Uncertain reasoning for business rules

Hamza Agli

SORBONNE UNIVERSITÉS, UPMC
ÉDITE de PARIS

UNCERTAIN REASONING FOR BUSINESS RULES

Hamza AGLI

Dissertation presented in partial
fulfillment of the requirements for
the degree of Doctor of Philosophy

July 2017



UNIVERSITÉ PIERRE ET MARIE CURIE

THÈSE DE DOCTORAT

École doctorale Informatique, Télécommunications et Électronique (Paris)

IBM France Lab/LIP6

Raisonnement Incertain pour les Règles Métier

par

Hamza AGLI

Pour obtenir le grade de :

DOCTEUR EN INFORMATIQUE

Spécialité :

Intelligence Artificielle/ Aide à la décision

présentée et soutenue publiquement le 7 Juin 2017

devant le jury composé de :

M. **Philippe Leray**,	Prof., Polytech'Nantes	Rapporteur
M. **Mathieu Serrurier**,	HDR, Université Toulouse 3	Rapporteur
M. **Hassan Aït Kaci**,	HDR, ACM	Examinateur
M. **Stephane Doncieux**,	Prof., UPMC	Examinateur
Mme. **Vanda Luengo**,	Prof., UPMC	Examinateur
M. **Christophe Gonzales**,	Prof., UPMC	Directeur de thèse
M. **Pierre-Henri Wuillemin**,	MdC., UPMC	Co-Directeur de thèse
M. **Philippe Bonnard**,	Ing., IBM	Co-Directeur de thèse
M. **Christian de Sainte Marie**,	Dr., IBM	Co-Directeur de thèse

All Praise is to Him

To Mohamed ...

Acknowledgments

I would like to express my gratitude towards everyone who supported me during this journey. I am tremendously grateful to my supervisor Christophe Gonzales for inspiring me how to do research, for teaching and correcting me and for his continuous support. My deepest appreciations to Philippe Bonnard and Pierre-Henri Wuillemin for their countless explanations and reviews, their availability, patience and fruitful discussions with me. Their insights, kindness and help are just priceless. I have learnt so much from them. Thank you for being such wonderful supervisors !

At IBM, Christian De Sainte Marie was more than a manager, with him I had insightful discussions and his comments were truly helpful to move forward.
I also would like to thank Patrick Albert for the inspirational enthusiasm he gave me to start this PhD. I am thankful to IBM France Lab for supporting this research with the IBM PhD fellowship and ANRT with the CIFRE grant.

I sincerely thank Philippe Leary and Mathieu Serrurier for the effort and time they spent to judge my work. I am thankful to Stephane Doncieux, Veronique Delcroix, Hassan Aït Kaci, Vanda Luengo for their participation in the jury and evaluating my work.

Many thanks to my colleagues at IBM and LIP6 who made this experience amazingly unique. Thanks to Stéphane for being there every time I need him. I am grateful to Nicolas for his smile and kindness to answer my several technical questions. I will never forget the kindness and help of Jean-Louis as he was always available to share his technical expertise. I want to thank Lionel for the time he spent to answer my questions about his programs and share his PhD experience

with me. Thanks to Thomas for his help to find a use case for our research. I also enjoyed chatting with other colleagues, I mention Changhai, Moussa, Yiquan, Pierre, Stephanie, Pierre-André, Françoise, Julie, Rachel.

During this PhD, I enjoyed meeting many PhD students with whom I spent unforgettable and pleasing time. I have to mention Nawal, Oumaima, Karim, Nicolas, Olivier, Ahmed, Santiago, Matthieu, Reda, Penelope, Hugo, Siao-Leu and Cécile. Finally, a very especial thanks to Youssef, El Houcine, Jamal, Ismail, Nidhal, Fatima, Amina, Mustapha, Saed, Abdalah, Abdel Hadi, Hamza, Youness, Houssame, Amine, Imad, Yvain, Alexis, Mélaine, Fayçal, Kamal, Ali, Alae and Daoud for their caring friendship, encouragement and advice.

Words disappear when I want to thank my beloved parents. Their bounties surround me and their continuous presence gives meaning to my life. Let them know my complete and deepest gratitude for their unlimited support, love, trust, encouragement and the hope they put on me. Similarly, I am profoundly grateful to my dearest brothers and sisters Fati, Amine, Tariq, Reda, Karim, Mimiya and Nora. Please accept my unbounded and eternal thanks. All credit to my family.

I am deeply indebted to my beloved wife and best friend Meriem. Your are the beauty and love of my heart. I cannot express my gratitude for what you have done for me. My heartfelt thanks to you for being always by my side, such a great wife. An affectionate thanks to my little hero Muhammad who came as sweet gift and reward to brighten up our life, I am proud of you dear son.
Shoukran'bzaaf ...!

شكرا بزاف !

Résumé

Nous étudions dans cette thèse la gestion des incertitudes au sein des systèmes à base de règles métier orientés objet (Object-Oriented Business Rules Management Systems ou OO-BRMS) et nous nous intersessions à des approches probabilistes. Afin de faciliter la modélisation des distributions de probabilités dans ces systèmes, nous proposons d'utiliser les modèles probabilistes relationnels (Probabilistic Relational Models ou PRM), qui sont une extension orientée objet des réseaux bayésiens. Lors de l'exploitation des OO-BRMS, les requêtes adressées aux PRM sont nombreuses et les réponses doivent être calculées rapidement. Pour cela, nous proposons, dans la première partie de cette thèse, un nouvel algorithme tirant parti de deux spécificités des OO-BRMS. Premièrement, les requêtes de ces derniers s'adressent seulement à une sous partie de leur base. Par conséquent, les probabilités à calculer ne concernent que des sous-ensembles de toutes les variables aléatoires des PRM. Deuxièmement, les requêtes successives diffèrent peu les unes des autres. Notre algorithme exploite ces deux spécificités afin d'optimiser les calculs. Nous prouvons mathématiquement que notre approche fournit des résultats exacts et montrons son efficacité par des résultats expérimentaux. Lors de la deuxième partie, nous établissons des principes généraux permettant d'étendre les OO-BRMS pour garantir une meilleure inter-operabilité avec les PRM. Nous appliquons ensuite notre approche au cas d'IBM Operational Decisions Manager (ODM) dans le cadre d'un prototype développé, que nous décrivons de manière générale. Enfin, nous présentons des techniques avancées permettant de compiler des expressions du langage technique d'ODM pour faciliter leur exploitation par le moteur probabiliste des PRM.

Abstract

In this thesis, we address the issue of uncertainty in Object-Oriented Business Rules Management Systems (OO-BRMSs). To achieve this aim, we rely on Probabilistic Relational Models (PRMs). These are an object-oriented extension of Bayesian Networks that can be exploited to efficiently model probability distributions in OO-BRMSs. It turns out that queries in OO-BRMS are numerous and we need to request the PRM very frequently. The PRM should then provide a rapid answer. For this reason, we propose, in the first part of this thesis, a new algorithm that respects two specifities of OO-BRMSs and optimizes the probabilistic inference accordingly. First, OO-BRMSs queries affect only a subset of their base, hence, the probabilities of interest concern only a subset of the PRMs random variables. Second, successive requests differ only slightly from each other. We prove theoretically the correctness of the proposed algorithm and we highlight its efficiency through experimental tests. During the second part, we establish some principles for probabilistic OO-BRMSs and we describe an approach to couple them with PRMs. Then, we apply the approach to IBM Operational Decision Manager (ODM), one of the state-of-the-art OO-BRMSs, and we provide a general overview of the resulted prototype. Finally, we discuss advanced techniques to compile elements of ODM technical language into instructions that are exploitable by the PRM probabilistic engine.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 General Context

In the contemporary business environment, a great deal of operations is performed or supported by information technology (IT) systems. Ironically, IT does not always "support" the business but also may cause its failure or lower the quality of its deliveries [1]. For example, it is known that a lot of IT-based projects miss their deadlines because of IT obstacles. In addition, initial requirements might be exposed to changes and IT systems, however, hardly follow the change. There are many causes of this problematic situation and a notable cause is related to business policies and rules. Despite the importance of these latter in driving business operations, there were surprisingly no well-established methodology, prior to the business rules approach that we introduce later, to emphasize these rules and efficiently encode them into a computer-based application. Information about such rules is often hidden or repeated throughout the application modules. Even worse, it could be lost, unknown, inconsistently captured or not precisely stated, and this leads to businesses that are run in contradiction to their rules and goals.

[1] around 66% of the US projects fail according to [123, 124]

This is because there is often a misalignment between the business requirements specification and the IT implementations. The separation of concerns principle promotes a modular architecture for developing IT systems, based on a loosely inter-operating services (concerns). After *data*, business rules (BRs) follow this principle and become an independent concern. Actually, in the last decades, the Business Rules Approach emerged as a well-established methodology to allow for a better management of BRs. The approach introduces a new layer of BRs to the IT systems design and architecture. Business systems development becomes business-centric and BRs become a separate service inside a Service Oriented Architecture (SOA), i.e., they are now independent entities, distinct from the data and the view aspects of the system and separated from processes [2]. There are many ways to express the rules that govern or guide business activities including the `IF/THEN-ELSE` constructs, e.g., "`If the category of the customer is Premium then add a 40% discount to his shopping card`".

In a competitive context, agility is the real time capacity to adapt to the market change and react in a direction that benefits the organization. The sake of agility is at the core of the business rules approach. The latter replies to the fast and constant change of the business environment with an agile and flexible development process [19]. Requirements and rules are no more hidden or scattered along business applications, but their automation is now managed and maintained trough a dedicated system called Business Rules Management Systems, BRMSs hereafter. In recent years, such tools have proliferated and many organizations adopted the approach for the sake of agility, re-usability and consistency of BRs. These business-driven tools have the advantage to enforce business policies and prove to help the enterprise to reduce costs and improve their performance and productivity. This explains why BRMSs industry witnesses an increasing annual

[2] a set of inter-related tasks and activities

growth, as it is shown in recent research reports such as [62, 64], which estimated the global BRMS market to reach $636.7 million in 2015 with a Compound Annual Growth Rate (CAGR) up to 17.8% during the period 2015-2020. It is worthy to notice that BRMSs can be also integrated with different Business Process management tools and to reinforce the alignment of computer science practices with business administration goals [131].

Having complete or certain information about our domain is not always a realistic scene. On the contrary, uncertainty is everywhere, it affects data, knowledge and processes. In particular, reasoning under uncertainty becomes an important and active area of research within the field of Artificial Intelligence (AI). Although there is no consensual definition for AI, one can say that it is about building programs or machines that try to imitate human intelligence at a certain level and improve their outputs as long as they consume more experiences. According to [115], different people approach AI with different methods while having different goals in mind. [115] also emphasizes four aspects about AI that are related to thinking humanly, thinking rationally, acting humanly and acting rationally. While the first two approaches are concerned with reasoning, the last ones address behavioral aspects. During the last decade, AI has witnessed a remarkable progress as well as a proliferation of its applications. Fig. 1.1 depicts some of the AI key sub-domains, although the list is not exhaustive.

Reasoning includes inferring the state of the domain at hand based on knowledge about it and some prior observations. For example, a diagnosis of a patient's symptoms, blood and urine might help doctors to determine the diseases in question, if any, and prescribe a treatment accordingly. A Nuclear-Magnetic-Resonance (NMR) well-logging tool helps petro-physicist, using sensor measurements, to investigate earth formations, in particular, to characterize hydrocarbon reservoirs

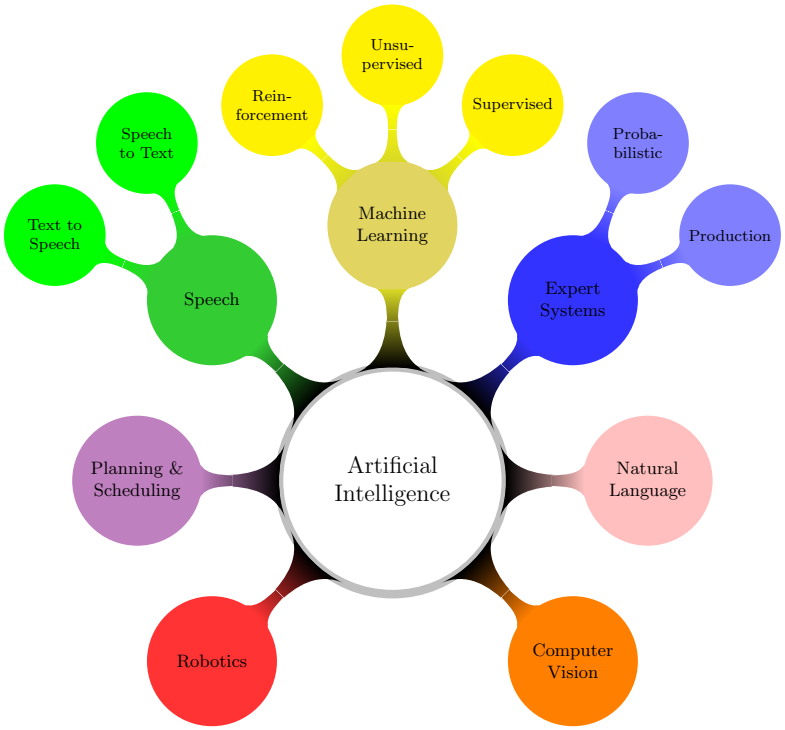


Figure 1.1: Some areas of AI

and decide whether water, gas or oil are present. In these examples, we may lack the adequate information to make a good decision. Some aspects about the patient are just unobservable, e.g., only few historical records are available, and other aspects must be estimated. Observations themselves may be affected by measurement errors, which implies lack of accuracy and precision. In the example of NMR well-logging, measurements could be noisy due to vibrations of the sensors and are unreliable in some particular environments. In all the above situations, the decision making process is no more deterministic and we must take into account uncertainty in the reasoning process to reach an acceptable decision.

Indeed, uncertainty is an inescapable aspect of real world data and several theories and frameworks have been proposed to handle it. However, we are interested in one popular solution that is based on Probabilistic Graphical Models (PGMs)

[67, 72, 102]. PGMs mix the mathematical power of probability theory with the intuitive representation of graph theory. Therefore, they have the advantage to separate knowledge representation from the reasoning process. Broadly speaking, a PGM, be it directed and undirected, exploits dependencies between the variables of interest to compactly capture the uncertainty of the domain and derive conclusions using sophisticated inference algorithms.

1.2 Motivation

Current BRMSs perform well in the presence of complete information about the domain using first order logic. The mechanism of Forward Chaining[3], for instance, permits to infer conclusions from observations about the domain. It turns out that the BRMS technology has roots in the Rule-Based Systems (RBSs), a sub-field of AI. Even though the latter has an active research community on uncertainty, there is no current BRMS that offers a satisfactory support for it. As a matter of fact, current BRMSs turn to have poor facilities for uncertainty management. They either assume the encoded knowledge to be deterministic or use some heuristic models that are unfortunately limited.

There are two needs that motivate this thesis. The first one is related to Operational Decision Manager, ODM hereafter, which is the IBM market leading BRMS and the company seeks to enrich its reasoning process by handling uncertainty. In this thesis, we study how AI techniques can answer such an issue and we focus on the PGM framework for it has common concepts to share with BRMSs. Thus our first research question is the following :

Q_1 : How can we efficiently manage uncertainty in BRMSs using the PGM framework?

[3] see Section 2.3.2

The working memory that stores facts about the knowledge in a BRMS is dynamic and incremental by nature. For example, new objects might be inserted or removed or new relations could be defined. This remark directs us to our second motivation. The probabilistic inference should take the property of incrementality into account. As we will see, the literature does not fulfill this need sufficiently. Therefore, our second research question can be formulated as follows:

Q_2: How can we perform probabilistic inference in an incremental and optimized manner?

1.3 Contributions and Outline

This thesis has two main contributions. First, we introduce a new incremental probabilistic inference algorithm, which is well suited for the dynamic nature of ODM. We provide experimental tests to shed light on the gain we obtain using our approach. Furthermore, our algorithm is being added as a contribution to the development of the open source aGrUM library (a Graphical Universal Model). The latter is developed by our research laboratory LIP6 to provide state-of-the-art implementations of graphical models for decision making. Note that we develop a general algorithm that can be applied whenever the system at hand is incremental and multi-target. As a consequence our algorithm is not restricted to BRMSs, that is why we will first handle $\mathbf{Q_2}$. Our second contribution provides an answer to $\mathbf{Q_1}$ through a coupling architecture with PGMs. Therefore, we propose an effective methodology to ensure the coupling and, as an application, we develop an internal prototype on top of ODM. Generally speaking, the prototype maps the model upon which the BRMS is built to a PGM and delegate the BRMS's probabilistic queries to the PGM's engine. Computations can then be performed using the algorithm developed in the first contribution. We also provide some principles to

ensure more advanced probabilistic reasoning for BRMSs.
Our research results in the following scientific publications:

1. Modèles Probabilistes Relationnels et leur inférence incrémentale pour les systèmes à base de règles orientés objet, Revue d'Intelligence Artificielle 2017, submitted [5].

2. Incremental Junction Tree Inference. In Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU) 2016 [3]

3. Un algorithm d'arbre de jonction incrémental, JFRB 2016 [4]

4. Business Rules Uncertainty Management with Probabilistic Relational Models. In Rule Technologies. Research,Tools, and Applications (RuleML) 2016 [2].

5. Uncertain Reasoning for Business Rules. In RuleML doctoral consortium 2014 [1]

6. Des règles métier pour la gestion de l'incertain, JFRB 2014 [6]

This manuscript is composed of 5 Chapters and is organized as follows. Chapter 2 gives an overview of the business rules approach and stresses upon its technological implications, which are manifested in the adoption of BRMSs. Then, we explain the origins of BRMSs and see different approaches used to tackle the question of uncertainty. As we already mentioned, we are interested in PGMs and in particular, in the formalism of Bayesian Networks (BNs) and its object-oriented extension Probabilistic Relational models (PRMs). Hence, Chapter 3 introduces BNs and PRMs and describes how they allow a decision maker to model uncertainties in the domain and, later, to infer its state given some observations. We also present some popular probabilistic inference algorithms used by BNs. In

Chapter 4, we provide an answer to $\mathbf{Q_2}$ and we propose a probabilistic inference algorithm that is adaptive to incremental and multi-target systems, and ODM in particular. In addition, the chapter highlights the effectiveness of our method based on experimental tests. In Chapter 5, we provide an answer to $\mathbf{Q_1}$. After introducing the necessary background, this chapter proposes a method to empower BRMSs capabilities with uncertainty management. Our results are implemented in a new prototype developed on top of ODM. We describe the prototype's main components and see how uncertainty can be handled in such an industrial context. Chapter 6 concludes this manuscript with a summary of our work and proposes some directions for future works.

Chapter 2

IT for Business Rules Management

We recall that this thesis aims at extending the capabilities of IBM ODM [63], a Business Rules Management System (BRMS), to handle uncertainty.

BRMSs are a family of computer-based systems that result from marrying a business rule approach with the technology of rule-based expert systems. We first describe such an approach, then we explain the components and mechanisms of the technology that support it.

2.1 Introduction

There is a well-established business tradition of automating parts of the policies that govern the decision-making process in computer-based applications [50, 90]. However, the business environment, being characterized by a rapid and constant change, does not facilitate such an automation. As a consequence, it becomes vital for modern organizations, first, to build more agile systems and second to adapt quickly to the highly competitive and dynamic market (competitive, new products

and prices, new customers and preferences, etc ...). Actually, changes that affect business have many facets including government and legislation policies changes such as taxation laws, interest rates and environmental regulations. Last but not least, IT systems have drastic influence on how business operates and organizes its processes, e.g., projects outsourcing, online services, communications tools, storing data and extracting valuable information from it. Indeed, the business and its supporting IT systems are very inter-related and both must meet the requirements of the business ever-changing environment. In [37], the authors highlight three components for the development of any business information system: data, process and policies. Whereas the first two have been integrated using the object-oriented paradigm, policies are commonly neglected and left implicit in the program code. In a business project, it is frequent to see business logic scattered over different applications in the information system [90] and paradoxically, business experts, who own the knowledge about business policies, have very limited access to the encoded business logic. The problem is worse when the business needs to change its operations and policies, and consequently the supporting IT system. This results in high costs to adapt current IT systems to the changes that occur, as business logic is distributed all over different applications. Hence, conflicts occur between business owners and application developers in term of aligning IT with business.

Even the *triggers* used in active databases to express business policies as constraints were very limited and therefore, business policies that were hard-coded as procedures in SQL or COBOL fail to provide flexible systems and result in many maintenance issues [50, 71]. The problem is identified as the missing link between high-level data architecture and project-level physical-database design or data-processing [9, 128]. One of the early answers to this problem consists of using object models to integrate and explicitly represent business policies in terms

of rules [37, 107, 108, 128]. However, the most popular approach that continues to gain acceptance among business practitioners and researchers is known as *the business rules approach* [53, 90, 113]. This approach promotes *Business Rules* (BRs) as an independent formalism to represent the business logic in terms of rules and manage them within centralized applications called Business Rules Management Systems (BRMS). The following paragraph further elaborates such an approach.

