR is eligible against a tuple when the probability of the tuple pattern matching equals or exceeds the probabilistic threshold.

Consider now the following rule where the probability rule property defines this threshold:

```
rule referPatientToSpecialist {
  probability >= 0.3;
  when { patient: Patient ( cancer );
         doctor: Doctor ( isCancerSpecialist();
                          isInSameCountyAs( patient ) );
  }
  then { planAppointement ( patient, doctor );
  }
}
```

This rule means that if the probability of having a cancer is higher that 30%, then the patient should be referred to a specialist.[11]

[11] Note that this notation may look ambiguous when it is not. Indeed, having the probability at the outset before the condition means that it applies to *all* but *only* the random attributes *mentioned* in the condition. It does not concern other unmentioned random attributes of an object's class.

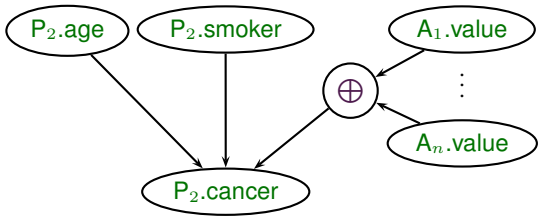

Fig. 3. Example of Aggregative Bayesian Influence

Another example means that a patient with a cancer risk lower that 10% is referred to a general practitioner.

```
rule referPatientToGP {
  probability <= 0.1;
  when { patient: Patient ( cancer );
         doctor: Doctor ( isGP();
                          isInSameCountyAs( patient ) );
  }
  then { planAppointement ( patient, doctor );
  }
}
```

It may be necessary in some complex situation to indicate which specific rule conditions will participate to the whole probability calculation. For example, the foregoing rule would then be rewritten as:

```
rule referPatientToGP {
  when { patient: Patient ( prob (cancer) < 0.1 );
         doctor: Doctor ( isGP();
                          isInSameCountyAs( patient ) );
  }
  then { planAppointement ( patient, doctor );
  }
}
```

Hence, the probability calculation is limited to the random variable `cancer`.

2.5 Stateful *vs.* stateless probability evaluator

The probability operator `prob` has the same meaning as the standard mathematical operator $P(\ldots)$ denoting the probability of a random event. This means that the expression `prob ( cancer | age; smoker )` denotes the probability of having a cancer, given the age and the smoking habits of a patient. The conditionnal context is explicitly written.

The situation is somehow different when we use the notion of PPR. Intuitively, what is relevant when we define the following rule:

```
rule referPatientToSpecialist {
  probability = 0.3;
  when { patient: Patient ( cancer );
         doctor: Doctor ( isCancerSpecialist();
                          isInSameCountyAs( patient ) );
  }
  then { planAppointement ( patient, doctor );
  }
}
```

is that, given everything we know about the patient, the probability of having a cancer of greater than 0.3. In that case, the conditional context of every facts of the WM is important but not explicitly written, for practical reason, since we don't care what has been posted previously in the working memory.

The `prob` operator presented above actually should be stateful for it to be easily used in a rule without knowing the underlying random variable model. In that perspective, `prob(cancer|age,smoker)` could be simply rewritten as `prob(cancer)`. Thus, the previous rule are simply rewritten as:

```
rule referPatientToSpecialist {
  when { patient: Patient ( prob( cancer ) > 0.3 );
         doctor: Doctor ( isCancerSpecialist();
                          isInSameCountyAs( patient ) );
  }
  then { planAppointement ( patient, doctor );
  }
}
```

The expression `prob(cancer)` means implicitly $P(\texttt{cancer} \mid \texttt{WM})$, which is indeed "reduced" in our model to $P(\texttt{cancer} \mid \texttt{age}, \texttt{smoker})$.

In order to allow distinguished contexts in which a probability may be interpreted, we introduce the concept of *world*. Essentially, a world encompasses every probabilisitc known fact in the WM. By default, there are two basic worlds: (1) `WM`, the stateful world—*i.e.*, the working memory itself (implicit by default), and (2) `STATELESS`, the stateless world containing no fact at all—*i.e.*, the empty world.

Specifying world context can be done by using a second argument to the `prob` operator. The optional condition part of the operator still exists and maybe combined with an explicit world selection.