2.2 Business Rules Approach overview

When the term of business logic is invoked, it consists of all parts of the IT system responsible of specifying the schema that defines and processes the information flow in a database. This encompasses essentially constrains and policies that govern various aspects of business operations. Although the term of BR seems to appear first in [9], where *rules* are identified as an independent component that links data architecture and data processing, such rules have been often conflated with constraints of the databases community or policies [50]. Before defining what is a business rule, let us clarify first the business policy.

Definition 2.2.1 (Business policy)**.** *A business policy is a statement of directions or guidelines that governs the decisions of an organization and controls its actions scope.*

For instance, an insurance organization might have a policy saying :

Example 2.2.2 (Discount policy)**.** *All Platinum customers in Paris who place an order during the summer receive a 45 percent discount.*

BRs have been given many definitions in the literature, albeit the most influencing was produced by the GUIDE Business Rules Project [58]. This latter aims at formalizing an approach, called the business rules approach, to identify

and articulate the constraints that define the structure and control the enterprise operational decisions. Thus, the approach advocates enforcing business policies by translating and decomposing them into more specific statements, which will constitute BRs. the Business Rules Group (BRG)[1] promotes the *independence principles* of BRs in their seminal upshot *Business Rules Manifesto* [112]. Accordingly, BRs should be managed separately and directly by business workers.
Finally, BRG has contributed to the work of the Object Management Group (OMG) to standardize the approach, which results in the emergence of the Semantics of Business Vocabulary and Business Rules (SBVR) standard [93]. SBVR is a metamodel for developing semantics models in a business domain in the form of a vocabulary and a set of BRs. The following definition is an adaptation from [50, 112]

Definition 2.2.3 (Business Rule)**.** *A BR is a compact, atomic, well-formed, declarative statement that defines or constrains an aspect of the business and its collaborators. It must be expressed against a domain ontology (business policies vocabulary) in a natural-language that is understandable by whom it may concern, such as business and IT professional and customer.*

BRs are the atoms that constitute a business policy and can be typically formulated via a set of conditions and actions. For example, the above business policy can be translated into the following two BRs.

Example 2.2.4. *(Discount BRs.)*
IF the value of the shopping cart of the customer is more than 1500
THEN set the category of the customer to "Platinum"
IF the customer's city is "Paris" and the customer's category is "Platinum" and the order date is between "June 21,2017" and "August 20, 2017"
THEN set the order discount to 45 percent

[1] a group formed by IT practitioners and business analysts inside the GUIDE project.

According to [113], a BR is based on a collection of noun concepts called *terms* and gathered in a glossary called *concepts catalog.*. For instance, `Customer`, `customer's city` and `Customer's category` represent terms in Example 2.2.4. Terms are words or phrases that reflect business concepts and whose exact meaning is specified and agreed upon among collaborators in a business.
Interactions between terms are known as *facts* [113] or *fact types* [93]. Facts are verb concepts that link together appropriate terms. For instance, the assertion `Customer has a city` is a fact that relates a customer to a city in Example 2.2.4. Now, BRs are simply built on top of these facts by expressing how these facts must/should (not) be constrained in order to guide business operations [113].
Finally, it is important to highlight the following aspects about a BR:

- Atomic structure: BRs cannot be reduced or decomposed into multiple or derived BRs, otherwise one can loose important information about the business.
- Business determination, in the sense that it must define or constrain some aspects of the business.
- BRs are under the 'business jurisdiction' and derived from its policy in the sense that they only regard what is under the company's authority to modify.

In the business rules approach, all BRs should be collected managed withing a central rule repository. The approach recommends the use of a business rule engine to execute rules directly rather than transcribing them into some procedural form [52, 112]. Although different ways to implement the BRs approach might exist, using BRMSs remains the most economical one [50]. In fact, BRMSs externalize BRs and provide facilities for a centralized management. Thus, BRs and other IT components have a separate life-cycle and yet they can easily communicate with each other. As a consequence, business workers benefit from direct access

to BRs while collaborating more independently with IT developers. In [114], the authors emphasize that the business would need its BRs even if it had no software, which means that developing a BRs solution does not necessarily require a software development. Although this may implicitly attribute less importance to the technology behind, the approach promotes the use of BRMSs to support, better than any previous systems, the continuous and fast changes in the business environment. According to [19], a BRMS is a necessary component to ensure an agile BRs development and without which the BRs approach would not be fully implemented. Putting business first should not be understood as neglecting the technology that supports them, because real world experience shows that good technology can even shape BRs and significantly enhance time, quality and money outcomes of business projects. Therefore, adopting a BRMS will definitely build better, changeable systems faster that any previous approach [54]. Today the BRs approach has reached a good maturity. This is shown, on one hand, by several studies that appear to cover the subject [19, 33, 50, 53, 54, 90, 113]. On the other hand, by the increasing number of applications that implement it such as in banking, insurance, health care, retail and manufacturing. The next section gives more details about the basics of BRMSs.

2.3 Business Rules Management Systems

The separation of concerns principle is wildly adopted in software engineering. It essentially consists of dividing as much as possible applications into separate components or modules. A very prominent application of this principle is the Service-Oriented-Architecture (SOA) paradigm. The adoption of SOA and BRMS together will let businesses deploy or integrate new applications far more easily [50]. As we mentioned in the previous paragraph, BRMS were introduced to help

aligning IT with business. In the IT architecture perspective, BRMSs come as a separate layer and can be seen as a service among others such as database management systems. BRMSs externalize and centralize the management of BRs to avoid distributed and redundant updates. Finally, BRMSs separate BRs from the technical implementation, thereby, BRMSs are the technical solution that gives business workers the control over BRs with greater agility. The way BRMSs manage BRs is largely borrowed from early rules-based systems (RBSs) . Indeed, BRMSs inherit from these how to effectively store, in a declarative manner, and reason, in a component-based architecture about that knowledge. We refer to the next section for more details about RBS.

Being a complex software system, a BRMS additionally implements the business logic by providing the environment for designing, authoring, checking, deploying and executing BRs. Therefore, modern BRMSs provide business experts, who own the business knowledge and processes, with tools to manage BRs directly without being IT specialists. It represents a common ground for collaboration with these latter, though.

It is worthy of note that besides their declarative aspect in the the condition part, which is inspired from functional and logic programming, BRMSs have also a procedural capacity à la Java or C# in their action part. Such languages, which are based on the conceptual framework of *Object Model* (OM), have proven to be particularly well adapted to construct well-structured complex systems. In addition, this use of an object-oriented (OO) programming at the core of BRMSs has naturally emerged as a de facto standard from the most successful real business applications. This is because an OM encompasses essentially the principles of abstraction, encapsulation, modularity and hierarchy. Therefore, it provides a clear advantage that facilitates design and software reuse. Having said that, the domain ontology mentioned in Definition 2.2.3 corresponds to the definition of an OM ex-

pressing business concepts. In practice, a business domain can often be modeled by identifying separate sub-domains, i.e., a set of interrelated *class of objects*. Each class is characterized by a number of fields or attributes. Illustration of classes from Example 2.2.4 are customer and order. Attributes might be a name or city for a customer. This latter might be related to many orders. An OM defines also the operations that can be performed, such as providing a discount or displaying a specific message. In this thesis we are only interested in OO-BRMS.

2.3.1 Features of a BRMS

A modern BRMS, be it commercial or open source, provides some common features including classical inter-operable modules.

- development environment (IDE) to edit rules and the underlying models. It also provides a graphical interface for non technical people to author rules. An IDE can be integrated as a plugin in Eclipse for example.

- inference engine that is responsible of executing the rule against the facts base and performing actions. The execution is often based on the Rete algorithm and its enhancements, but also a sequential algorithm can be applied. Engines operate essentially in a forward strategy, a backward or mixed strategy are also possible. [2]

- the rule base, which is a centralized and searchable repository to store and manage collaboratively BRs. Note that other alternatives to enter rules, exists such as decision tables, decision trees and rule-flows.

- support for rule flows and events processing to chronologically order business processes execution and specify transition time from one process to another.

[2] see next section for details

For simplicity, a process can be regarded as a set of activities that will evaluate a set of rules. Most of BRMSs provide a graphical editor for creating flows and allows a drop and drag functionality.

- programming and execution environment are different from one BRMS to another. Every product has its own language to write rules technically, but all of them provide a mechanism to write in a natural language-like format. The runtime is generally based on Java or .NET. Facilities for checking the rule syntax in real time and debugging do exist in modern BRMSs.

2.3.1.1 Mainstream BRMSs

The mainstream BRMSs include IBM ODM [63], Fair Isaac Blaze Advisor [43], Red Hat JBoss Enterprise BRMS [110], Progress Corticon [109], Oracle Business Rules [96], Microsoft BizTalk [86], SAP NetWeaver Business Rules Management [117], PegaSystems PegaRules[104], InRule Technology [65], OpenRules [95] and Bosch Visual Rules [18]. Comparing these systems is out of the scope of this thesis, we refer to [51] for a comprehensive comparison including the first two of them. In addition, [51] provides a multi-criteria decision analysis-based method to evaluate them. In particular, the author concludes that none of the leading BRMS products available today has any sophisticated facilities for managing or reasoning under uncertainty.

As we already mentioned, BRMS technology is heavily inspired from RBSs technology. RBSs have a key feature to manage effectively the business knowledge. Notably, they ensure a clear separation between the declarative knowledge from the code that uses that knowledge to solve a specific problem or to satisfy a particular business requirement.

2.3.2 Rule-based Expert Systems

The story of BRMS technology is tightly linked to AI and all began with rule-based expert systems. These form a sub-area of AI that had flourished from mid-sixties till the end of the past century and still influences today AI technologies. Technically speaking, an expert system might be defined as a computer program designed to simulate, support or improve upon the decision-making process of a human expert in a specific area, where intensive knowledge is required. So, an expert system can either replace the human expert, e.g., control systems in manufacturing or provides helpful insights to the expert, e.g., medical diagnosis. Particularly, one popular way to capture and store the knowledge in a RBS is in the form of *production rules* i.e., a set of implications having the form of "IF-THEN-ELSE" statements [3], that is why RBSs are referred to as production (rules) systems. Production systems were popularized by Newell and his collaborators while working on reproducing human reasoning and problem solving, see for example [75, 91]. The most influential programming language for these systems was OPS5 by Charles Forgy [20, 45], which is written in LISP and uses the Rete algorithm [46] to draw conclusions from rules. Actually, OPS5 was successful and surpassed other implementations due to its efficiency and capacity to scale up.
The potential of RBS, which has been first demonstrated and well-publicized by AI researchers, led to great successful applications. This success attracted many organizations to use RBS technology to the point that over two thirds of the Fortune 1000 companies during the eighties applied the technology to daily business activities. Furthermore, RBS applications have then proliferated even though the business, manufacturing and medical fields remained the dominant areas in which RBS are used [39]. At this stage, it is worth mentioning some of the early expert

[3] the IF part is also called condition or premise, or left hand side (LHS) and the THEN part is also called conclusion, action or right hand side (RHS)

systems that influenced the course of AI. The seminal one is DENDRAL [21], a chemical RBS used to infer molecular structure from spectroscopic analyses. This RBS was a cornerstone in the history of expert systems and inspired the development of numerous RBSs that came later. The other one is MYCIN[22], which is perhaps the most famous one, is designed to diagnose bacterial infections in human blood. Other successful RBSs include XCON [13], which is written in OPS5 for configuring computer systems orders and was one of the first to be industrialized. Finally *PROSPECTOR* [57], a geological analysis system to discover rocks and minerals deposits. The common point between all these RBSs is their solving-problem power that comes especially from the encoded knowledge and this emphasized the crucial role of the knowledge base. Building expert systems for specific application domains has since then become a separate subject known as *knowledge engineering*, a term coined by Freigenbaum who is a key contributor to DENDRAL [42]. Broadly speaking, the very term "knowledge-based system" refers now to information systems, in which part of human knowledge is (symbolically) represented and reasoned about by mimicking human reasoning.

There are two important components that form any RBS, which can be expressed by the following equation : `RBS = Knowledge Base + Inference Engine`. A user interface is useful as a channel between the system and different users, e.g., an automatic way to enter/update knowledge by a human expert, a question-answer session or an explanation facility to help a user to understand the reasoning process or an action is made. Fig 2.1. depicts the central components of a RBS where arrows reflect information flows

2.3.2.1 Knowledge Base Representation

There are two types of memories in a RBS : a long term and a short term memory. The first one corresponds to the Rules Base (RB) that contains the expert

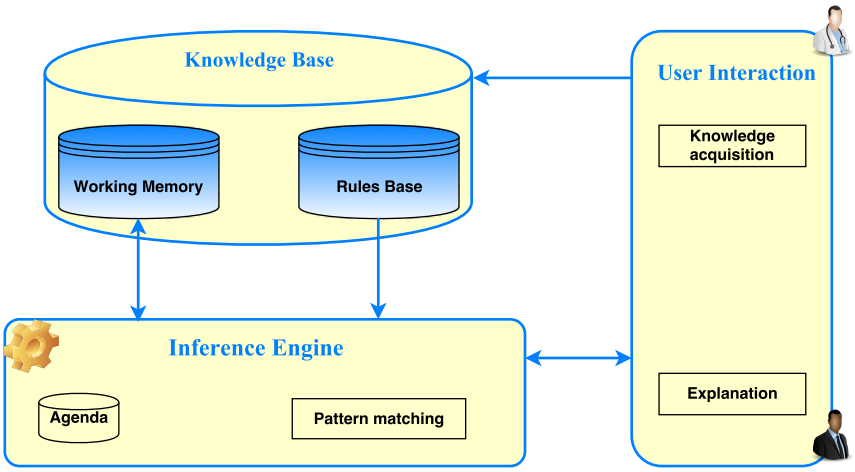


Figure 2.1: Central components of a RBS

domain-specific knowledge. The knowledge it contains is abstract and encoded in the form of production rules, which are expressed informally as rules that are sometimes called heuristic rules of thumb. As we discussed earlier, a rule consists of set of conditions if fulfilled, some conclusions or actions are performed: `If <condition> Then <action>`. An example of such knowledge abstraction might be the following:

Example 2.3.1. *A naive air conditioner system:*
$RB = \{R_1, R_2, R_3\}$ *such that:*
R_1*: IF the air conditioner is on THEN close the window*
R_2*: IF the window is closed THEN open the curtain*
R_3*: IF the room temperature is more than* $35°$ *and the air conditioner is off THEN switch-on the air conditioner*

The order in which such statements are entered into the base is not important. Actually, rule languages are declarative, in the opposite to conventional programming languages like Java or C++, which are procedural and where the order of the `IF/THEN-ELSE` statements matters. In the case of using an OO paradigm, we can see that we are based on an OM to define terms used in the RB. We For instance,

we can use two classes, namely `Conditioner` and `Room`, to define the OM in the Example 2.3.1.

The second memory, called the *Working Memory* (WM) or Facts Base, can be evaluated with respect to the RB. WM is a continuously changing database that contains known facts about different states of the domain objects. Hence WM stores rather a concrete knowledge related to interaction with users. The WM's objects are called WM elements, WMEs hereafter. In our running Example 2.3.1, a WM may contain for instance :

- $f_1 = \{$ `the temperature of the room is more than` $35°\}$,
- $f_2 = \{$`the curtain is open` $\}$,
- $f_3 = \{$`the window is closed` $\}$,
- $f_4 = \{$`the air conditioner is on`$\}$.

RBSs contain a mechanism to control the global coherence and assure the consistency of the knowledge base in order to prevent contradictory knowledge to reach the rule base . Particularly, it specifies constraints on assignable values w.r.t variables domain (coherence of facts) or detect contradictory rules (coherence of rules). The component that is responsible for drawing conclusions by applying the rules in RB to the WM facts is the inference engine, which is the aim of the next subsection.

2.3.2.2 Inference Engine

This component is the RBS's brain that matches rules in RB with WM known facts by applying a pattern matching algorithm. A rule whose condition part is satisfied by a fact is *activated* and is usually stored in an *agenda*, also called a *conflict set*. When several rules are activated, the engine applies a solving procedure called

conflict-resolution strategy to decide which rule to execute, the rule is then said to be *fired*. The simplest conflict-resolution strategy is, of course, just to apply the production rules in the order in which they entered the agenda. However, other heuristics might be applied to prioritize the set of activated rules in the agenda. Then, the engine fires then rules beginning from the highest until no active rule is left. The inference engine processes the knowledge base in cycles, which have been given many names such as select-execute and recognize-act cycles. Algorithm 1 shows an example of the engine execution cycle. After updating the agenda, by either activating a rule if its conditions are satisfied by the current WM or removing it otherwise, the engine executes all activated rules and loops until no new active rule remains.

There exist many algorithms to implement RBSs, but the aforementioned Rete algorithm [46] is the most influential one and is still used in modern rule engines. The key idea of Rete is to compile RB into a graphical representation called the *Rete network*, where shared conditions are no more duplicated. In practice, conditions are first associated with nodes called *alpha-nodes* that select WMEs whose attributes match rule conditions, just like the *select* operator in the conventional

Algorithm 1: Select-Execute cycle

```
1 done ← false
2 while not done do
3     Select a rule to execute // w.r.t., a Conflict Resolution
4     Execute actions of the selected rule and remove it from agenda
5     Update agenda
6     if agenda is empty or a stop condition is reached then
7         done ← true
```

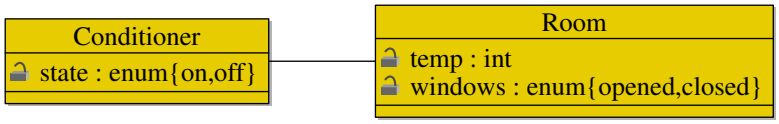


Figure 2.2: UML diagram for air conditioner's Example 2.3.1

database formalism. Then, *beta-nodes* combine two or more satisfied conditions just like the *join* operator.

Let us illustrate this through Example 2.3.1. Assume that our domain is composed of two classes, `Conditioner` and `Room` as it is showed in the UML diagram of Fig. 2.2. Then, Fig. 2.3 depicts a Rete network associated with RB in Example 2.3.1.

Obviously, there is no unique Rete representation since different combinations of conditions exist. The network can be seen as a data-flow graph. Each time a fact is added, it is filtered to match or not the network nodes down to the bottom. If it has passed all the nodes, then the corresponding rule is activated. Rete is a

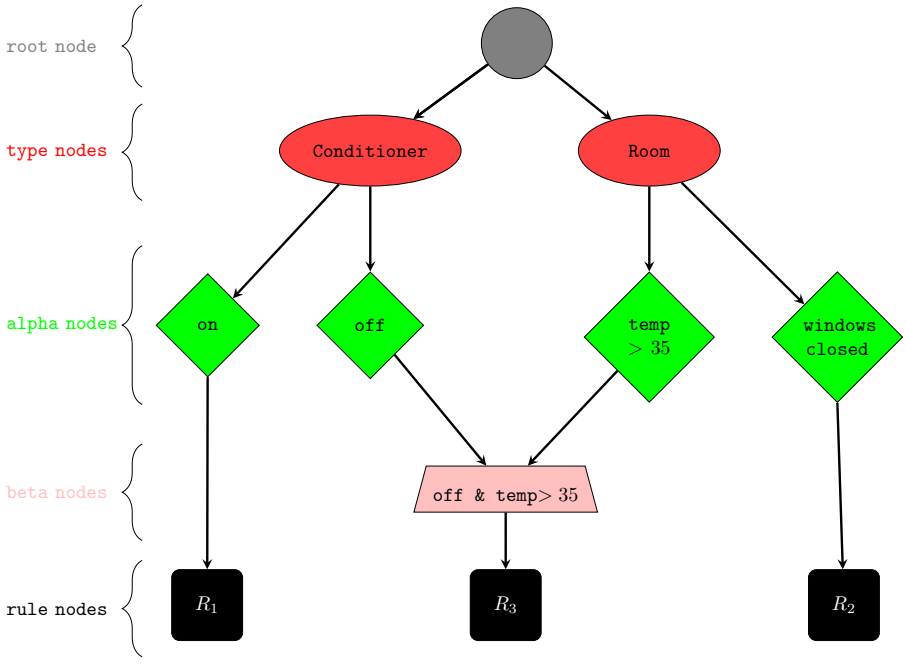


Figure 2.3: Rete network for air conditioner's Example 2.3.1

pattern-matching algorithm such that every path in the network corresponds to a pattern in the rule. Furthermore, every node in the network is a local memory that holds WMEs satisfying the associated conditions. As we can see, Rete is memory-consuming, albeit it efficiently pairs WMEs the with rule base. Note that there exist other algorithms for implementing production systems in general, such as TREAT [87], which introduces a new method of state saving in production system interpreters called conflict-set support, and Gator [56] a generalization of Rete and TREAT.

There are two famous strategies used by the inference engine to derive new knowledge or draw conclusions, namely, the forward chaining and backward chaining. Even though BRMSs[4] are generally implemented using the first strategy, the second one can generally be emulated. Here we use the term chaining to reflect the reasoning chain, which is based on a series of inferences that allow RBS to connect a problem with its solution. Note that hybrid strategies, where we mix backward and forward chaining, are not covered by this section.

Forward Chaining We talk about Forward Chaining when we start the reasoning process by evaluating facts present in the WM w.r.t rules conditions to end with conclusions that follow from the facts. A rule is executed when the conclusion part is executed.
The engine will try to match the required condition for all rules until none is satisfied. If we get back to Example 2.3.1, with $WM = \{f_1\}$, then the strategy consists of selecting rules whose conditions match f_1, hence R_3 will be applied first. The application of the selected rules may change the WM content, enabling consequently other rules to be fired. So after applying R_3, we have $WM = \{f_1, f_4\}$ which results

[4]being a product system, a BRMS is used to produce actions or change the behavior of an environment, hence the forward chaining strategy is naturally adopted in BRMSs

in $R1$ now being applicable and f_3 is added to the WM. As a consequence R_2 is activated and fact f_2 is added. We therefore conclude that the final WM is equal to $\{f_1, f_2, f_3, f_4\}$. The inference process is terminated as soon as all applicable production rules have been processed or a certain stop condition is reached. As we see, this strategy is fact-driven and it is best for prognosis and control systems or systems where no specific goal is being explored. Forward chaining is also called bottom-up reasoning as it reasons from the low-level facts to the top-level conclusions that are based on the facts. It was used in the DENDRAL system that we mentioned previously.

Backward Chaining This strategy is the reverse of the previous one. It is applied when we reason from conclusions and move backward to the facts that support them. If there are no matching facts, the conclusion at hand will be rejected and other will processed until a conclusion is obtained. Hence, this strategy is goal-driven as conclusions can be identified as goals to achieve. More precisely, a goal may match with a conclusion of one or more rules present in the knowledge base. All rules that match a certain goal are selected. In our example, assuming the goal to be f_3, then the only selected rule is R_1. Each one of the selected rules is subsequently processed by considering its conditions as new sub-goals to be achieved, where a sub-goal is achieved when the WM contains a fact supporting it. Otherwise, when a sub-goal finds no supporting fact, we recursively apply the same process by selecting rules whose conclusions correspond this sub-goal. In our example, since R_1 is the only selected rule, we get the new sub-goal f_4, which in turn causes the selection of R_3. When all the sub-goals, i.e., the conditions of the selected rule, have been achieved, then the rule conclusions are executed, which may cause the WM to change. In our example, since the WM contains the fact f_1,

the condition of the selected rule R_3 is satisfied, i.e., f_4 is achieved. Thus, R_3 is executed and f_4 is added, that is, $WM = \{f_1, f_4\}$. Finally, R_1 can be now executed as all its sub-goals, namely f_4, are achieved. In a backward strategy, the engine selects only necessary rules to achieve initial goals and terminates the inference once they are achieved, that is why rule R_2 is not executed in our example even if its condition is fulfilled. This strategy, also called top-down inference, is suitable for diagnosis problems. An example of backward RBSs is the MYCIN diagnosis system.

2.3.2.3 User-System Interaction

The user-system interaction component allows RBS to communicate with users, which hereafter stand for end-users, knowledge engineers or domain experts. For example, it helps the knowledge engineer to develop and maintain knowledge base or to enter additional data. Another scenario may consist of displaying conclusions made by the system. Furthermore, different modes of user interfaces can range from simple command line to a sophisticated graphical user interface, but all modes should provide an easy and user-friendly experience. A good user interface hides the internal technical rules representation and presents them in an understandable form to users.

Another aspect of the human-machine interaction is to provide explanation facilities that explain or justify, in understandable and acceptable terms, the process followed by the system to reach conclusions. Providing such a feature was at the core of research from the beginning of expert systems and was considered as one of their most important characteristics that impacts the user experience acceptance [38, 135]. In a dialogue session, when the system prompts the user to enter some information and the user asks *why*, this facility may reveal which rule of the system is being used. Moreover, tracing activated rules and contents of the WM

can be useful to answer *how* an action is reached. Another interesting explanation facility is the *what-if* analysis to explore alternate solutions using a hypothetical reasoning. However, simply tracing the activated rules or paraphrasing the chain reasoning does not help the user to easily understand the process employed by the system [89]. Actually, although the research community stressed the importance of explanatory capability, most early expert systems have only limited part of it and provide an explanation limited to a description of the reasoning steps [82, 132]. This research area is fertile and still witnesses active efforts to overcome limitations of unsatisfactory explanations. According to [125], limitations stem essentially from the inadequacy of generating explanations directly from a very low-level representation (production rules), which fails to capture all the information needed for explanations and fails to distinguish the roles that different kinds of knowledge play. Relatively recent advances are concerned about building new architectures that capture more of the knowledge that is needed for explanation. The key idea is, on one hand, to enrich explicitly the system with dedicated and separate knowledge bases. On the other hand, to view explanations generation as a problem-solving activity that needs its own advanced techniques and architecture. Examples of these systems include the Explainable Expert System [125, 126], which uses *strategic* knowledge about how the system is designed and how it reasons or the Reconstructive Explainer [133], which reconstructs causal chains from a separate knowledge base to justify the reasoning path from inputs to answers. In another direction, [14] claims that making the distinction between reasoning knowledge and communication knowledge is theoretically appealing and allows to manage better the software engineering development of explainable expert systems. Finally we direct the reader to [28, 88] for more detailed surveys and discussions of approaches to explanation.

Note that many efforts has been done to construct development environments as

a general base for building specific expert systems. These environments are independent of the domain-specific knowledge and use a predefined formalism for knowledge representation and corresponding inference engine, which gave birth to what is known as expert system *shells*. First of them, the EMYCIN system [85] was designed by stripping MYCIN of its knowledge base. Nevertheless, most of the early expert systems were ad hoc and case specific applications, which were generally hard-coded in the LISP (LISt Processing) language.

Despite the optimism and early successes of expert systems, these often involve expensive development and maintenance costs. In [79] for example, it is argued that expert systems lack some validations and critical assessment because they are not widely tested and this raises legal and ethical issues. Finally, we want to emphasize that it is indubitable that early RBSs suffer from fundamental limitations when dealing with uncertainty [103] and these problems, among others, caused the emergence of a new class of expert system called probabilistic expert systems, a subject discussed in much details in in the next chapter.

2.4 Discussion and Conclusion

Although the term of imperfection is used by [121] to include all types of uncertain or imprecise knowledge, the term uncertainty is often equivalently used. Unfortunately, it turns out that whereas BRMSs are well adapted to deal with structured and complete data by using classical Boolean inference, they face some difficulties when they take into account imperfect data [94]. BRMSs must integrate mechanisms to handle the issue of uncertainties in the domain. To this aim, three approaches are commonly used in the RBS community:

- Heuristic models, which weight rules with a degree of truth, e.g., certainty factors (CF) and likelihood ratios (LR) [22, 57]. These deal with uncertainty

in the knowledge (rules) not the data. However probabilistic interpretation given to CF is incoherent with probability theory [61]. On the other hand, the conditional independence between evidence and rules actions in LR is seldom satisfied in real applications and LR-based expert systems have poor performance [92].

- *Fuzzy logic* (FL) [136] which describes imprecision or vagueness by associating variable values to fuzzy sets, a kind of moving from a bivalent logic to a fuzzy one. However, FL is not in essence designed to deal with incomplete data or to express relations between variables in the knowledge base as in OO frameworks. In addition, FL when applied to systems that perform chains of inference, such as BRMS, may lead to inconsistent conclusions [40].
- Bayesian techniques, which are essentially based on Bayesian Networks (BNs) [102], to consistently model domains with uncertainty. In addition, several algorithms have been proposed to learn their graphical structure and their conditional probability tables (CPTs) parameters. Even if they are a very popular tool to deal with uncertainty, BNs are not suited for complex systems, in which they involve high design and maintenance costs [73, 84]. Moreover, they do not support well object-oriented and dynamic systems.

One can also find hybrid approaches that combine, for instance, BNs with CF [16, 74]. Obviously these methods incur some problems discussed previously. Moreover they are developed for specific uses and cannot handle effectively the frequent changes of business policies, where BRMS perform better. Another approach is Probabilistic Logic Programming [35]. But this is not suited for the BRs procedural side effects and the OO-BRMS upon which we build our application.

To summarize, this chapter introduces the business rules approach as a method-

ology where BRs are regarded as first class citizen in the business operational decision management. The main technical implication of such an approach is the adoption BRMSs. We give a brief overview of such applications and we finally describe RBS as they represent their technical ancestors. Most of current BRMSs are based on the OO paradigm, that is why we are particularly interested in OO-BRMS. The aim of this thesis is to enhance these systems with uncertainty management capabilities. Again, in the light of SOA paradigm, we want to integrate a probabilistic reasoner as a service on top of current BRMSs implementations. To this purpose, we use Bayesian Networks and one of their OO extensions. The next chapter introduces such a framework.

Chapter 3

Bayesian Networks for Uncertainty Management

3.1 Introduction

In developing intelligent systems, it is inevitable to deal with the uncertainty that pervades real-world applications. As we discussed in the introduction, this issue has multiple origins such as measurement errors, noisy automatic processes, the modeling process itself, etc. Many approaches have been adapted to tackle this question, such as probability and possibility theories. For detailed surveys on these approaches, the reader might look at [55, 119, 121].

One interesting approach that is well suited is called Probabilistic Graphical Models (PGMs) [67, 72, 102]. This framework combines graph theory with probability theory to represent and reason with complex and real-world data. It has now become a very popular and dominant tool for reasoning under uncertainty in the AI field. PGMs earn such a popularity for many reasons. First, their graphical structure allows them to capture conditional independence properties of a joint probability distribution. This allows for a compact factorized representation of

the distribution and provides sophisticated methods for automated reasoning. Second, their representation is natural and human readable, allowing even non experts to understand the conditional interactions between the variables of the domain. Lastly, the framework is based on rigorous probability calculus that makes the framework mathematically sound. The construction of the joint probability distribution permits to probabilistically draw useful conclusions about one or more variables of the domain knowing some observations about other variables. This technique covers a large spectrum of interesting queries. For example, in a printer troubleshooting application, the specification of the joint distribution of the components and device status allows the computation of the posterior probability that the device will work properly, given some observations of the device behavior. However, specifying the distribution over a high-dimensional space of variables appears intractable since the size of such distribution grows exponentially with respect to the domain size. That is exactly where PGMs play an extremely interesting role to encode such complex distributions in a tractable manner. For the purpose of this thesis, we are interested in directed PGMs that are called Bayesian Networks (BNs) [72, 102]. In this chapter, we are interested in the BN framework and its OO extension, the so called Probabilistic Relational Models (PRMs). We discuss also a graphical test for deciding which conditional independence relations are implied by the structure of the BN

3.2 Bayesian Networks

3.2.1 Definition and Design

BNs [72, 102] were proposed in the late 80s and rapidly became one of the most popular framework to reason under uncertainty in AI and expert systems. BNs [1] are used in a wide range of real-world applications, including medical diagnosis, risk management, fraud detection and clinical decision support [31, 134].

As a directed PGM, the core of a BN representation is a directed acyclic graph (DAG) whose nodes [2] represent random variables (e.g., the status of a device, the occurrence of an event) and whose arcs represent probabilistic dependencies (cause, influence, information flow, etc) between these variables of interest. This enables them to encode compactly the joint probability $\mathbb{P}$ over their nodes as the product of the conditional probabilities of the nodes given their parents in the DAG. This compact factorization of $\mathbb{P}$ into local distributions is known as *the chain rule* of BNs and is similar to the chain rule of conditional probabilities:

Definition 3.2.1 (Chain rule of BNs)**.** *let $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ be a directed acyclic graph over a set of random variables $\mathcal{V}$ (continuous or discrete), where $\mathcal{A}$ is a set of arcs. We say that a joint probability distribution $\mathbb{P}$ factorizes over $\mathcal{G}$ if $\mathbb{P}$ is defined over $\mathcal{V}$ and can be written as the following product:*

$$\mathbb{P}(\mathcal{V}) = \prod_{X \in \mathcal{V}} \mathbb{P}(X|\mathbf{Pa}(X)) \tag{3.1}$$

where $\mathbf{Pa}(X)$ denotes the set of immediate ancestors (parents) of X in $\mathcal{G}$. Now we can define more formally the BN as follows:

Definition 3.2.2 (Bayesian Network)**.** *A BN is a pair $(\mathcal{G}, \mathbb{P})$, where $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ is a DAG and $\mathbb{P}$ is a joint probability distribution that satisfies the chain rule 3.1.*

[1] different designations can be found in the literature: probabilistic networks, belief networks, causal diagrams.

[2] We assume that the random variables are identified with the graph nodes.

In practice, $\mathbb{P}$ is given as the set of local conditional probability distributions of the variables given their parents in the DAG : $\Theta = \{\mathbb{P}(X|\mathbf{Pa}(X)) : X \in \mathcal{V}\}$. Whereas the joint size grows exponentially with the BN size, the total number of parameters is in general bounded by $|\mathcal{V}| \times v^{p+1}$, where v and p are bounds on the domain sizes of the variables and on their number of parents respectively. In the rest, we are interested in discrete BNs, i.e., variables in $\mathcal{V}$ are discrete, and we will rather use the term conditional probability tables (CPTs). To illustrate these concepts, we use the example from [72]:

Example 3.2.3 (The Student Example)**.** *A company wants to hire recently graduated intelligent students based on their recommendation letters, which reflect the student grade in a particular course, and SAT scores, which reflect the student intelligence.*

The process of modeling with BNs follows in general two steps, which will be emphasized through the example. The goal is then to develop a BN to capture the knowledge from the domain problem at hand and use it to help a decision process:

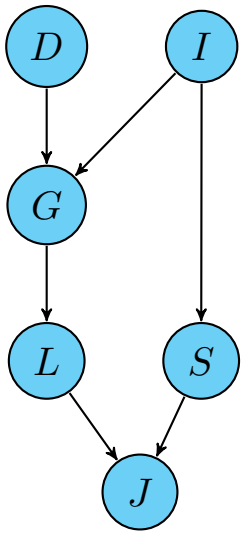


Figure 3.1: A DAG for the student example

3.2.1.1 Qualitative step

During this step, the *independence structure* $\mathcal{G}$ is constructed by defining, first a set of interest variables and their domains (possible values or states) and second, by linking these variables to form a DAG. To this end, one can be guided, for example, by using causal dependencies that capture interactions among variables in terms of cause-effect relationships [59].

One possible way to capture domain variables and their interactions for the student Example 3.2.3 is as follows: a job variable (J) is introduced to reflect whether the student got the job and this only depends on the recommendation letter (L) and SAT scores (S) variables. We also assume that the grade (G) is only influenced by the student intelligence (I) and the course difficulty (D). Only the student intelligence determines his SAT score. Finally, letter recommendation depends only on the student grade. The table in Fig. 3.2 describes the proposed domain variables and their possible values. For instance, D can take two values, *easy* or *hard*. We use the notation $Val(X)$ to denote the possible values of random variable X. Fig. 3.1 depicts the corresponding graphical structure, whose nodes represent the random variables and whose arcs represent direct influence between them.

X	$Val(X)$
Difficulty (D)	{easy,hard}
Intelligence (I)	{low,high}
SAT (S)	{low,high}
Grade (G)	{A,B}
Letter (L)	{weak,strong}
Job (J)	{no,yes}

Figure 3.2: Random variables for the student Example 3.2.3

3.2.1.2 Quantitative step

This phase concerns the probability assessment by building the so called *network parameterization* Θ, i.e., to each variable of $\mathcal{G}$, we associate a CPT that quantifies its relationship to its parents in the DAG. In the absence of the parents, we only specify a prior probability distribution. As we see, the advantage is that, instead of specifying the whole $\mathbb{P}$, we only need to specify more compact local CPTs and we are able to get significant savings, for instance in our running example, instead of storing $2^6 = 64$ entries, all we need is $2+2+2^3+2^2+2^2+2^3 = 28$ entries. Fig. 3.3 depicts arbitrary CPTs generated for the student example's random variables. Now the joint probability distribution can be obtained from the factorization product as:

$$\mathbb{P}(D, I, G, S, L, J) = \mathbb{P}(D)\mathbb{P}(I)\mathbb{P}(G|D, I)\mathbb{P}(S|I)\mathbb{P}(L|G)\mathbb{P}(J|L, S)$$

I	
low	*high*
0.5433	0.4567

D	
easy	*hard*
0.2111	0.7889

	S	
I	*low*	*high*
low	0.4995	0.5005
high	0.7268	0.2732

	L	
G	*weak*	*strong*
A	0.8935	0.1065
B	0.5584	0.4416

		G	
D	I	*A*	*B*
easy	*low*	0.3313	0.6687
hard		0.4194	0.5806
easy	*high*	0.9555	0.0445
hard		0.6997	0.3003

		J	
S	L	*no*	*yes*
low	*weak*	0.5466	0.4534
high		0.9549	0.0451
low	*strong*	0.4372	0.5628
high		0.1675	0.8325

Figure 3.3: Randomly generated CPTs for Example 3.2.3's random variables

Once constructed, a BN can be used to efficiently answer different probabilistic queries such as the computation of most probable explanations (MPE), maximum a posteriori (MAP) and the conditional probability. For instance, in the student example, $\mathbb{P}(S = low|I = high)$ can help us to update our opinion about a student score knowing he is highly intelligent. We direct the reader to Section 3.2.3 for more details about techniques for querying a distribution based on BNs.
Until now, we focused on the compact representation of $\mathbb{P}$ that is ensured using local CPTs. On top of that, the BN also encodes some local independencies, that is why we previously called $\mathcal{G}$ the independence structure. The intuition behind this semantics is that a variable depends directly only on its parents, its children and their parents. We formalize this intuition throughout the next section and we will see that the formal semantic of a BN corresponds actually to a set of conditional independence assertions.

3.2.2 Graphical Semantics

The DAG also compactly represents a set of independence statements, which are visualized through the existence/absence of arcs in the DAG. This notion is actually an intrinsic property of the induced joint probability $\mathbb{P}$ that goes beyond the simple graphical representation. It is a characteristic of probabilistic conditional independences. All the following definitions assume a given $\mathbb{P}$ that is decomposable according to a DAG $\mathcal{G}$. The latter is composed of a nodes set $\mathcal{V}$ that corresponds to a set of random variables and $\mathcal{A}$ a set of arcs between these nodes. Capital letters will denote nodes or variables and lower-case letters will denote their values or assignments. Finally, boldface letters will denote sets of variables.

Definition 3.2.4 (Probabilistic Conditional Independence [34])**.** *Let* $\boldsymbol{X, Y, Z} \subset \mathcal{V}$*, we say that* $\boldsymbol{X}$ *is conditionally independent of* $\boldsymbol{Y}$ *given* $\boldsymbol{Z}$*, w.r.t* $\mathbb{P}$*, and we write* $(\boldsymbol{X} \perp_{\mathbb{P}} \boldsymbol{Y}|\boldsymbol{Z})$*, iff:* $\mathbb{P}(\boldsymbol{X}, \boldsymbol{Y}|\boldsymbol{Z}) = \mathbb{P}(\boldsymbol{X}|\boldsymbol{Z})$*. When* $\boldsymbol{Z} = \emptyset$*, we say that* $\boldsymbol{X}$ *and* $\boldsymbol{Y}$ *are*

marginally independent and we write $(\boldsymbol{X} \perp_{\mathbb{P}} \boldsymbol{Y})$. *The variables in* $\boldsymbol{Z}$ *are often said to be observed.*

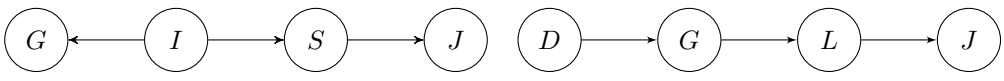
The intuition behind this definition is that two sets $\mathbf{X}$ and $\mathbf{Y}$ are conditionally independent given a set $\mathbf{Z}$ when, given our knowledge about this latter, our belief on $\mathbf{X}$ is not influenced by any knowledge we learn about $\mathbf{Y}$. Before drawing the link between this definition and the graphical representation, let us first introduce some graph theoretic notions.

Now, $X \in \mathcal{V}$ is said to be a parent of $Y \in \mathcal{V}$ if the arc (X, Y) is in $\mathcal{A}$. The set of all parents of a node X is denoted as $\mathbf{Pa}_{\mathcal{G}}(X)$. We can omit subscript $\mathcal{G}$ when there is no ambiguity about the DAG at hand.

Definition 3.2.5 (Trail)**.** $\{X_1, \ldots, X_{n+1}\} \subset \mathcal{V}$ *is said to be a trail in* $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ *if* $(X_\alpha, X_{\alpha+1})$ *or* $(X_{\alpha+1}, X_\alpha)$ *are in* $\mathcal{A}$ *for all* $\alpha \in \{1, \ldots, n\}$. *When only* $(X_\alpha, X_{\alpha+1}) \in \mathcal{A}$ *for all* α *the trail is called a (directed) path from* X_1 *to* X_n *and denoted* $X_1 - X_n$.

For instance, Fig. 3.4a (resp. Fig. 3.4b) depicts a trail (resp. directed path) obtained from the student DAG 3.1.