In that meaning:

$$\text{(1)}\quad \begin{cases} \texttt{prob(cancer)} = \texttt{prob(cancer,WM)} = P(\texttt{cancer} \mid \texttt{WM}), \\ \texttt{prob(cancer | age>50)} = P(\texttt{cancer} \mid \texttt{age} > 50, \texttt{WM}), \\ \texttt{prob(cancer,STATELESS)} = P(\texttt{cancer}). \end{cases}$$

More generally, if `R` is a random attribute and `(a,b,...)` is a tuple satisfying its condition part $Q(a,b,\ldots)$, then:

$$\text{(2)}\quad \texttt{prob(R;(a,b,...))} = P(Q(a,b,\ldots) \mid \texttt{WM}).$$

Finally, other worlds could be defined in order to extend and control the hypothetical reasoning of the rule engine.

2.6 Known/Unknown operator over attributes

When an attribute is mapped into a random variable, its value is linked to a probability distribution. Hence, this attribute is not guaranteed to hold always a unique value at any given time as are usual object attributes. Therefore, when the actual value of a random attribute is not known, some instructions, such as operators or methods expecting this attribute as argument, may not be carelessly invoked on this attribute. A special `isKnown` operator is introduced that determines whether such an attribute has a value or not.

Let us take for example the following rule, supposing that `analysis.value` is linked to a random variable:

```
rule referPatientToSpecialist {
  probability > 0.7;
  when { patient: Patient ( );
         analysis: CEAAnalysis ( analysis.value  > 0.3 )
                   in patient.analysisList;
         doctor: Doctor ( isCancerSpecialist();
                          isInSameCountyAs( patient ) );
  }
  then { planAppointement ( patient, doctor,
                            new HighCEAAnalysis(analysis.value) );
  }
}
```

The meaning of this rule is to plan an appointement for a patient when the probability that *his/her CEA analysis value (some indicator measure) exceed .3* is greater than 70%. Observe that the action `planAppointement` is one that now has new parameter of type, say, class `HighCEAAnalisys` requiring that the actual analysis value be actually provided. In order for the action of this rule to be safely executable, it is required that `analysis.value`'s actual value be known. When this is the case, it means that its distribution is set to a unique value with the probability set to `100%`. However, when it is *not* the case that `analysis.value`'s actual value is known, the above rule is clearly unsafe.

To control this situation, the following safer rule may be specified using a guard explicitly testing whether that the rand variable's actual value is known:

```
rule referPatientToSpecialist {
  probability > 0.7
  when { patient: Patient ( );
         analysis: CEAAnalysis ( analysis.value isKnown;
                                 analysis.value > 0.3 )
                   in patient.analysisList;
         doctor: Doctor ( isCancerSpecialist();
                          isInSameCountyAs( patient ) );
  }
  then { planAppointement ( patient, doctor,
                            new HighCEAAnalysis(analysis.value) );
  }
}
```

Note however that such a test is only required depending on the level of static analysis sophistication of the probabilistic rule processing system. Indeed, these `isKnown` tests could be automatically generated for any occurrence of a random attribute in an expression requiring its actual value. Note however that, whether explicit or automatically generated, these "`isKnown`" safety tests are part of the rule's guard.

2.7 Modifying the working memory

Formally, a WM might be seen as a set of facts concerning a set of objects. These facts are properties of the objects that are used by the object-matching engine to calculate which objects satisfy the rules patterns. The execution of the rules potentially may change the state of the WM by inserting, updating, or retracting objects, as well as inserting or retracting facts.

Consider the following rule updating the family risk whenever cancer was diagnosed for a patient's parent:

```
rule setFamilyRisk {
  when { patient: Patient ( );
         exist Patient ( cancer, isParentOf (patient) );
  }
  then { update patient { patient.familyRisk == HIGH; }
  }
}
```

The action part instruction means that the WM is changed since the `familyRisk` has changed. In addition to `update`, the usual `insert`/`delete` can be used to insert a new object into the WM, or retract an object from it.

In order to post facts, we use the operator `assertFact`. Thus, the previous rule may be rewritten as:

```
rule setFamilyRisk {
  when { patient: Patient ( );
         exist Patient ( cancer, isParentOf (patient) );
  }
  then { assertFact( patient.familyRisk == HIGH );
  }
}
```

Indeed, this operator may post complex facts to the WM, such as:

```
assertFact ( patient.age > 21 );
```

or

```
assertFact ( patient1.age > patient2.age );
```

Note that this type of facts are usually not managed by PR engines such as JRules. However, they become handy in a PPR engine for asserting specific prior probabilities for some facts and propagating them into their Bayesian network for updating posterior probabilities of its random variables they are connected to.