Definition 3.2.6 (Descendants and ancestors)**.** *Let* $X, Y \in \mathcal{V}$. *If there is a directed path from* X *to* Y, *then* Y *is said to be a descendant of* X. *We also say that* X *is an ancestor of* Y. ***Descendants****(X) (resp.* ***Ancestors****(Y)) denotes the set of the descendants of* X *(resp. ancestors of* Y*). Finally, let* ***NonDescendant****(X) define all nodes in* $\mathcal{G}$ *other than* X *and* ***Descendant****(X)*



(a) (b)

Figure 3.4: A trail (left) and a directed path (right)

Using these notations, the already mentioned independence statements can be encoded graphically by the BN as follows:

Definition 3.2.7 (Markovian Assumption)**.** *In the BN* $(\mathcal{G}, \mathbb{P})$*, every node is conditionally independent of its non-descendants given its parents in DAG* $\mathcal{G}$*:*

$$\forall X \in \mathcal{V}, (\{X\} \perp_{\mathbb{P}} \boldsymbol{NonDescendant}(X) | \boldsymbol{Pa}(X)) \tag{3.2}$$

For instance, the Markovian assumption states that $\{L\} \perp_{\mathbb{P}} \{S, I, D\} | \{G\}$ in Fig. 3.5 It is important to emphasize that one can prove that having a joint distribution $\mathbb{P}$, which obeys Markovian Assumption 3.2 is equivalent to say that $\mathbb{P}$ satisfies the chain rule of BNs 3.1 [25]. This also implies that every independence statement derived using such an assumption is necessarily satisfied by $\mathbb{P}$. However the inverse is not always true, there might be other independences that are only satisfied by $\mathbb{P}$ and not expressed by the Markov property.
Given a variable $X \in \mathcal{V}$, only a subset of variables, called the Markov blanket of X, is required to update our knowledge about X.

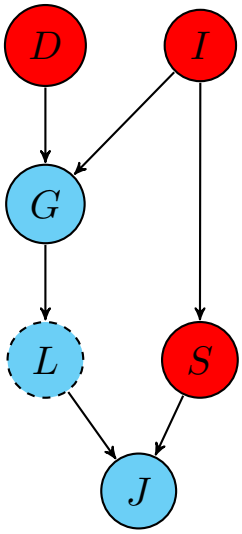


Figure 3.5: Markovian assumption for node L

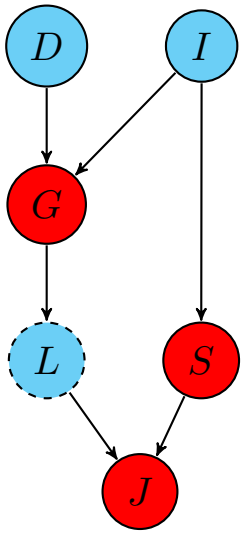


Figure 3.6: The Markov blanket for node L

Definition 3.2.8 (Markov Blanket)**.** *Given a DAG $\mathcal{G} = (\mathcal{V}, \mathcal{A})$, the Markov blanket of $X \in \mathcal{V}$, denoted $\boldsymbol{MB}_{\mathcal{G}}(X)$ is defined as the union of the parents of X, of the children of X and of these children's parents.*

This means that any instantiation $Y = y$ such that $Y \notin \mathbf{MB}_{\mathcal{G}}(X)$ does not impact our belief on X once we know the values of the variables in $\mathbf{MB}_{\mathcal{G}}(X)$. In other words, X is separated from the rest of the network by $\mathbf{MB}_{\mathcal{G}}(X)$. For instance, in the student DAG 3.6, the Markov blanket of variable L (dashed node) is composed of the red nodes, i.e., $\mathbf{MB}_{\mathcal{G}}(L) = \{G, J, S\}$

In this chapter, we restrict ourselves to the independences that are derived from the DAG using Assumption 3.2. We refer to [34, 97], where one can be guided by set of properties called the *graphoid axioms* for conditional independence, to infer additional independences from $\mathbb{P}$.

Using these axioms to find conditional independences may be fastidious. Fortunately, there exists a graphical test called *d-separation (directional separation)* [102] that allows us to efficiently detect the independences implied by these axioms and those entailed by the Markov assumption. This test is intuitive and is

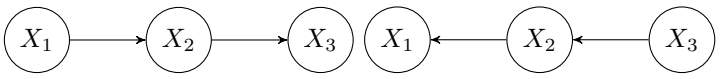
much easier to exploit by human reasoning than the graphoid axioms. Given three disjoint sets of nodes $\mathbf{X,Y,Z} \subset \mathcal{V}$, the *d-separation* criterion graphically detects whether $\mathbf{X}$ is independent of $\mathbf{Y}$ given $\mathbf{Z}$ by testing whether every trail between $X \in \mathbf{X}$ and $Y \in \mathbf{Y}$ is blocked by $\mathbf{Z}$. Hence, let us first define the notion of blocked trails.

Moreover, the notion of blocked trail helps us to see the probabilistic influence as a flow of information in the graph. Given the set of nodes $\mathbf{Z}$, a trail is blocked whenever it contains a closed two-arc trail given $\mathbf{Z}$, otherwise the trail is said to be active. So, what is a closed two-arc trail? Depending on the type of the two-arc trail, we will be able to determine whether the trail is closed or not. Consider a two-arc trail $\{X_1, X_2, X_3\}$, this can have three forms:

1. Sequential:

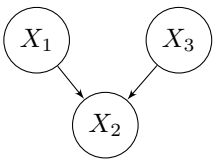
   (a) $X_1 \to X_2 \to X_3$ when $\{X_1\} \subseteq \mathbf{Pa}(X_2)$ and $\{X_2\} \subseteq \mathbf{Pa}(X_3)$.

   (b) $X_1 \leftarrow X_2 \leftarrow X_3$ when $\{X_3\} \subseteq \mathbf{Pa}(X_2)$ and $\{X_2\} \subseteq \mathbf{Pa}(X_1)$.



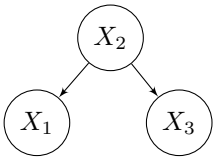
(a) (b)

2. Convergent, aka *v-structure* $X_1 \to X_2 \leftarrow X_3$ when $\{X_1, X_3\} \subseteq \mathbf{Pa}(X_2)$.



3. Divergent, $X_1 \leftarrow X_2 \to X_3$ when $\{X_2\} \subseteq \mathbf{Pa}(X_3)$ and $\{X_2\} \subseteq \mathbf{Pa}(X_1)$.

Then, table 3.1 summarizes when a two-arc trail is said to be closed given a node set $\mathbf{Z}$. For instance, in Fig. 3.7 trail $\{X, X_1, X_2, X_3, X_4, Y\}$ is blocked given



a trail $\{X_1, X_2, X_3\}$	closed given $\mathbf{Z}$?
Sequential and Divergent	$X_2 \in \mathbf{Z}$
Convergent	$\{X_2\}, \mathbf{Descendant}(X_2) \not\subset \mathbf{Z}$

Table 3.1: Conditions under which a two-arc trail is closed

$\mathbf{Z} = \{X_2, X_3\}$, since it contains the closed two-arc trail $\{X_2, X_3, X_4\}$, which is divergent and $X_3 \in \mathbf{Z}$ Now we have all the ingredient to provide a definition of *d-separation*.

Definition 3.2.9 (*d-Separation* [100])**.** *Let* $\mathbf{X}, \mathbf{Y}$ *and* $\mathbf{Z}$ *be disjoint sets of nodes in* $\mathcal{G}$*. We say that* $\mathbf{X}$ *and* $\mathbf{Y}$ *are d-separated by* $\mathbf{Z}$*, written* $\mathbf{X} \perp_{\mathcal{G}} \mathbf{Y} | \mathbf{Z}$*, iff every path between a node in* $\mathbf{X}$ *and a node in* $\mathbf{Y}$ *is blocked by* $\mathbf{Z}$*.*

For instance, in Fig. 3.7, we have $\{X\} \perp_{\mathcal{G}} \{Y\} | \{X_2, X_3\}$.
Two variables that are not d-separated are said to be *d-connected*. The purpose of the d-separation test is to provide a simple and yet efficient graphical criterion to assert whether two variables are probabilistically dependent given our knowledge about other variables. So far, we have been guided by an intuitive approach based on the influence propagation, but one may wonder whether for every two nodes X, Y d-separated by some set $\mathbf{Z}$, we have X is conditionally independent from Y

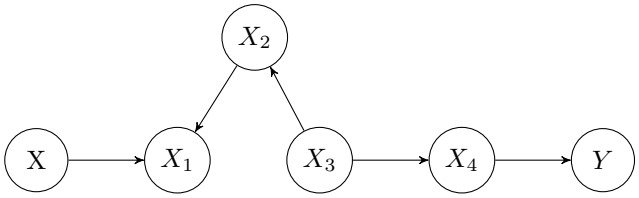


Figure 3.7: An example of a blocked trail between X and Y

given $\mathbf{Z}$, w.r.t $\mathbb{P}$. This is the purpose of the following proposition:

Proposition 3.2.10 (soundness of d-separation [129]). *Let* $\boldsymbol{X}$, $\boldsymbol{Y}$ *and* $\mathbf{Z}$ *be disjoint sets of nodes in* $\mathcal{G}$. *Then* $\boldsymbol{X} \perp_{\mathcal{G}} \mathbf{Y} | \mathbf{Z} \Rightarrow \mathbf{X} \perp_{\mathbb{P}} \mathbf{Y} | \mathbf{Z}$.

One legitimate question concerns the completeness of d-separation, that is, we want to know whether the criterion can detect conditional independences each time they hold for $\mathbb{P}$. Unfortunately, the answer is negative. However, we have the following weaker result:

Theorem 3.2.11 (Weak completeness). *For every DAG* $\mathcal{G}$, *there exists a joint probability distribution* P *factorizing over it, s.t.,* $\boldsymbol{X} \perp_{\mathcal{G}} \boldsymbol{Y} | \mathbf{Z} \Leftrightarrow \boldsymbol{X} \perp_{P} \boldsymbol{Y} | \mathbf{Z}$.

This weaker notion of completeness is telling us that one cannot improve on the d-separation test, i.e., this criterion discovers all the independences that one can derive from the DAG [47]. As a consequence, the set of independences that is expressed by the Markovian assumptions are the maximum that we can derive from the independence structure. In other words we have the following theorem:

Theorem 3.2.12 (Dependent variables [31]). *Given a DAG* $\mathcal{G}$, $\mathbf{X}$ *and* $\mathbf{Y}$ *are dependent given* $\mathbf{Z}$ *in some distribution* $\mathbb{P}$ *that factorizes over* $\mathcal{G}$ *if* $\mathbf{X}$ *and* $\mathbf{Y}$ *are not d-separated given* $\mathbf{Z}$ *in* $\mathcal{G}$.

Finally, we would like to emphasize that there exist other equivalent and yet simpler methods to test independences in BNs, such as Markov separation [77] and *u-separation* [23].

3.2.3 Reasoning with BNs

A BN can be considered as a probabilistic knowledge base, in which the process of querying is called inference. In fact, inference is the main task behind constructing a BN. The general form of the probabilistic inference is $\mathbb{P}(\mathbf{Q}|\mathbf{E}{=}\mathbf{e})$, which reflects

the following human reasoning question: suppose we observe $\mathbf{e}$ on a subset of $\mathbf{E}$ of the domain, what are the probability values that another subset $\mathbf{Q}$ can take? $\mathbf{Q}$ is called the query or target and $\mathbf{E}$=$\mathbf{e}$ is called an observation or evidence.

3.2.3.1 Querying the distribution

We present here the most common types of probabilistic queries:

Marginal distribution can be viewed as the projection of $\mathbb{P}$ on a smaller subset of variables by marginalizing out over the other variables. When $\mathbf{Q}, \mathbf{E} \neq \emptyset$, the query is called the *posterior marginal* of $\mathbf{Q}$ given the knowledge on observed variables $\mathbf{E}$. It is called also the conditional probability query. In the sequel, $\mathbf{x}$ will be used to denote the assignment $\mathbf{X} = \mathbf{x}$. Answering the marginal distribution query involves the computation of both the joint distributions $\mathbb{P}(\mathbf{Q}, \mathbf{e})$ and $\mathbb{P}(\mathbf{e})$ as :

$$\mathbb{P}(\mathbf{Q}|\mathbf{e}) = \frac{\mathbb{P}(\mathbf{Q}, \mathbf{e})}{\mathbb{P}(\mathbf{e})} \tag{3.3}$$

Both distributions can be computed by marginalizing out the variables in $\mathbf{W} = \mathcal{V} \setminus \mathbf{Q} \cup \mathbf{E}$. In this case, the numerator can be computed by summing out over $\mathbf{W}$, $\mathbb{P}(\mathbf{Q}, \mathbf{e}) = \sum_{\mathbf{w} \in \mathbf{W}} \mathbb{P}(\mathbf{Q}, \mathbf{e}, \mathbf{w})$, while the denominator, which is called the probability of evidence, can be deduced from the previous equation by

$$\mathbb{P}(\mathbf{e}) = \sum_{\mathbf{q} \in \mathbf{Q}} \mathbb{P}(\mathbf{q}, \mathbf{e})$$

When $\mathbf{E} = \emptyset$, the distribution $\mathbb{P}(\mathbf{Q})$ is called *prior marginal*.

Most Probable Explanation (MPE)

This class of probabilistic queries tries to find the most probable instantiation of query variables $\mathbf{Q} = \mathcal{V} \setminus \mathbf{E}$ given the evidence $\mathbf{E} = \mathbf{e}$. To put it another way,

this amounts to find the assignment of $\mathbf{Q}$ that maximizes $\mathbb{P}(\mathbf{Q}|\mathbf{e})$ i.e., computing the quantity :

$$argmax_{\mathbf{q}\in\mathbf{Q}}\mathbb{P}(\mathbf{q}|\mathbf{e})$$

Maximum A Posteriori (MAP)

This is a more general case in the sense that while $\mathbf{Q}$ covers all non evidence variables in MPE, it is now just a subset of variables for which we therefore seek to find a high-probability joint assignment. It is also called the marginal MAP, because this involves both marginalizing out some variables and computing some ArgMax on others. More precisely, if $\mathbf{W} = \mathcal{V} \setminus \{\mathbf{Q} \cup \mathbf{E}\}$, this amounts to compute:

$$argmax_{\mathbf{q}\in\mathbf{Q}} \sum_{\mathbf{w}\in\mathbf{W}} \mathbb{P}(\mathbf{q}, \mathbf{w}|\mathbf{e})$$

Now if we come back to student Example 3.2.3, and rely on one of the probabilistic inference algorithms, which will be detailed in the next section, we can predict how likely an intelligent student would get the job by answering the marginal distribution $\mathbb{P}(J|I = high)$, where $\mathbf{Q} = \{J\}$ and $\mathbf{E} = \{I = high\}$. Note that in general, the prediction flow of information is in the same direction as the BN arcs. Another query would try to see the impact of knowing the student got the job on our knowledge about his grade. This could be done through computing $\mathbb{P}(G|J = yes)$. This kind of reasoning is called diagnosis and the information flow is in the opposite direction of the BN arcs. Observing that the student got an A grade, we would like to know what is the most probable assignment of variables D and S that allows such a situation. This is done by answering the query $MAP(D, S|G = A)$ and if we add J and L to the targets, the query will then reduce to an MPE.

3.2.3.2 Probabilistic inference

Several algorithms were proposed in the literature to answer efficiently and reduce computations amount of the previous queries. By essence, the inference problem is simple. However, as we noticed, the challenging part is the sum computations over combinations of values, whose number is exponential in the number of variables. A straightforward computation of the sum is therefore only feasible when this number is not very large, otherwise the problem becomes intractable. Broadly speaking, we can classify inference algorithm into two categories. First, exact inference algorithms, including elimination and conditioning algorithms. Second, approximate inference algorithms.
Historically, the first exact algorithm to get every node's marginal probabilities has been proposed in [98] and was valid only for trees. Its extension to poly-trees, also called singly- connected networks, was proposed in [69, 100] and was known as the *poly tree algorithm*. The algorithm is based on a message-passing protocol, in which a node is entitled to propagate a message only after it has received messages from its neighbors except from the receiving node. In viewing the BN as a causal structure, every distribution over a node is viewed as a *belief* in that node, that is why it is common to call this algorithm *belief propagation*. Note that its complexity is linear in the size of the network. The two main approaches to extending this algorithm to general (multiply-connected) networks are the cycle-cut set algorithm, also known as loop cutset conditioning [99, 102], and join or junction tree algorithms [78].

The first approach tries to reduce the structure to a poly-tree, where it is possible to apply the poly-tree algorithm, that is, the algorithm tries to solve the problem under the condition of instantiating some variables and repeating this for all possible values of the variable. It is clear that this technique yields to multi-

ple calls of the poly-tree algorithm since conditioning on enough variables (loop cut set) makes the structure a tree The final solution can then be re-constructed by aggregating the computations performed from each poly-tree algorithm call. A more advanced technique known as *recursive conditioning*, tries to split recursively the network into disconnected sub-networks to be solved by classic conditioning. Unfortunately, this kind of inference algorithm is known to be very time consuming and might yield to higher complexity in some complex networks, even if we can control such a complexity in general.

In another direction, a second approach called the join tree algorithm was proposed in [78, 83, 120] and became one of the most popular exact inference algorithms. Basically, this algorithm transforms the initial BN into a corresponding join tree, which is a secondary structure obtained from clustering the BN nodes into cliques. Probabilistic inference is then based on passing *messages* through the join tree. This algorithm proves to be efficient and is considered one of the state of the art algorithm. That is why it is the most widespread algorithm to implement exact inference in practice. Another alternative is the *variable elimination algorithm* [36, 137], which is based on the simple technique of eliminating non query and evidence variables. The algorithm initializes a set of factors with the BN CPTs, then, each time a variable needs to be eliminated, the product of factors that mentions that variable is computed. The variable is then marginalized out from the combined factor. By repeating this elimination process until no more variables need to be removed, one can obtain the marginal distribution over any subset of variables. The performance of such an algorithm is very related to the elimination order of the variables and finding an optimal elimination order turns out to be is a NP-Hard problem [17, 70].

In general, it is known that the probabilistic inference is a NP-hard problem [24, 26]

and it turns out that variable elimination and join tree algorithms are very related[3] and have the same space and time complexities, which are exponential in the tree-width, i.e., the size of largest cluster in the join tree minus one.
The join tree algorithm has some attractive properties. On one hand, it can be seen as a sophistication of variable elimination algorithm that allows simultaneous eliminations of many variables. On the other hand, it is able to compute multiple marginals at once [32]. These properties are interesting for our context, where we deal with multi-target systems. That is why this thesis is particularly interested in the join tree algorithm. We refer to the next Chapter (4) for a detailed description of how such an algorithm works. But now let us emphasize the fact that there exist also many other approaches to inference different from the aforementioned techniques. In particular efforts have been done to exploit the BN local structure in order to represent more compactly factors and speed up the inference, see for example [8, 11, 30, 116]. Even if these methods offer a promising ground for improving inference, they suffer from theoretical justifications regarding their complexity and the savings they might offer [32].

Finally, it turns out that exact inference can be intractable and in this case various approximation methods try to find iteratively an approximation from random samples of instantiations. We recall briefly some of these techniques, as this subject is not treated in this thesis. For example, belief propagation can be used to search an approximation of the probability distribution and then uses optimization techniques to minimize an error function [69, 102]. There are also other popular methods of approximate inference that are based on stochastic sampling, that is, estimating the probability using a frequentist approach. The most famous ones are Markov Chain Monte Carlo methods, which include Gibbs sampling and Metropo-

[3]Intuitively, junction tree algorithm consists of performing variable elimination in all directions at once and storing intermediate results at each step.

lis Hastings algorithms [101, 111] that are used for generating samples from the posterior distribution. Importance Sampling is another approach that randomly samples another distribution and modifies these samples in such a way that they provide a good estimate of the distribution of interest, see, e.g., Likelihood weighting [118]. For a good grasp on these methods, the reader might find more details in [72] or [31].

3.3 Probabilistic Relational Models

When dealing with complex and large systems, BNs do not seem to be well adapted. Indeed, the cost of designing and maintaining such systems becomes high [73, 84] and the inference might be intractable. A complex system may induce a big number of probabilistic dependencies that can, not only lead to intractable modeling costs, but also implies important update costs when changes occur in the underlying structure. Actually, every time an entity and/or a relation in the domain changes, a new BN should be defined. Such shortcomings are essentially due to the fact that BNs completely ignore the concept of objects and their interactions. To cope with the aforementioned problems several extensions were proposed in the literature, among which, one can find the following:

- Object-oriented extension, such as Object-Oriented BNs [12, 73, 84]
- First Order logic extension, such as Markov Logic Network [66, 68]
- Relational models such as Entity-Relationship [60, 76].

In this thesis, we are particularly interested in Probabilistic Relational Models (PRMs) [12, 106], an extension of the BNs attribute-based representation that combines the OO paradigm with the relational representation in database theory. Hence, PRMs specify a generic template defining a probabilistic model, a set of dependencies, that hold in a relational domain, which can be represented by a

relational database. The OO paradigm helps to abstract and capture general properties about similar entities in the domain in terms of classes having similar properties and can be instantiated for specific contexts as many times as suited. While the relational model helps to describe interactions or relationships between classes and their instantiated objects. To understand these concepts and give more formal definitions, we use the PRMs version described in [127].

For the illustration purpose, let us consider Fig. 3.8, which depicts a BN. In such a configuration, one can identify repeated patterns and abstract them as a generic type or *class* of objects. For instance, in Fig. 3.8, random variables X_i, Y_i form a pattern. A PRM class also abstracts all the interactions between its random variables by drawing arcs expressing the probabilistic dependencies. Class' random variables, which describe and characterize the pattern represented by the class, are called *attributes*. Basically, a class attribute can only be visible to attributes of the same class. However, one might want to refer to attributes outside the class. That is why PRMs define the notion of reference slots allowing communications between different classes and hence between attributes of different classes. Now let us give formal definitions for PRM concepts

Definition 3.3.1 (Class)**.** *A class $\mathcal{C}$ is defined by a DAG over a set of attributes $\mathbf{A}(\mathcal{C})$ and a set of reference slots $\mathcal{R}(\mathcal{C})$ (see Definition 3.3.2).*

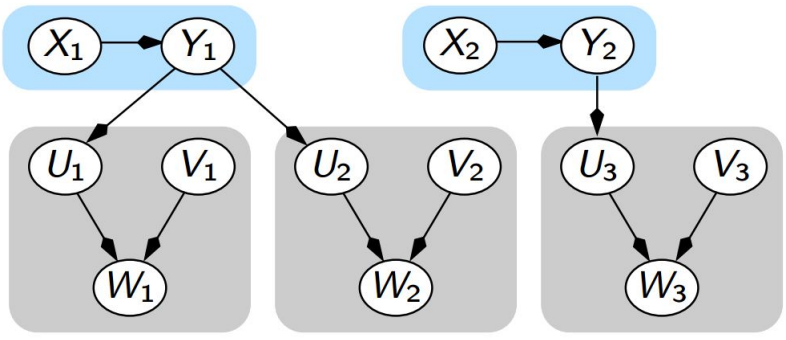


Figure 3.8: A BN with repeated patterns

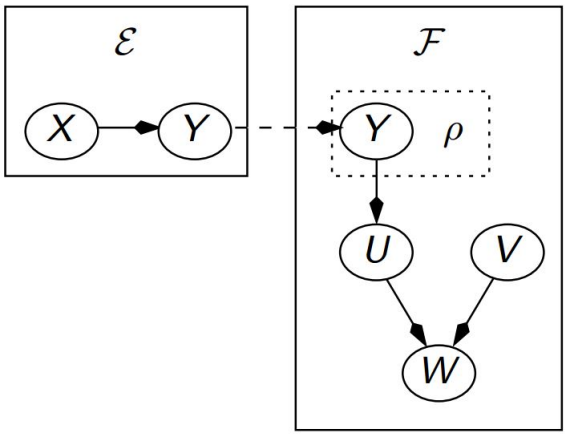


Figure 3.9: Abstraction of the repeated patterns of Fig 3.8 as classes

With this definition, a class represents a template for a fragment of a BN over its attributes. To refer to a given attribute X (resp. reference ρ) of class $\mathcal{C}$, we use the standard OO notation $\mathcal{C}.X$ (resp. $\mathcal{C}.\rho$).

Definition 3.3.2 (Reference slots)**.** *Let $\mathcal{C}$ and $\mathcal{D}$ be two classes. A reference slot $\mathcal{C}.\rho = \mathcal{D}$ is a pointer in $\mathcal{C}$ that refers to $\mathcal{D}$ and allows to access elements of $\mathcal{D}$ through $\mathcal{C}$. We say that $\mathcal{C}$ is the domain of $\mathcal{C}.\rho$ denoted $domain(\mathcal{C}.\rho)$ and $\mathcal{D}$ is the range of $\mathcal{C}$, denoted $range(\mathcal{C}.\rho)$. A reference slot is simple if it relates one class to one class and it is complex if it relates one class to many. The inverse of $\mathcal{C}.\rho$ is called inverse slot and is denoted $\mathcal{D}.\rho^{-1}$. We have $range(\mathcal{D}.\rho^{-1}) = domain(\mathcal{C}.\rho)$ and $domain(\mathcal{D}.\rho^{-1}) = range(\mathcal{C}.\rho)$.*

For instance, Fig 3.9 depicts the two classes obtained by abstracting the repeated patterns in Fig 3.8 and the corresponding reference slot (dashed parts). The mechanism of reference slot can be applied recursively to allow one attribute to access attributes of different classes by navigating through other classes, we call this path of reference a slot chain.

It is important to highlight that class attributes are a generic template for random variables, and in that way they differ from their instantiations that are

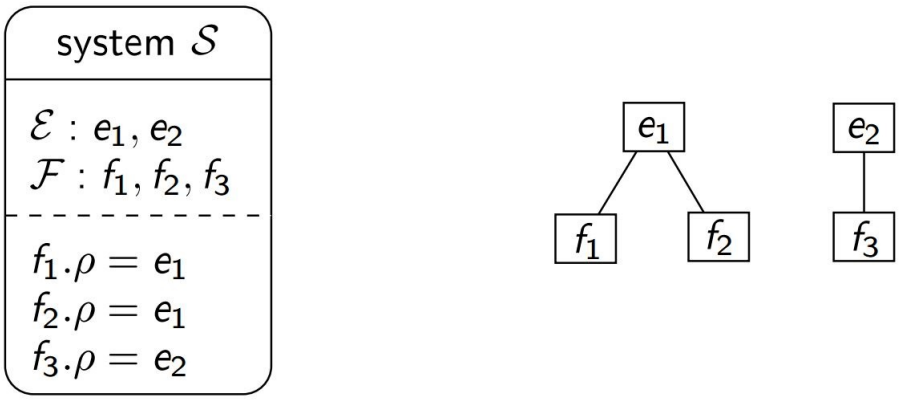


Figure 3.10: The relational skeleton related to the BN in Fig 3.8

actually distinct random variables.

Definition 3.3.3 (Instance)**.** *An instance c of class $\mathcal{C}$ is an actual object of the the class $\mathcal{C}$, i.e., a BN fragment whose attributes are random variables generated from the class level template and where reference slots refer to sets of their range's instances.*

Similarly, given an instance i, $\mathbf{A}(i)$ denotes the set of the instantiated attributes of instance i, which are random variables. We call a set of instances linked together using reference slots a relational skeleton and we refer to its graphical representation as the instance graph. Given an instance i and a reference slot $\rho \in \mathcal{R}(i)$, we denote by $range(i.\rho)$ the set of instances connected to i through ρ.

Definition 3.3.4 (Relational Skeleton)**.** *A relational skeleton $\mathcal{S}$ is a set of instances such that for any instance c of class $\mathcal{C}$ and any reference slot $\mathcal{C}.\rho = \mathcal{D}$, there exists at least one instance $d \in \mathcal{S}$ such that d is an instance of $\mathcal{D}$ and $d \in range(c.\rho)$.*

The BN shown in Fig 3.8 can be compactly represented by the system in Fig 3.10 with the corresponding relational skeleton. It may happen that, for two instances i_1 and i_2 of a same class $\mathcal{C}$, the number of parents of random variable

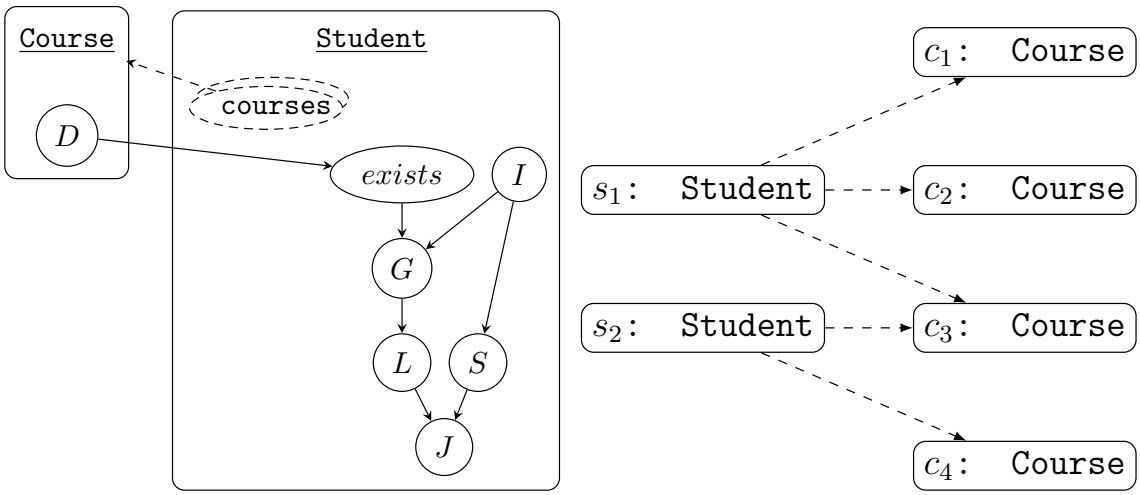
$i_1.X$ differs from that of $i_2.X$. This is the case, for example, when $i_1.X$ and $i_2.X$ refer to SAT scores of two different students who did not follow the same courses, the parents of these variables being these courses. In this case, the precise set of parents of an attribute cannot be determined precisely at class level, which implies that a standard CPT cannot be provided for v ariable X in Class $\mathcal{C}$. To cope with this kind of situation, we need to specify a generic function capable of encoding at class level X's CPT whatever the number of parents in the instances of $\mathcal{C}$. For this reason, PRMs define *aggregators* to express such a many-to-one instance relation. In addition, aggregators are useful to keep the probabilistic semantic and genericity at class level and avoid otherwise multiple class definitions w.r.t. the variable number of configurations depending on relation arities, i.e., the size of the multiple slot.

Definition 3.3.5 (Aggregator)**.** *An aggregator is a deterministic functional attribute, whose CPT is inferred once the class is instantiated and connected to its instantiated references.*

Examples of classical aggregators include *min, max, for all* and *exist*. Let us present an illustration of these concepts and consider the updated Student Example 3.3.6.

Example 3.3.6 (Updated Student Example)**.** *In Example 3.2.3, the student was related to one course. Now, we consider the student attending several courses. Based on the remark that the course difficulty does not characterize a student, but rather is a course property, it would be interesting to divide the domain variables into two types. The first one gathers properties that describe a student and the second one gathers those describing a course.*

From the description in Example 3.3.6, It would be natural to define two classes `Student` and `Course`. Fig. 3.11a depicts such a situation and represents class'



(a) Class definitions and their dependencies (b) A possible system

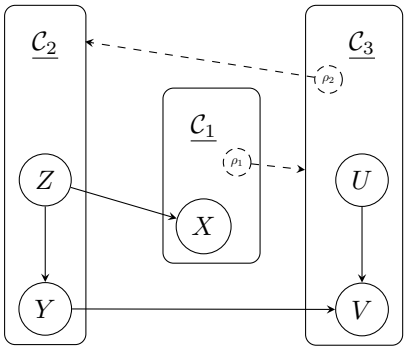
Figure 3.11: some PRM concepts for Example 3.3.6

references using a dashed arrow. While the former has six attributes and a multiple reference slot (dashed node), the latter has only one attribute. In this scenario, we assume that our belief on the grade variable changes if the student subscribes to at least one *hard* course. To express this, we introduce the *exists* aggregator, which returns true if the student is related to at least one *hard* course. To get an intuition of how such an aggregator works, let us suppose that we have a system composed of three `Course` instances (c_1, c_2, c_3) and one `Student` instance (s_1, s_2) as the relational skeleton in Fig. 3.11b shows. In this case, the *exists* aggregator CPT is shown in Fig. 3.12a. When an attribute of a class $\mathcal{C}_1$ depends on another attribute in class $\mathcal{C}_2$ such that $\mathcal{C}_1$ and $\mathcal{C}_2$ are connected through a third class $\mathcal{C}_3$, a slot chain is involved to allow such a referencing:

Definition 3.3.7 (Slot chains)**.** *A slot chain $\mathcal{C}.K$ is a sequence of reference and/or inverse slots $\rho_1, \rho_2, \ldots, \rho_n$ such that $domain(\rho_1) = \mathcal{C}$ and $\forall i \in \{1, \ldots, n-1\}$ $range(\rho_i) = domain(\rho_{i+1})$. We say that $range(\rho_n)$ is the range of $\mathcal{C}.K$ and $domain(\rho_1)$ its domain.*

			s_1.exists	
c_1.D	c_2.D	c_3.D	0	1
easy	*easy*	*easy*	1.00	0.00
hard			0.00	1.00
easy	*hard*		0.00	1.00
hard			0.00	1.00
easy	*easy*	*hard*	0.00	1.00
hard			0.00	1.00
easy	*hard*		0.00	1.00
hard			0.00	1.00

(a) A possible CPT for the exists aggregator



(b) An example of a slot chain relating $\mathcal{C}_1$ and $\mathcal{C}_2$ through C_3

Figure 3.12: An example of an aggregator CPT and a slot chain

Fig. 3.12b illustrates such a situation, where attribute $\mathcal{C}_1.X$ refers its parent $\mathcal{C}_2.Z$ through the slot chain $\mathcal{C}_1.\rho_1.\rho_2 = \mathcal{C}_2$.

Now, we have all the necessary ingredients to define formally PRMs. Given a relational skeleton $\mathcal{S}$ and a class $\mathcal{C}$, let $\mathcal{I}_{\mathcal{S}}(\mathcal{C})$ denote the set of instances of $\mathcal{C}$ in $\mathcal{S}$.

Definition 3.3.8 (PRM)**.** *A PRM* Π *is a pair* $(\mathscr{C}, \mathcal{S})$*, where* $\mathscr{C}$ *is a set of classes and* $\mathcal{S}$ *is a relational skeleton. It encodes the joint probability distribution over* $\mathbf{A}(\mathcal{S})$*, the set of all instance attributes in* $\mathcal{S}$*, as the following product:*

$$\mathbb{P}(\mathbf{A}(\mathcal{S})) = \prod_{\mathcal{C}\in\mathscr{C}} \prod_{i\in\mathcal{I}_{\mathcal{S}}(\mathcal{C})} \prod_{i.X\in\mathbf{A}(i)} \mathbb{P}(i.X|\mathbf{Pa}(i.X)) \tag{3.4}$$

Definition 3.3.9 (Ground BN [48])**.** *Given a PRM* $\Pi = (\mathscr{C}, \mathcal{S})$*, its associated ground Bayesian Network is a BN constructed using the following steps:*

1. *There is a node for every attribute* $i.X$ *of every instance* $i \in \mathcal{S}$*, named* $i.X$*.*
2. *Each* $i.X$ *depends probabilistically on parents of the form* $i.Y$ *or* $j.Y$ *such that there exists a slot chain* K *with* $j \in range(i.K)$*.*
3. $i.X$*'s conditional probability distribution is a CPT generated from the attribute's CPT of the corresponding class.*

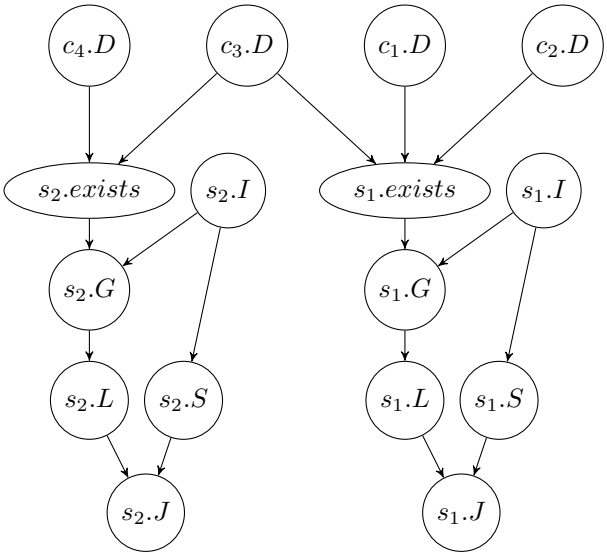


Figure 3.13: The ground BN for the system in Fig. 3.11b

Fig. 3.13 depicts the ground BN for system in Fig. 3.11b.

3.4 Conclusion

In this chapter we introduce BNs, a well-established framework to represent and reason under uncertainty in IA. Furthermore, we present some of their graphical properties, notably the graphical independences they encode and we relate this to the conditional independences of the induced joint probability distribution. We also give a graphical criterion, called d-separation, that reveals all independence statements from the BN structure. Finally, observing that BNs might suffer from some limitations, we present PRMs, an object-oriented extension of BNs, that allow BNs to scale up to complex large systems.

We previously saw in Chapter 2 that techniques used to quantify uncertainties in the RBS domain have theoretical and practical limitations. We believe that this could be overcome by exploiting a probabilistic reasoning. In this context, BNs

could prove to be useful, particularly their PRM extension. The reason why we are interested in PRMs is twofold. First, PRMs provide a mathematically sound framework and integrate the OO paradigm and relational models. This allows BNs to scale up and their inference to speed up. Second, modern BRMSs are based on the OO paradigm and their knowledge bases have several connections with relational databases. Additionally, probabilistic inference engines, notably those based on the junction tree algorithms, seem to be good candidates to speed-up the rules inference, as they allow querying multiple targets at the same time. However, the rule engine queries of the WM are, by essence, incremental and frequent, hence, the probabilistic inference shall also be performed incrementally. In the following chapter, we investigate a new approach to extend BNs inference to the context of modern BRMSs.

Chapter 4

Incremental Junction Tree Inference

4.1 Introduction

In a BRMS, interactions with the WM are dynamic and incremental. Moreover, the WM is by essence, multi-targets, which means that it contains many query variables. As a consequence, if we are to integrate a probabilistic feature based on the PGM framework, the probabilistic inference has to take this property into account. However, this is a challenging task in general. Actually, when the system, its evidence and/or its targets evolve, most of probabilistic inference algorithms either recompute everything from scratch, even though incremental changes do not invalidate all the previous computations, or they do not fully exploit incrementality to minimize computations. The very idea of taking advantage of previous computations to optimize the current ones is not new. For example, [27] defines an incremental system that can make use the results of previous computations to reduce the cost or improve the quality of results for subsequent computations. Four incrementality criteria relevant to probabilistic inference were then identi-

fied: incrementality w.r.t (time) resource, queries, evidence and representation. In this chapter, we are interested in all these aspects, especially in the last three ones. Surprisingly, very few inference algorithms address all these aspects. For example, [83] exploits partially the incrementality, but it is far from optimal when the set of targets is smaller than the set of all the random variables or when this set changes. In [29], for instance, the query point of view is taken into account by reconfiguring dynamically some join trees (JT) when queries change but the *BN* structure is assumed to remain static, which may not necessarily be the case in rule-based systems. In [81], the authors exploit relevance-based reasoning to identify the parts of the network that are relevant for computations and, then, update several subnetworks whose union covers the original one. Unfortunately, this algorithm does not take into account computations performed previously. In [83], an incremental JT-based inference algorithm has been proposed that exploits independences induced by incremental evidence updates. But the JT structure never evolves and it is assumed that all the nodes are targets, which is not optimal in our context. On the opposite, the incremental JT structure is addressed in [44] but not the queries incrementality nor the exploitation of previous probabilistic computations.

Another direction is investigated in [80], where authors argue that compiling the original *BN* into a conjunctive normal form (CNF) coupled with caching techniques improves inference when the network structure is updated. But this does not take optimally into account evidence and queries. Along similar lines and in the context of logic programs, [130] proposes an online inference that allows updating the set of facts while keeping the set of (logic) rules intact. At the end, the process comes down to adding or removing clauses in the corresponding logic program. This is obviously a non PGM-based approach that deals only with a particular case of incrementality, that which is of evidence. Indeed, our framework

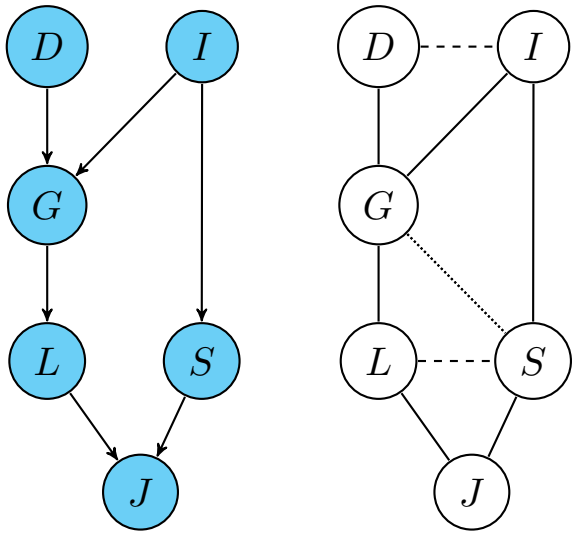
is to be integrated within a BRMS where the rules have side-effects on the facts in opposition with logic programs.

This chapter tries to avoid the previous limitations, as they may incur strong unnecessary overheads when the system under study is large. In order to alleviate the question of incrementality, we introduce a new junction tree-based message-passing inference algorithm that, given a new query, minimizes computations by identifying precisely the set of messages that differ from the preceding computations. Additionally, our algorithm takes advantage of computations sharing within junction tree algorithms to answer multiple queries.

In the sequel, $\mathcal{B} = (\mathcal{G}, \Theta)$ stands for a BN and for any graph G, we use the notation $\mathcal{V}(G)$ and $\mathcal{E}(G)$, to express the set of its vertices/nodes and edges respectively. Finally, $\mathcal{T}$ will refer to a junction tree (see below) associated with $\mathcal{B}$.