By default, the fact are asserted in the common working memory world, following the same logic than the `prob` operator. Nevertheless, other worlds might be chosen, specified by a second parameter of the `assertFact` operator. For example, consider the assertion of the fact `patient.age > 21` in the world named `MY_WORLD`:

```
assertFact ( patient.age > 21, MY_WORLD );
```

The complexity of the asserted fact depends on the capability of the compiler to process it. Some expressions might be rejected by the system or simply ignored.

By asserting the fact f in a world `W` $= \{f_1, \ldots, f_n\}$, we schematically mean that the world's state represented by the set of facts `W` is being changed to the new state represented by the set $\{f, f_1, \ldots, f_n\}$. This is essentially equivalent to saying that the model represented by the world `W` has been updated with a new evidence f. Any reference to world `W` will now take this new fact into account along with all the facts previously making up `W`.

2.8 Resetting the original probabilistic values of variables

When an attribute is mapped into a random variable, its probability distribution will evolve according to the facts posted in the WM. In order to reinitialize the processing, it may be required that this value be reset to its original distribution. A special operator `assertUnknown` is provided to perform this reinitialization.

Consider the following rules that reinitialize some patient's random variables:

```
rule reinitPatient {
  when { patient: Patient ( );
  }
  then { assertUnknown ( patient.cancer );
         assertUnknown ( patient.smoker );
  }
}
```

For all practical purposes, this operator may be extended to be applied to an object so as to initialize every random variables of that object.

2.9 Extending the Boolean logic with unknown values

As could be seen in the previous section, the value of a random attribute is not always known, defined only to be a probability distribution. The semantics of random attribute access must be adapted depending on the kind of the operators or methods that are invoked on it, the location in the rules (*i.e.*, condition or action part), and the parameters of the compilation, in such a way that:

1. an exception is raised as soon one try to access an unknown value (this semantics is recommended for the right part of the rules); and,
2. the Boolean expression is evaluated taking into account a 3-valued logic with the semantics given in Table 1 for the common Boolean operators.

For example, assuming that the patient's age is unknown, if `(smoker == true)` holds, then so does the condition:

x	**not** x
true	false
false	true
$\perp$	$\perp$

x	y	x **or** y	x **and** y
true	true	true	true
true	false	true	false
true	$\perp$	true	$\perp$
false	true	true	false
false	false	false	false
false	$\perp$	$\perp$	false
$\perp$	true	true	$\perp$
$\perp$	false	$\perp$	false
$\perp$	$\perp$	$\perp$	$\perp$

Table 1. 3-Valued Boolean Operators

```
patient.smoker || patient.age > 17;
```

A last extension, fully compatible with the distribution definition, enables evaluating the probability of Boolean expression as a whole. In that case, the previous expression is said to be true if:

```
prob ( patient.smoker || patient.age > 17 ) == 1;
```

The 3-valued logic with `unknown` (*i.e.*, $\perp$) defined by the operations described in Table 1 may then be used when the probability calculations become too complex.

3 Conclusion

In this paper, we have discussed ways to enhance Business Rule Management Systems based on condition/action rules under uncertainty using Bayesian techniques. We have reviewed relevant work with similar pursuits and positioned our approach in the current context. We think that the time is ripe as several "hot" notions and techniques have recently been put forth in decision-making software research—namely, implementation techniques like the Generalized Distributive Law and Bucket Elimination [22,12].

Finally, one can see our efforts trying to to make a Production Rule engine such as JRules able to accommodate uncertainty as a necessary step toward Data and Rule Mining [23]. Investing in data analytics for business decisions is part of a larger effort at IBM.[12] To wit, *"IBM Roars into Business Consulting"*—or so recently wrote *Business Week*'s columnist Steve Hamm. And he adds, *"Its new 4,000-strong Business Analytics & Optimization Services will mine IBM's research and software divisions for innovations."* It is hoped that our efforts will contribute to leverage this potential.

ACKNOWLEDGEMENTS:
The authors wish to thank Jean-Louis Ardoint, Pierre-André Paumelle, and Hugues Citeau for comments.

[12] See also IBM's event proceessing.

References

1. Aït-Kaci, H., Bonnard, P.: Probabilistic production rules. IBM ILOG Technical Report, International Business Machines, Gentilly, France (2012)
2. Forgy, C.L.: RETE: a fast algorithm for the many pattern/many object pattern match problem. In Raeth, P.G., ed.: Expert Systems: A Software Methodology for Modern Applications. IEEE Computer Society Press, Los Alamitos, CA, USA (1990) 324–341
3. Darwiche, A.: Bayesian networks. Communications of the ACM **53**(12) (2010) 80–90
4. Ardoint, J.L., Bonnard, P.: Composite production rules—a better support of business rules implementation. IBM ILOG Technical Report[13], International Business Machines, Gentilly, France (2012)
5. Novak, G.S.: TMYCIN expert system tool. Technical Report AI-TR-87-52, The University of Texas, Austin, Texas, USA (1987)
6. Sottara, D., Mello, P., Proctor, M.: Configurable rete-oo engine for reasoning with different types of imperfect information. IEEE Transactions on Knowledge and Data Engineering **22**(11) (2010)
7. Sottara, D.: Integration of symbolic and connectionist AI techniques in the development of Decision Support Systems applied to biochemical processes. PhD thesis, Department of Electronics, Computer Science and Systems, University of Bologna, Bologna, Italy (2010).
8. Shortliffe, E.H.: Computer-Based Medical Consultations: MYCIN. Elsevier/North Holland, New York, NY, USA (1976)
9. Getoor, L., Friedman, N., Koller, D., Pfeffer, A., Taskar, B.: Probabilistic relational models. In Getoor, L., Taskar, B., eds.: An Introduction to Statistical Relational Learning. MIT Press, Cambridge, MA, USA (2007) 129–174
10. Howard, C., Stumptner, M.: Automated compilation of object-oriented probabilistic relational models. International Journal of Approximate Reasoning **50**(9) (2009) 1369–1398
11. Torti, L., Wuillemin, P.H., Gonzales, C.: Reinforcing the object-oriented aspect of probabilistic relational models. In Myllymäki, P., Roos, T., Jaakkola, T., eds.: Proceedings of the Fifth European Workshop on Probabilistic Graphical Models, Helsinki, Finland, Helsinki Institute for Information Technology, HIIT Publications (2010) 273–280.
12. Dechter, R.: Bucket elimination: A unifying framework for reasoning. Artificial Intelligence **113**(1–2) (1999) 41–85
13. Cowell, R.G., Dawid, A.P., Lauritzen, S.L., Spiegelhalter, D.J.: Probabilistic Networks and Expert Systems—Exact Computational Methods for Bayesian Networks. Springer Science+Business Media, New York, NY, USA (2007)
14. Paskin, M.A.: A short course on graphical models.
15. Murphy, K.: A brief introduction to graphical models and Bayesian networks (1998).
16. Kjærulff, U., Madsen, A.: Probabilistic networks—an introduction to Bayesian networks and influence diagrams (2005).
17. Niculescu, R.S., Mitchell, T.M., Rao, R.B.: Bayesian network learning with parameter constraints. Journal of Machine Learning Research **7** (2006) 1357–1383.
18. Howe, E., Lenfestey, J., Temple, T.: Intro to Probabilistic Relational Models. MIT Open Courseware, Avanced Lectures (2005).
19. Mengshoel, O.J., Wilkins, D.C.: Abstraction and aggregation in belief networks. In: Abstractions, Decisions, and Uncertainty: Collected Papers from the 1997 Workshop. AAAI Press, Menlo Park, CA, USA (1997) 53–58.
20. Koller, D., Pfeffer, A.: Object-oriented Bayesian networks. In: Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-97), Providence, Rhode Island, USA (1997) 302–313.

[13] Based on IBM US Patent 13/281,037 (2011).

21. Howe, E., Lenfestey, J., Temple, T.: Extensions of Bayesian Networks. MIT Open Courseware, Avanced Lectures (2005).
22. Aji, S.M., McEliece, R.J.: The generalized distributive law. IEEE Transactions on Information Theory **46**(2) (2000) 325–343.
23. Apte, C., Liu, B., Pednault, E.P.D., Smyth, P.: Business applications of data mining. Communication of the ACM **45**(8) (2002) 49–53.