4.2 Junction Tree algorithm

This section provides a concise overview of the join tree algorithm, where inference is based on a message-passing algorithm within the so called secondary structure called junction tree (JT). Let us first present a general description of the algorithm before introducing formal definitions. The JT of $\mathcal{B}$ is obtained after converting DAG $\mathcal{G}$ into an undirected graph as follows. First, for each node in $\mathcal{V}$, we add edges between all of its parents, this step is called *moralization* and its result is the *moral* graph. Second we remove the orientations of the remaining arcs and we add edges in the resulting undirected graph so that, in every cycle of at least four nodes, there exists a pair of nodes non-adjacent in the cycle but adjacent in the undirected graph; this step is called the *triangulation* and the resulting undirected graph is called the *triangulated* graph. We recall the DAG from of Student Example 3.2.3 in Fig. 4.1a. Fig. 4.1b illustrates the edges added during moralization (dashed



(a) (b)

Figure 4.1: A DAG (a) and its corresponding moral then triangulated graph (b)

edges) and triangulation (dotted edges). Based on the cliques of the triangulated graph, i.e., its maximal complete subgraphs, we will form the nodes of the JT. As we see, the JT definition relies on the notion of graph of cliques, hence we first provide a definition for cliques:

Definition 4.2.1 (clique)**.** *A complete sub-graph of an undirected graph G is a sub-graph in which every pair of distinct nodes are connected. A clique $\mathbf{C}_i$ of an undirected graph G is a maximal complete sub-graph of G, i.e., there does not exist any complete sub-graph that strictly contains it.*

For instance, $\mathbf{C}_1 = \{L, S, J\}$ is a clique in the graph of Fig. 4.1b. Then a clique graph is defined as follows:

Definition 4.2.2 (Clique graph)**.** *A clique graph $\mathcal{U}$ for an undirected graph G is an undirected graph such that each of its nodes i is mapped to a clique $\mathbf{C}_i$ of G. In addition, for every pair of nodes i and j such that $\mathbf{C}_i \cap \mathbf{C}_j \neq \emptyset$, i and j must belong to the same connected component, i.e., there must be an undirected path linking these nodes in $\mathcal{U}$.*

If we use natural numbers to represent $\mathcal{V}(\mathcal{U})$, then $\mathbf{C}_i$ represents the contents of ith node. We also call $\mathcal{V}(\mathcal{U})$ and $\mathcal{E}(\mathcal{U})$ clusters and separators respectively.

In a JT based algorithm, $\mathcal{U}$ must satisfy the so called *running intersection property* to be be ready for inference.

Definition 4.2.3 (Running intersection property)**.** *Let $\mathcal{T}$ be a clique tree. We say that $\mathcal{T}$ satisfies the running intersection property if any non empty intersection of two cliques belongs to every clique in the unique path between those two cliques.*

Since $\mathbf{C}_i \cap \mathbf{C}_j \neq \emptyset$, for $(i, j) \in \mathcal{E}(\mathcal{U})$, we call such intersection a separator, noted $\mathbf{S}_{ij}$, and we will label (i, j) with $\mathbf{S}_{ij}$. Now, let us put Definition 4.2.3 another way, the property says that whenever we have a variable $X \in \mathbf{S}_{ij}$, X is also in every clique in the path that link i and j in $\mathcal{U}$. Now we have all ingredients to define the JT.

Definition 4.2.4 (Junction Tree)**.** *A junction tree is a clique tree that satisfies the running intersection property 4.2.3.*

Since there are different ways to triangulate the moral graph, the JT representation is not unique. For instance, Fig. 4.2 depicts a JT obtained from DAG $\mathcal{G}$ of student Example 3.2.3. In particular, it shows three separators and four nodes that correspond to cliques from the triangulated graph of $\mathcal{G}$. We can also see that $\{S\}$, which is equal to $\{G, I, S\} \cap \{J, L, S\}$, is included in cliques 1, 3 and 4.

Now let us populate $\mathcal{T}$ with some data to actually be able to perform inference. This kind of data is called factors:

Definition 4.2.5 (Factor)**.** *A factor over a set of random variables $\mathbf{X}$ is a function ϕ such that $\phi : Val(\mathbf{X}) \mapsto \mathbb{R}$, where $Val(\mathbf{X}) = \otimes_{X \in \mathbf{X}} Val(X)$. $\mathbf{X}$ is called the domain of ϕ and denoted $dom(\phi)$*

During the inference process, we need to define the combination of factors, hence we introduce the notion of product between factors, which is defined as follows:

Definition 4.2.6 (Factor product)**.** *Let ϕ_1 and ϕ_2 be two factors over $\{\mathbf{X_1}, \mathbf{X_2}\}$ and $\{\mathbf{X_2}, \mathbf{X_3}\}$ respectively, with $\mathbf{X_1}, \mathbf{X_2}$ and $\mathbf{X_3}$ three disjoint sets of random variables (possibly empty). The factor product of ϕ_1 and ϕ_2, denoted $\phi_1 \times \phi_2$, is the factor ψ such that:*

$$\psi : Val(\mathbf{X_1}, \mathbf{X_2}, \mathbf{X_3}) \to \mathbb{R}$$
$$(\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}) \mapsto \phi_1(\mathbf{x_1}, \mathbf{x_2}) \times \phi_2(\mathbf{x_2}, \mathbf{x_3})$$

We can say that a CPT in Θ is an example of factor. The next step is to assign each $\phi \in \Theta$ to a node $i \in \mathcal{V}(\mathcal{T})$ such that $dom(\phi) \subset \mathbf{C}_i$. Let ϕ_i be the factor associated with i and initially obtained by computing the product of factors assigned to the clique $\mathbf{C}_i$.

Definition 4.2.7 (Adjacency)**.** *Let $i, j \in \mathcal{V}(\mathcal{T})$ such that $i \neq j$. i and j are*

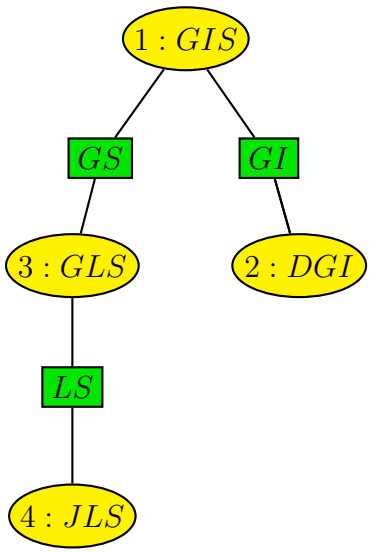


Figure 4.2: A JT for the student example

adjacent in $\mathcal{T}$ *iff* $(i,j) \in \mathcal{E}(\mathcal{T})$. *The set of nodes (cliques) adjacent to* i *is denoted by* $\mathrm{Adj}(i)$, *i.e.,* $\mathrm{Adj}(i) := \Big\{k \in \mathcal{V}(\mathcal{T}) : (i,k) \in \mathcal{E}(\mathcal{T})\Big\}$.

Then the JT algorithm is based on the idea of passing factors between adjacent nodes using a message passing protocol. For $(i,j) \in \mathcal{E}(\mathcal{T})$, let $\psi_{i \to j}$ denote the factor that is associated with Separator $\mathbf{S}_{ij}$ and represent the message sent from i to j over $\mathbf{S}_{ij}$. Note that since messages will be sent in both directions, we distinguish between $\psi_{i \to j}$ and $\psi_{j \to i}$. In practice, $\mathcal{T}$ will also store the factors of cliques and separators. For instance, in Fig 4.3, which corresponds to the JT in Fig. 4.2, we have $\phi_1(G,I,S) = \mathbb{P}(S|I) \times \mathbb{P}(I)$ and $\psi_{1 \to 2} = \psi_{2 \to 1}$ are initialized to the unity factor $\mathbb{1}_{G,I}$.

In order to guarantee the correctness of computations, the message-passing protocol says that the message $\psi_{i \to j}$ should not be sent in an arbitrary manner, but it is computed only when clique i has received messages from all its neighbors except from j. The goal of exchanging a message from node i to j in the junction

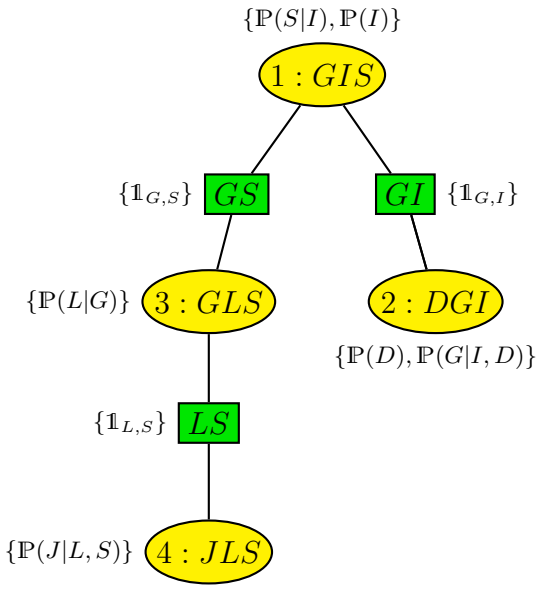


Figure 4.3: An initialized JT data structure for the Student Example 3.2.3

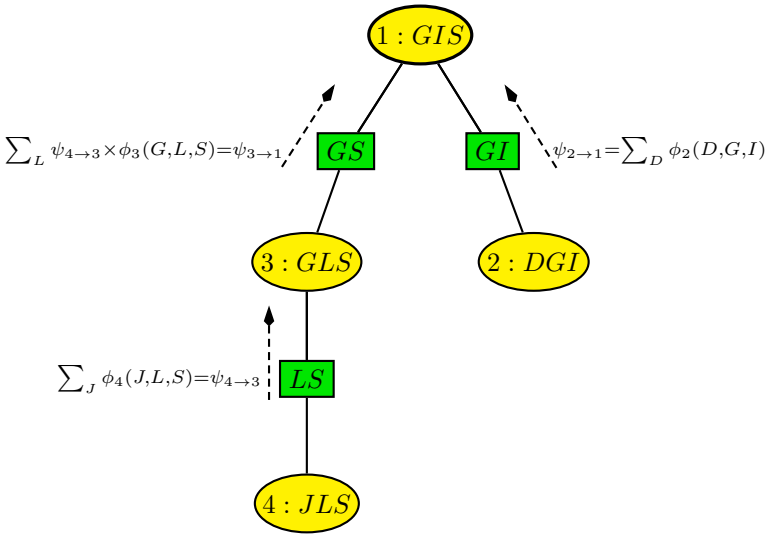


Figure 4.4: Message-passing during *collect*

tree is to propagate some information that is known to i but not to j toward j. After propagating all the messages, all nodes j have received sufficient information about the other nodes in the JT to be able to compute the joint posteriors of $\mathbf{C}_j$.

One way to organize messages computationis to perform it in two phases, namely a **Collect**, using Algorithm 2, and a **Distribution**, using Algorithm 3 from a predetermined root $r \in \mathcal{V}(\mathcal{T})$. During the first phase, messages are sent along edges from leaves toward r and, during the second phase, they are sent in

Algorithm 2: *Collect*

Input : an initialized JT $\mathcal{T}$ and $i, j \in \mathcal{V}(\mathcal{T})$

Output: Updated $\psi_{i\to j}$

1 $\phi \leftarrow \phi_i$ **for** $\Big(k \in \text{Adj}(i) \setminus \{j\}\Big)$ **do**

2 $\quad$ ***Collect***(k, i)

3 $\quad$ $\phi \leftarrow \phi \times \psi_{k\to i}$

4 $\psi_{i\to j} \leftarrow \sum_{X\in dom(\phi)\setminus \mathbf{S}_{ij}} \phi$

Algorithm 3: *Distribute*

Input : an initialized JT$\mathcal{T}$ data structure and $i, j \in \mathcal{V}(\mathcal{T})$

Output: Updated separator factors

$\psi \leftarrow \phi_i \times \prod_{k \in \text{Adj}(i) \setminus \{j\}} \psi_{k \to i}$

$\psi_{i \to j} \leftarrow \sum_{X \in dom(\psi) \setminus \mathbf{S}_{ij}} \psi$

for $l \in \text{Adj}(j) \setminus \{i\}$ **do**

    Distribute(j, l)

Algorithm 4: Junction Tree Algorithm

Input : a JT $\mathcal{T}$ of $\mathcal{B} = (\mathcal{G}, \Theta)$

Output: A JT $\mathcal{T}$ with messages in both directions on all the separators

for $(i, j) \in \mathcal{E}(\mathcal{T})$ **do**

    $\psi_{i \to j} = \psi_{j \to i} = \mathbb{1}$

for $\phi \in \Theta$ **do**

    Assign ϕ to a clique $\mathbf{C}$ such that $dom(\phi) \subseteq \mathbf{C}$

Choose a root $r \in \mathcal{V}(\mathcal{T})$

Collect(r, r)

Distribute(r, r)

the opposite direction. Algorithm 4 corresponds to the JT algorithm and summarizes the discussion we held above. Note that if $\mathbf{C}_i$ contains an evidence node X, then we instantiate X in each $\phi \in \Theta$ such as $X \in dom(\phi)$ and we obtain a reduced factor domain. We also want to emphasize that a JT can be a forest, e.g., when DAG $\mathcal{G}$ is not connected. In this case, the same algorithm is performed in every connected component of the forest. Later, for simplicity, we assume that we deal with a tree. Now let us illustrate the JT algorithm on our running ex-

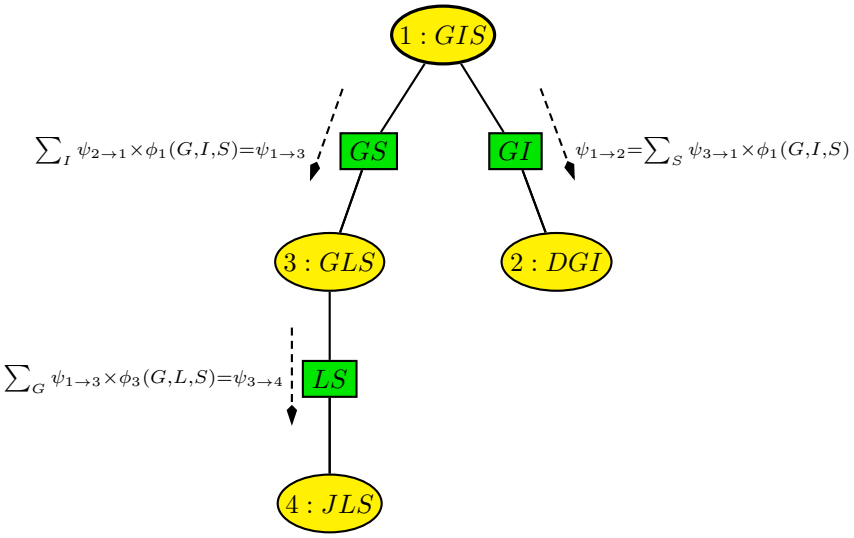


Figure 4.5: Message-passing during *distribution*

ample. We assume $\mathcal{T}$ is rooted at clique 1. Then Fig. 4.4 illustrates the collect phase, where we begin to send messages from leaves. In Fig. 4.5 we proceed with the distribution of messages from the root 1 toward the leaves. After performing these two phases, in order to have the joint posterior distribution over the variables in $\mathbf{C}_i$, up to some normalization constant, all we need is to compute the product $\phi_i \times \prod_{(k,i)\in\mathcal{E}(\mathcal{T})} \psi_{k\to i}$. For instance, after *Distribution* phase in Fig. 4.5, $P(D,G,I) = \phi_2(D,G,I) \times \psi_{1\to2}$.

In the next section we develop an extension of the JT algorithm.

4.3 Incremental Junction Tree Inference

This section investigates a new approach to overcome the shortcomings discussed in the introduction section (4.1). It aims at improving the efficiency of inference for very large and dynamic systems. The key idea of our algorithm, called Incremental Junction Tree Inference (*IJTI*), consists of restricting the computations only to parts of the JT that are relevant to *targets* and that have been invalidated by

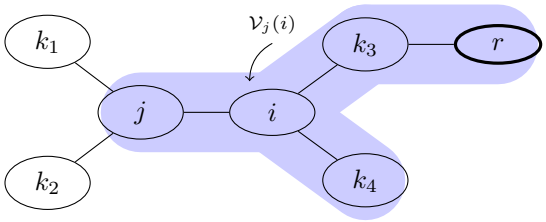


Figure 4.6: A maximal sub-tree in a JT

incremental changes. IJTI graphically characterizes those parts and thereby minimizes probabilistic computations. Let us introduce some notations and definitions

Definition 4.3.1. *Let $r \in \mathcal{V}(\mathcal{T})$, $r \neq i$, then $Adj_r(i)$ denotes the singleton set containing the clique adjacent to i that is on the path between i and r, i.e., $Adj_r(i) := \left\{k \in \mathrm{Adj}(i) : k \in i-r\right\}$, where $i-r$ denotes the undirected path in $\mathcal{T}$ linking i and r. We also define $Adj_r(r) := \emptyset$. Finally, let $\mathrm{Adj}_{\text{-}j}(i) := \mathrm{Adj}(i) \setminus \{j\}$.*

For instance, in Fig. 4.6, $Adj_r(i) = \{k_3\}$ and $Adj_r(k_3) = \{r\}$. Finally, let $\mathcal{V}_{\text{-}j}(i)$ stands for the set of nodes of the maximal subtree in $\mathcal{T}$ that contains i and not $Adj_j(i)$, and let $\mathcal{V}_j(i) = \mathcal{V}_{\text{-}j}(i) \cup \{j\}$ (see the shadowed area in Fig.4.6).

A message $\psi_{i\to j}$ sent within $\mathcal{T}$ is directed by nature. It propagates toward j (and, by induction, toward $\mathcal{V}_{\text{-}i}(j)$) all the relevant information coming from the cliques in $\mathcal{V}_{\text{-}j}(i)$, notably all the evidence they received (by abuse, we say that a clique received evidence when at least one of its random variables received evidence). As a consequence, if $\psi_{i\to j}$ has already been computed previously and no new evidence has been received nor structural changes occurred in $\mathcal{V}_{\text{-}j}(i)$, there is no need to recompute it. But even if $\mathcal{V}_{\text{-}j}(i)$ received evidence, $\psi_{i\to j}$ needs not be computed/updated if $\mathcal{V}_{\text{-}i}(j)$ contains no target. In this case, $\psi_{i\to j}$'s state becomes "invalid" since the content of $\psi_{i\to j}$ is now incorrect. This is not an issue for the current inference but, for future ones, we have to take this state into account to recompute $\psi_{i\to j}$ if it is to be used. Let $\mathcal{A}(\mathcal{T})$ be the set of all arcs induced from

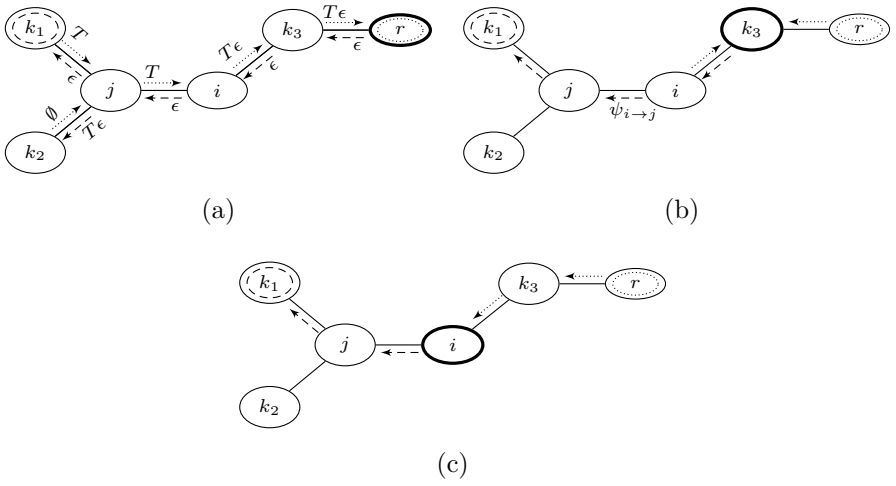


Figure 4.7: Message passing within a JT $\mathcal{T}$.

$\mathcal{E}(\mathcal{T})$, taking into account orientations, i.e., $\mathcal{A}(\mathcal{T}) := \bigcup_{(i,j)\in\mathcal{E}(\mathcal{T})}\big\{(i,j)\big\} \cup \big\{(j,i)\big\}$. To formalize the above conditions, we begin with characterizing the information that is "local" to i and j. By local we mean information that only depends on the content or state of i and j:

Definition 4.3.2 (Local label-message λ)**.** $\lambda : \mathcal{A}(\mathcal{T}) \mapsto 2^{\{\epsilon,T\}}$ *is a function s.t.*

$$(i,j) \longmapsto \lambda_{i\to j} := \begin{cases} \{\epsilon\} & \textit{if } \psi_{i\to j} \textit{ is in "invalid state" or "new evidence or structural changes" have occurred } (1) \\ \{T\} & \textit{if } i \textit{ contains targets (2)} \\ \{T,\epsilon\} & \textit{if (1) and (2)} \\ \emptyset & \textit{otherwise} \end{cases}$$

To simplify the notation, hereafter, we will remove braces and denote $\{T,\epsilon\}$ by $T\epsilon$. Then, the idea of our algorithm consists of marking every arc (i,j) in $\mathcal{A}(\mathcal{T})$ by labels $\mu_{i\to j}$ expressing all the "local" information that $\mathcal{V}_{\text{-}j}(i)$ contains.

Definition 4.3.3 (Label-message μ)**.** *For $(i,j) \in \mathcal{A}(\mathcal{T})$, the label-message sent from i to j is a function $\mu : \mathcal{A}(\mathcal{T}) \mapsto 2^{\{\epsilon,T\}}$ such that $\mu_{i\to j} := \bigcup_{\substack{k'\in\mathcal{V}_{\text{-}j}(i) \\ \{k\}=Adj_j(k')}} \lambda_{k'\to k}$.*

As an example of the previous discussion, imagine that a first incremental update impacts the initial DAG and consequently the initial $\mathcal{T}$ of Fig. 4.6. This consists of the removal of k_4, the insertion of an evidence on r and a new target on k_1. Fig. 4.7a depicts the μ-messaging within $\mathcal{T}$ after this update, where dashed and dotted ellipses stand for the cliques containing targets and evidence respectively. One can easily see that, for instance, $\mu_{i\to j} = \epsilon$, $\mu_{j\to i} = T$ and $\mu_{j\to k_2} = T\epsilon$. The following proposition allows to recursively construct the μ-messages:

Proposition 4.3.4 (μ construction)**.** *Let* $(i,j) \in \mathcal{A}(\mathcal{T})$*, then we have :* $\mu_{i\to j} = \lambda_{i\to j} \cup \bigcup_{k\in \mathrm{Adj}_{\mathcal{T}-j}(i)} \mu_{k\to i}$.

All the proofs are provided in the appendix at the end of this manuscript.

4.3.1 Optimal Roots

Let us recall that $\psi_{i\to j}$ denotes the message exchanged between cliques i and j during an inference computation. It shall not be confused with the label message $\mu_{i\to j}$ of Definition 4.3.3. As we can notice from the JT algorithm description, the number of computations performed by a JT-based message-passing algorithm does not depend on the root clique selected for collect/distribution because the $\psi_{i\to j}$ messages are sent on both directions on *all* the edges of the JT. For IJTI, this is not the case, since this algorithm computes and sends *only* the $\psi_{i\to j}$ messages necessary for the computation of the posterior distributions of its target nodes. On some edges, IJTI will therefore not compute some $\psi_{i\to j}$ messages because they are irrelevant w.r.t. the targets posterior distributions. As a consequence the number of computations performed in IJTI is sensitive to the root selection. For instance, in the JT of Fig. 4.6, if clique i received evidence and the only target is j, only message $\psi_{i\to j}$ from i to j is necessary, which is precisely what is sent if clique i is selected as root (here, only a distribution is necessary). But

if clique k_4 is selected instead, message $\psi_{i\to k_4}$ needs to be sent during the collect and messages $\psi_{k_4\to i}$ and $\psi_{i\to j}$ need to be sent during the distribution, which is clearly not optimal. To determine the optimal roots, let us define $\delta_{i\to j}(r)$ as an indicator of whether message $\psi_{i\to j}$ is recomputed (in this case, $\delta_{i\to j}(r)=1$) or not ($\delta_{i\to j}(r)=0$) when r is selected as a root. In IJTI, we therefore seek to minimize $\delta(r)=\sum_{(k',k)\in\mathcal{A}(\mathcal{T})}\delta_{k'\to k}(r)$, which corresponds to the total number of messages recomputed and sent. Based on the discussion of the preceding section, we can write:

$$\delta_{i\to j}(r)=\begin{cases}1 \text{ if } (\epsilon\in\mu_{i\to j} \text{ and} \{j\}=Adj_r(i)) \text{ or}\\ \qquad (\epsilon\in\mu_{i\to j} \text{ and} \{i\}=Adj_r(j) \text{ and } T\in\mu_{j\to i})\\ 0 \text{ otherwise}\end{cases} \tag{4.1}$$

The first line of Eq. (4.1) concerns *collect* messages ($\{j\}=Adj_r(i)$). It asserts that *collect* message $\psi_{i\to j}$ needs to be recomputed only if it is currently in an invalid state or if new evidence or structural changes have occurred in $\mathcal{V}_{\text{-}j}(i)$ ($\epsilon\in\mu_{i\to j}$). When this is not the case, clearly, this message is up to date and does not need recomputation. The second line of Eq. (4.1) concerns *distribution* messages ($\{i\}=Adj_r(j)$). It asserts that $\psi_{i\to j}$ needs to be recomputed only if there exists a target farther toward the leaves of the JT ($T\in\mu_{j\to i}$) and if some evidence has been received on $\mathcal{V}_{\text{-}j}(i)$ or some message coming from $\mathcal{V}_{\text{-}j}(i)$ has been updated ($\epsilon\in\mu_{i\to j}$). Eq. (4.1) can be rewritten more compactly as:

$$\delta_{i\to j}(r)=\begin{cases}1 \text{ if } \epsilon\in\mu_{i\to j} \text{ and}\\ \qquad (\{j\}=Adj_r(i) \text{ or } (\{i\}=Adj_r(j) \text{ and } T\in\mu_{j\to i}))\\ 0 \text{ otherwise}\end{cases} \tag{4.2}$$

Fig. 4.7b and Fig. 4.7c illustrate that $\delta(k_3)=5$ and $\delta(i)=4$ respectively. In this case, it is better to select i as a root rather than k_3 since this avoids the unnecessary

computation of one message. The following theorem states the existence of some optimal roots and characterize them:

Theorem 4.3.5 (Optimal roots)**.** *Suppose we computed the μ-messages within $\mathcal{T}$. Then there exists $r \in \mathcal{V}(\mathcal{T})$ fulfilling one of the following mutually exclusive and exhaustive properties:*

1. $(\mathcal{V}(\mathcal{T}), \mathcal{E}(\mathcal{T})) = (\{r\}, \emptyset)$
2. $\exists r' \in \mathcal{V}(\mathcal{T}) : \mu_{r' \to r} = \mu_{r \to r'} = T\epsilon$
3. $\forall k \in Adj(r) : \mu_{k \to r} \in \{T, \epsilon, \emptyset\}$

In addition, $r \in Argmin_{k \in \mathcal{V}(\mathcal{T})} \delta(k)$, i.e., r is an optimal root w.r.t. inference computations.

4.3.2 A new Incremental Inference

In this section, we propose a new algorithm designed to deal with incremental inference. We assume that a first inference has been performed by message-passing within $\mathcal{T}$, using for instance a collect-distribute algorithm in a JT-based inference architecture. Afterwards, incremental changes occur. Then *IJTI* is called to optimize the inference process. We recall that we use a target-driven approach, hence, we recompute only invalidated *collect* messages and we only *distribute* messages up to the *targets*. Under these assumptions, the proposed algorithm is described in Algorithm 5. It runs a revised message-passing algorithm to compute $\psi_{i \to j}$ only when $\delta_{i \to j}(r) = 1$ for all i, j in the modified junction tree $\mathcal{T}$. In line 5, a leaf clique i is such that $|\text{Adj}(i)| = 1$. We emphasize that computing messages is performed similarly to a classic JT-based inference algorithm. The correctness of IJTI is guaranteed by the following proposition:

Algorithm 5: Incremental Junction Tree Inference (IJTI)

input : modified $\mathcal{T}$, Q targets cliques

output : posteriors on targets

`// set the number of neighbors visited during the collect`

for $i \in \mathcal{V}(\mathcal{T})$ **do**
    $i.nbVN \leftarrow 0$

Compute the μ-labels in $\mathcal{T}$

Find r using Theorem 4.3.5

$\mathbf{L} \leftarrow$ the set of leaves of $\mathcal{T}$

`// collect phase`

foreach *clique* $i \in \mathbf{L}$ **do**
    $p \leftarrow Adj_r(i)$
    if $\delta_{i \to p}(r) = 1$ **then** Compute $\psi_{i \to p}$
    $p.nbVN \leftarrow p.nbVN + 1$
    if $p \neq r$ **and** $p.nbVN = |Adj(p)| - 1$ **then** $\mathbf{L} \leftarrow \mathbf{L} \cup \{p\}$
    $\mathbf{L} \leftarrow \mathbf{L} \setminus \{i\}$

`// distribution phase`

$\mathbf{L} \leftarrow \{r\}$

foreach *clique* $i \in \mathbf{L}$ **do**
    foreach $j \in Adj(i) \setminus Adj_r(i)$ **do**
        if $\delta_{i \to j}(r) = 1$ **then**
            Compute $\psi_{i \to j}$
            $\mathbf{L} \leftarrow \mathbf{L} \cup \{j\}$

foreach *clique* $t \in Q$ **do**
    Compute the posterior distributions of the target nodes in clique t

return posterior distributions

Proposition 4.3.6. *The IJTI algorithm is sound, i.e., computing only messages* $\psi_{i\to j}$ *such that* $\delta_{i\to j}(r) = 1$*, for all* $(i,j) \in \mathcal{A}(\mathcal{T})$*, results in the correct computation of the posterior distributions of the target variables.*

4.4 Evaluation

We performed two kind of experiments to evaluate IJTI. First, the gain is evaluated in terms of the number of saved messages and second, it is expressed in terms of computations time.

4.4.1 Messages Optimization

In this section, we highlight the effectiveness of our algorithm by comparing the gain of using it instead of any non-incremental JT-based inference algorithm. The gain can be expressed as the percentage of unnecessary messages that IJTI avoids to compute compared to the messages sent by classical inference algorithms on both directions on all the edges and this amounts to $1 - \delta(r)/(2|\mathcal{E}(\mathcal{T})|)$. For this purpose, we performed tests using the aGrUM library on 9 real-world BNs of different complexities as well as on randomly generated BNs. The latter contained $nbNodes$ Boolean random variables, ($6 \leq nbNodes \leq 900$, see Fig. 4.9) and, for each value of $nbNodes$, 3 BNs were generated with $nbArcs$ arcs, $nbArcs$ being chosen randomly in the interval $[nbNodes - 1, 4/3 * nbNodes - 1]$. We simulated the incrementality by randomly choosing for each inference a set of targets and modified cliques. This induced invalid messages in $\mathcal{T}$. Fig. 4.8 and Fig. 4.9 show the average gains and their standard deviations (error bars over 20 incremental inference queries) [1]. Note that the behavior of the algorithm is the same for real-

[1] In Fig. 4.8.a, due to the small number of nodes and arcs in the BNs, percentages of modifications lower than 10% imply no modification at all, hence the lack of error bars.

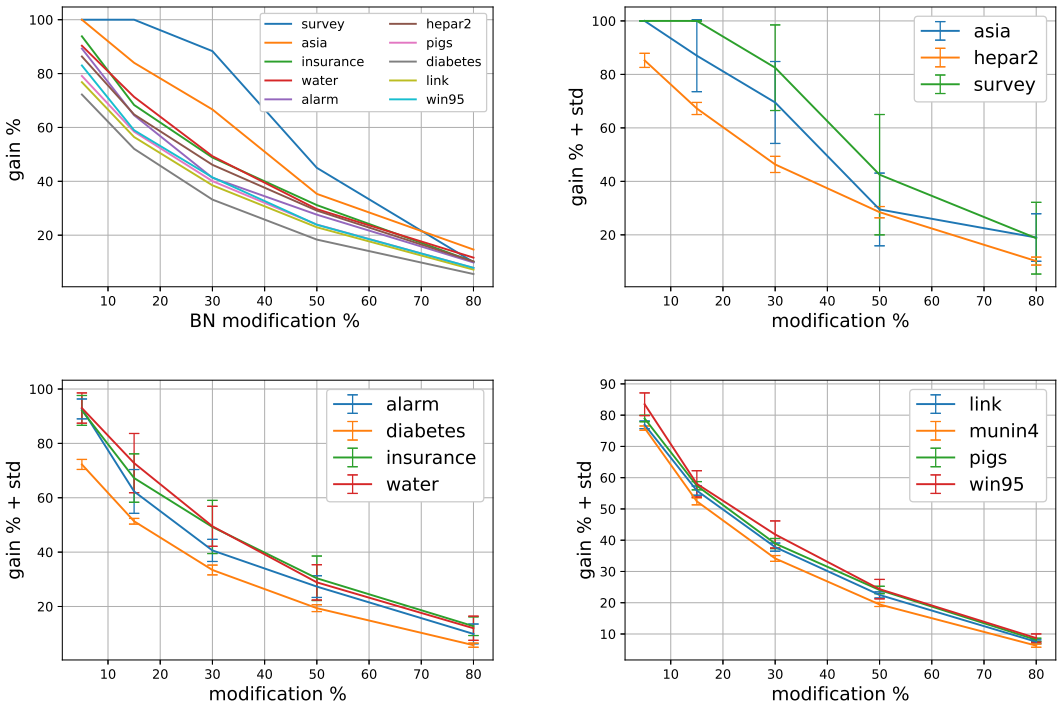


Figure 4.8: IJTI messages optimization for real BNs

world BNs and for randomly generated ones. As could be expected, the smaller the modifications, the bigger the gain. Note also that the gain is not too sensitive to the size of the BN. Fig. 4.9 also shows the same behavior of the gain for larger randomly generated BNs.

4.4.2 Time Optimization

In this experiment, we want to test IJTI in terms of time consumption while reflecting incremental queries of the WM by the BRMS engine. As we are interested in local modifications of the JT, we evaluate the performance of IJTI when the modification impacts less than 40% of the BN size.

The BRMS engine execution may induce incremental changes in the WM and trigger probabilistic inference many times. Hence, a simulation of such an ex-

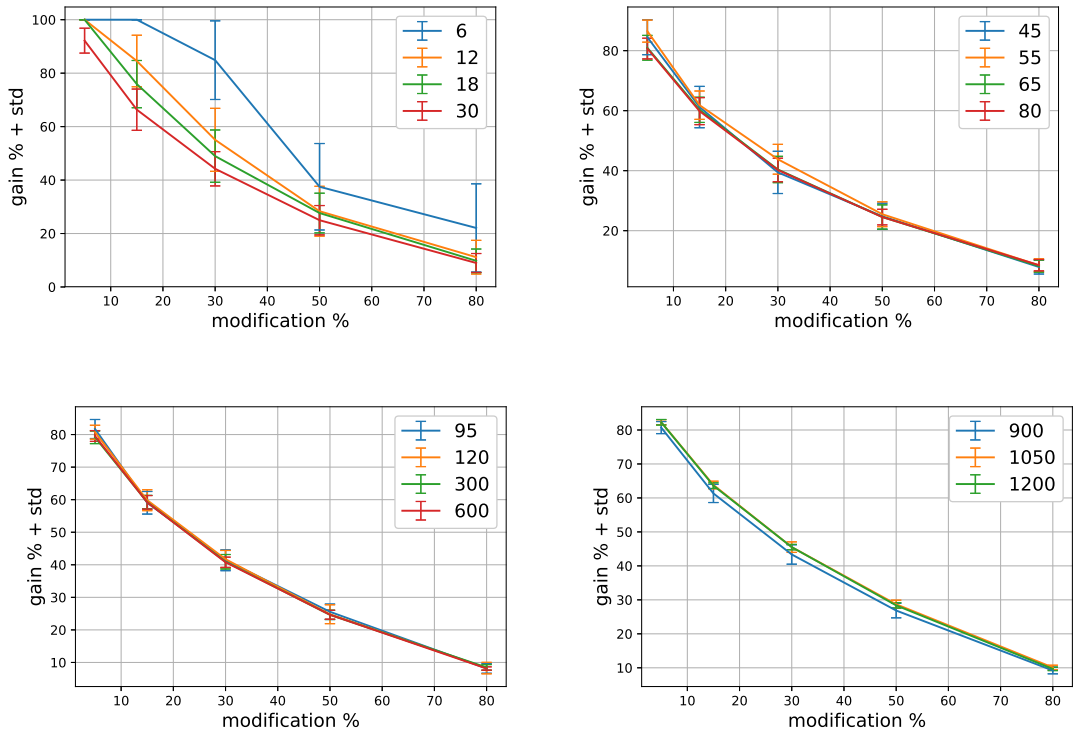


Figure 4.9: IJTI messages optimization for artificial BNs, each curve corresponds to a BN size

ecution will be reflected on the probabilistic side by changing evidence and/or structure of the JT (new arcs, nodes and evidence added or removed) and performing a number of probabilistic inference. Moreover, in order to see the impact of evidence/targets size, each simulation fixes the set of targets and evidence. We therefore control the percentage of modifications that produce the structural changes, let it be `hard_change`, and those which only change evidence without impacting the structure, let it be `soft_change`. Actually our junction tree-based architecture processes differently hard and soft evidence as follows. On one hand, when adding/removing hard evidence to/from BN, one must systematically reconstruct a new JT. On the other hand, a soft evidence entails such a reconstruction only when the evidence node does not belong to the current JT. Finally, note that

the probabilistic inference is only triggered after some change occurs in the JT, otherwise the previous computations are still valid. Algorithm 6 summarizes the approach we use to simulate the incrementality and record the computations time to quantify the gain obtained using IJTI.

Algorithm 6: Simulation of incremental inference using IJTI

Data: BN, hard_change and soft_change percentages, target and evidence percentages

Result: Records for computation times

```
1  Set initial target and evidence set
2  Compute posterior of targets
3  foreach hard_change and soft_change percentage do
4      repeat
5          Randomly choose targets and evidence w.r.t initial percentages
6          Compute posterior of targets
7          repeat
8              Introduce hard_change, soft_change into the JT of BN
9              Compute posteriors of targets
10             Record inference time
11         until number of inferences per simulation is reached;
12     until an average criterion is reached;
```

As we randomly select evidence and targets set, the simulation repeats the selection at line 4 in order to report average results. In practice we iterate this 20 times. As we discussed earlier, a simulation corresponds to a sequence of calling the probabilistic inference after introducing some changes. This is expressed by the repeat loop at line 7, which also corresponds to 20 in practice .

As the percentage of evidence/targets increases, the inference times increase and

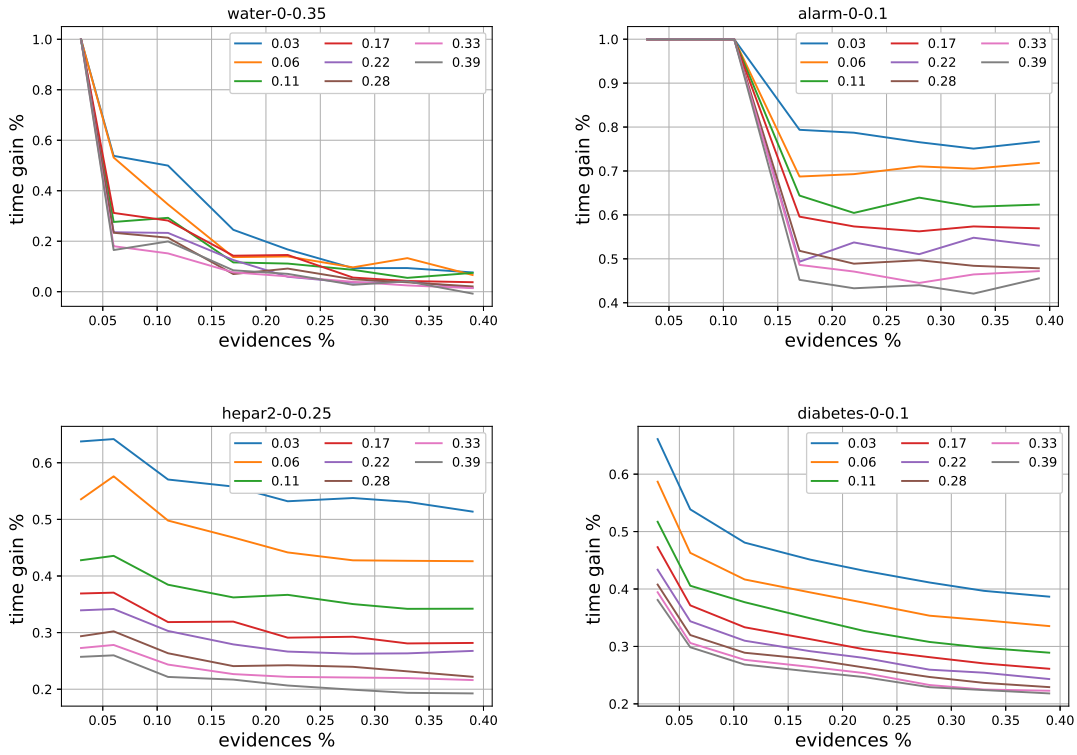


Figure 4.10: Average time gains for some real BNs. The title expresses hard and soft change percentages respectively. Each plot corresponds to a target %

the computations that are still valid from previous inferences are reduced. So the average gain decreases. The latter is computed as follows. For each pair of targets and evidence percentages, we averaged inference time over $20 \times 20 = 400$ probabilistic queries following the schema in Algorithm 6. Let t_{IJTI} (resp. t_{LP}) be the resulting average for IJTI (resp. Lazy Propagation, a state-of-the-art algorithm similar to that described in Algorithm 4). The average gain can be expressed as $1 - t_{IJTI}/t_{LP}$.

In this experiment, we are only interested in real BNs. The x-axis of Fig. 4.10 shows evidence percentages while the averaged time gain appears on the y-axis. For each BN, a figure corresponds to a fixed parameter $soft\_change$ and $hard\_change$, which is also showed in the title. We recall that these two parameters are used to

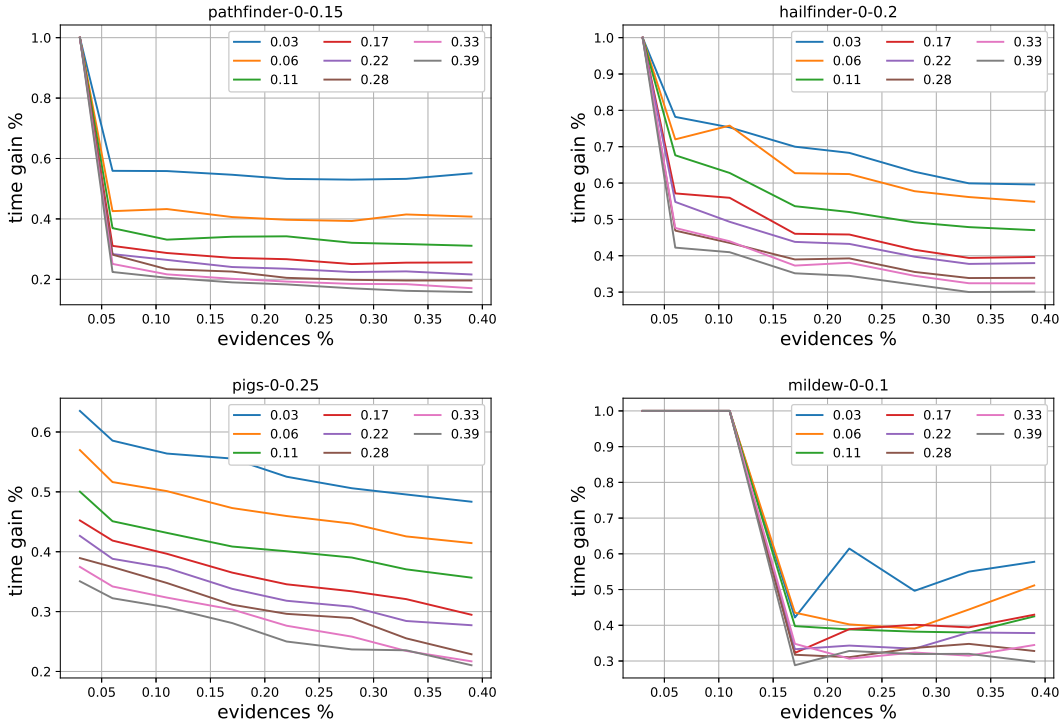
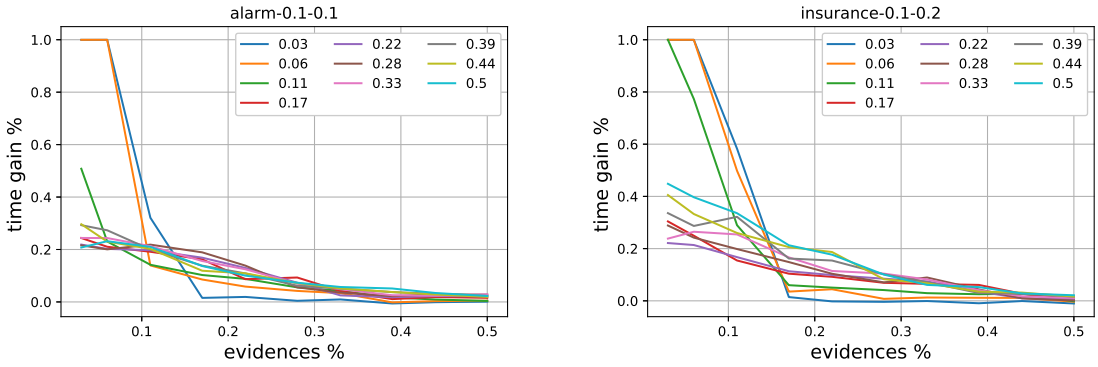
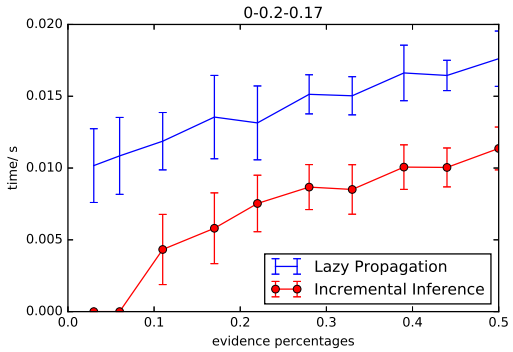


Figure 4.11: Average inference time gains for other real BNs. Each plot corresponds to a target %

update soft and hard evidence between inference iterations. It may be seen clearly that, after being significant in small percentages, the gain falls as the percentage of change increases. We explain this by the fact that the reuse of previous computations, hence our optimization, is more significant in small or local changes. We have similar behavior for other real BN as Fig. 4.11 depicts. We also notice a general behavior of both algorithms showing that the bigger the percentage of changes, the bigger the inference times. Fig. 4.12b shows an example of this behavior in the Alarm network. Finally, we want to emphasize that, if in addition we introduce structural changes to the JT, i.e., $hard\_change \neq 0$, the gain decreases steeply, even for small structural changes, and stabilizes quickly to almost no gain. Fig. 4.12a illustrates this in the case of Alarm and Insurance networks. This is due



(a) Average gain in the presence of JT structural changes



(b) Inference time for Alarm network. Title corresponds to $(hard\_change, soft\_change, targets\%)$

to the fact that the current implementation of IJTI spends more time to discover the valid zones in the JT, and this time impacts negatively the overall inference time. We refer to Section B of the Appendix for more results about the inference times and the gain obtained for different networks and parameters.

4.5 Conclusion

In this chapter, we introduce *IJTI*, a new incremental junction-tree-based inference algorithm for multi-target dynamic systems. Assuming that a first complete infer-

ence has been performed, it first identifies an optimal root and second optimizes the inference accordingly. The correctness of these two optimizations is proved and experiments highlight that the algorithm allows for important savings compared to classical ones in the presence of local changes. In particular, we show that IJTI outperforms state of the art inference algorithms for small or local changes and the gain is more important when there is no structural changes impacting the JT. In the next chapter, we deal with the issue of uncertainties in modern BRMSs. We present a coupling approach with PRMs and we see that IJTI is well-suited to answer probabilistic queries in such a context.

Chapter 5

Business Rules Uncertainty Management

5.1 Introduction

We mentioned in Chapter 3 that uncertainty is inevitably present in real world applications. Thus, in order for BRMSs to support more realistic decisions, handling domain uncertainties becomes a must. In this chapter, we see how a coupling approach can be a solution for modern BRMSs to effectively manage uncertainties. The solution we propose adds probabilistic reasoning feature by coupling the BRMS with a probabilistic engine. A complete integration is also conceivable as in [122], where the RBS engine is extended to manage rules and their uncertainties in the same time. However, we argue that separating concepts and architectures simplifies the software development and maintenance, and provides more control over the environment complexity (language refactoring, testing, etc.). In fact, probabilistic inference is a complex process that would be inefficient for a rule engine to manage besides its own inference. Therefore, we are interested in a loosely coupled architecture to facilitate inter-operability and insure a clear sepa-

ration between business and probabilistic logics. Doing so, we want to avoid the inter-independence between modules in tightly coupled systems, where a change in the uncertainty management will require changes in rules management. The loose coupling services principle is largely advocated in SOA alongside the decoupled contract design pattern [41]. The key idea of such a principle is to allow a service contract to be expressed independently of its implementation. In our context, this would reduce dependencies between business logic and uncertainty management implementation. We want to have a transparent uncertainty management while being independent from the probabilistic implementation of that service. For example, one might change the probabilistic engine in the future without altering the service contract. This chapter, provides an approach that couples BRMSs and PRMs. It describes how such an approach is articulated and how it ensures a practical probabilistic reasoning support. Finally, the chapter applies the approach to ODM.

5.2 Coupling BRs with PRMs

5.2.1 Uncertain OO-BRs Principles

From Chapter 2 we recall that BRs are rules in the form "`If <condition> Then <action>`" that are exploited for reasoning by forward chaining inference engines. OO-BRMSs execute BRs against an OM that describes the application objects based on a data model. In this latter we can differentiate two components, which include an executable data model that can be implemented in JAVA or XML schema, and a business-oriented model, which uses specific vocabulary and terms familiar to business users and is expressed in a domain-specific language. Hence, WMEs (or facts) are objects in the sense of the OO paradigm. It follows that an elementary action or condition operates on tuples rather than simple values.

A single action affects WMEs by inserting a new object, removing an existing object or updating an existing object. Each of these actions potentially leads to a re-evaluation of any rule that matches the object in question.
It turns out that there exists a natural mapping between model elements in BRMSs and PRMs. We illustrate this through a simplified example from an insurance application. This example is not intended to be exhaustive but rather to illustrate necessary concepts.

Example 5.2.1 (Simple insurance application)**.** *An insurance organism allows its subscribers to request reimbursements depending on invoice types. A request must be validated by a health care professional and each subscriber can have many reimbursement requests.*

A simplified representation of the model would consist of three classes representing a healthcare professional, a subscriber and a reimbursement request. Fig. 5.1 gives a UML class diagram for such an application. In a fraud detection context, we want to verify, using a BR-based approach, whether the healthcare professional is fraudulent. In such a context, anomalies that indicate fraud are detected by executing a set of rules and using scoring heuristics. For instance, if a fraud detection rule says that an excessive invoice alert must be raised on a healthcare provider who submits a high price reimbursement request for one of his subscribers, the corresponding object-oriented BR in Rule 5.1 will look for objects in the WM that correspond to providers with subscribers requesting reimburse-

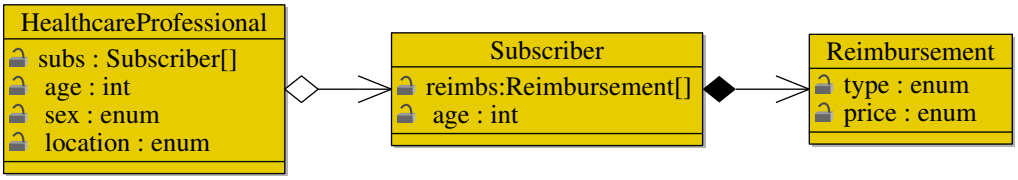


Figure 5.1: UML diagram for Example 5.2.1

ments with a high price.

Rule 5.1: detect invoice anomaly

```
IF hp has type HealthcareProfessional
  & sub has type Subscriber
  & reimb has type Reimbursement
  & sub in hp.subs & reimb in sub.reimbs
  & reimb.price == high
THEN raiseAlertExceededInvoicePrice(hp)
```

Similarly, Rule 5.2 says that a lens age anomaly alert must be raised on a healthcare professional who submits a lens reimbursement request for a subscriber under age 10.

Rule 5.2: detect lens anomaly

```
IF hp has type HealthcareProfessional
  & sub has type Subscriber & reimb has type Reimbursement
  & sub in hp.subs
  & reimb in sub.reimbs
  & sub.age < 10 & reimb.type=lens
THEN raiseAlertLensAgeAnomaly(hp)
```

When the data is completely known and well adapted to the classical logic paradigm, such rules are well handled using variants of pattern matching algorithms, e.g, enhanced Rete. However, when coping with uncertain or missing data, such rules cannot be executed. That is exactly where PRMs can play an important role to handle such a situation.

In Example 5.2.1, we specified three classes. Fortunately, PRMs allow also to handle OO concepts, such as class and references. This remark is fundamental in our case since it will simplify the communication between both rules and probabilistic engines. The corresponding PRM representation for the running Example

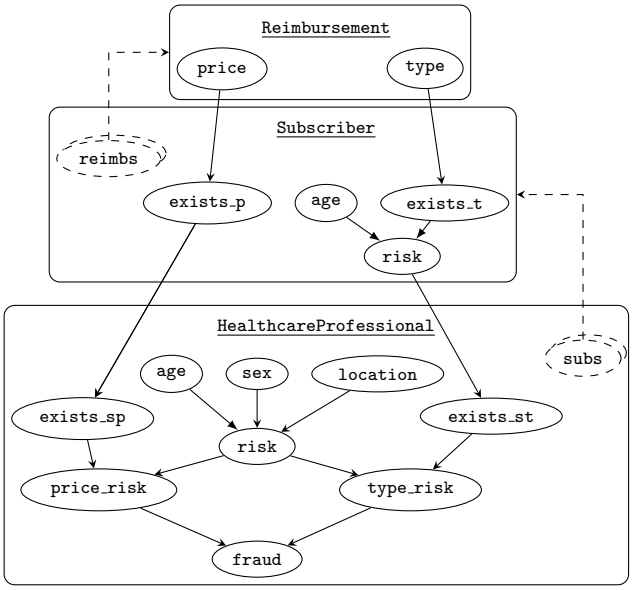


Figure 5.2: Class dependency schema for the insurance example

5.2.1 is given in Fig. 5.2. In particular, we found exactly three PRM classes related to each other, in a way that is similar to class interactions in the UML diagram 5.1. The attribute `reimbs` of class `Subscriber` can be seen as a multiple reference slot, which shows that the class points to a set of `Reimbursement`. In this example, we need to aggregate the informations about all reimbursement requests that belong to a subscriber. Hence, some PRM aggregators should be introduced. A divide-and-conquer approach is used to build PRM aggreagators in Fig. 5.2. We first determine whether the `Subscriber` has a `Reimbursement` with a `high price` (by the `exists_p` aggregator); second, we determine if the `HealthcareProvider` is linked to a `Subscriber` satisfying the previous condition (by the `exists_sp` aggregator). We follow the same reasoning to generate the aggregator `exists_st`. Fig. 5.3 depicts an example of a relational skeleton obtained from the fraud example instances. A dashed arc stands for a reference slot. For instance `sub1` references `reimb1` and `reimb2`. Now that we have an equivalent PRM for rules classes definitions, we are able to execute rules that manipulate objects with uncer-

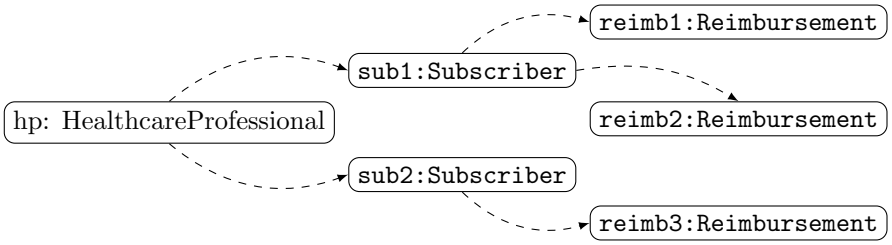


Figure 5.3: A relational skeleton for the fraud example

tainty in their attributes. As an obvious consequence, attributes with uncertainty in the rules are directly mapped to their equivalent PRM attributes. In addition, instead of evaluating Boolean expressions in the condition part, we introduce a `probability` operator to trigger the probabilistic engine inference. Thus, an enhanced version for Rule 5.1, which says basically that an alert must be raised when a healthcare professional submits a expensive price reimbursement for a subscriber, we can have Rule 5.3 that says that an excessive invoice alert must be raised on a healthcare professional if there is a 80% probability that the price of a reimbursement request is excessive.

Rule 5.3: detect invoice anomaly with probability

```
1 IF hp has type HealthcareProvider
2   & probability(hp.price_risk=high)>.8
3 THEN raiseAlertExceededInvoicePrice(hp)
```

To summarize the approach, handling uncertainty in OO-BRMSs using PRMs is based on the idea of model mapping and the introduction of a probability operator in the rules syntax, which is responsible for triggering probabilistic inference. The mapping itself relies on the fact that both models share many interesting OO concepts such as classes and relations. The probabilistic knowledge encoded by the PRM pertains to our knowledge about the business domain. This is emphasized through relating the PRM to the OM that expresses business concepts,[1] rather

[1] see discussion about OM vs knowledge base in Section 2.3

than rules that are likely to evolve. The next subsection specifies in more details the extensions we need to implement these principles.

5.2.2 Model Extension

As we discussed previously, PRMs relate attributes of different classes, and those of generated instances consequently, and have the advantage to create complex networks covering multiple instances. Although, random variables are generated from the same classes, they should be regarded as distinct variables with their own life-cycle. For instance, `sub1.risk` and `sub2.risk` in Fig. 5.3 are two distinct random variables generated from class `Subscriber`. We know from Fig. 5.2 that `price_risk` attribute is linked by reference mechanism to attributes of classes `Subscriber` and `Reimbursement` in the PRM, hence, in our extension, there is no need to evaluate conditions that can be processed by probabilistic inference. For instance, the operator `in`, used in Rule 5.1 to constrain the domain of an object, is implicitly specified through the link of reference slot `subs`. When the engine encounters the `probability` operator, it immediately launches a probabilistic query, i.e., it queries the underlying PRM. In such an extension, probabilistic data is explicitly identified and can be processed by PRM engines.

We extend both model and meta-model of the rules. First by enriching the model with new attributes as in the previous section and second by adding probabilistic annotations. This has two advantages. The first is moving probabilistic definitions from rules to their data meta-model. In making this move, probabilistic model and inference handling[2] are externalized to the PRM engine, which implies a separate management of business and probabilistic logic. Second, this enables the model to be more independent w.r.t. the rules, which means an independent evolution of both models.

[2]Bear in mind that the probabilistic model definition is held by the rules OM.

Annotations are a type of meta-data that enriches the meta-model at hand. In this work, they are added to indicate that a class contains probabilistic information, as well as that an attribute is mapped to a PRM attribute and is parameterized by its corresponding CPT and parents. If the attribute is an aggregator, annotations show its type, its domain and the concerned modalities (the possible states or values of the random variables). Therefore, an OM class (resp. an attribute) is mapped to a PRM class (resp. an attribute) and the probabilistic data and how classes are related to each other are extracted from the OM annotations. In the OM, a restricted type represents a type whose domain is restricted, for instance an `integer` that is restricted to $\{0, 1, 2\}$. Only discrete random variables are supported in PRMs, they can be user-labeled (e.g, `state_type`) or built-in types (e.g, `Boolean,int`). Thanks to these annotations, the rules engine can generate the underlying PRM classes and system at compile time. Before the generation process, the model is parsed and checked. For example we check if the given list of a PRM attribute parents is valid and consistent with its CPT. This latter is also checked to verify it represents a well defined probability distribution. Actually, there are two possible modes for PRM system definition. The first is a static declaration, which assumes that all WM instances are known at compile-time. The PRM system is then generated either by directly processing the WM instance graph, or by an explicit declaration inside a special annotated class, which also specifies necessary relations. The second mode allows a dynamic definition in addition to the previous mode. Here, rules' executions may also update the system by incrementally inserting new instances or modifying relations for instance. The last mode is obviously much more interesting since it reflects BRs and the WM dynamic nature. The mapping we use allows the rules to generate complex probabilistic networks via the simple mechanism of class instantiation and reference slot. This property enables the rules, for instance, to handle many sets of random variables, which are

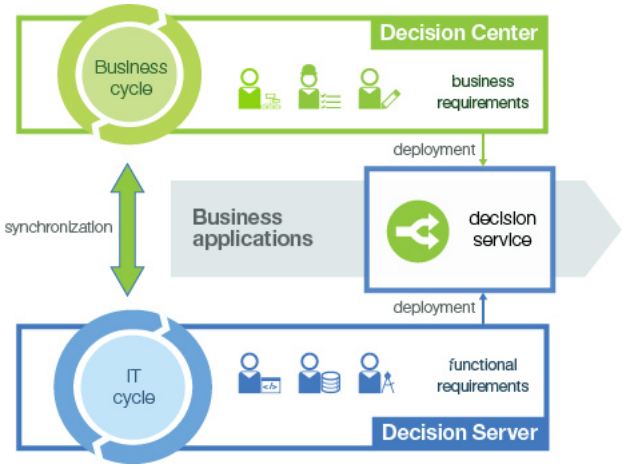


Figure 5.4: ODM life-cycles

obtained for free, just by means of linking instances.

5.3 Application to IBM ODM

The aim of this section is to provide a practical view of the foundations introduced in the previous section. Our implementation is based on IBM ODM [63] as an OO-BRMS and a Graphical Universal Modeler (aGrUM) [49] as a probabilistic engine. It is important to emphasize that the methodology we applied can be easily generalized to any OO-BRMS as we showed previously. This section gives an overview of the ODM functionalities and is mainly built from material in [19, 63], where a more detailed presentation is given.

5.3.1 Overview

The ODM platform is a set of complex modules and applications that operate together in different environments while ensuring a clear separation between IT and business life-cycles as it is shown in Fig. 5.4. ODM enables organizational policies to be defined, deployed, monitored and executed in an application server. In ODM,

business users and developers use tools that reflect their different skills and role within the business application. Fig. 5.5 depicts the main components that ODM provides for decision service development, rule management and authoring, and the execution environment. On one hand, Decision Server provides tools development and runtime components. It mainly consists of:

- The Rule Designer, which is an IDE that comes as an Eclipse plugging and provides many capabilities to design, author and test rules. Fig. 5.6 shows an in-depth view of the rule designer

- Rule Execution Server is a JEE container that provides the runtime environment for running and monitoring decision services and ensuring a transactional propagation and security control. The console is a web-based application to manager ruleset of the server.

On the other hand, Decision Server also interacts with Decision Center where business users can author, manage, validate, and deploy decision services relatively independently from IT specialists. The rule designer provides necessary tools to develop all rules artifacts, such as OMs, rule projects and rule flows. Fig. 5.6 depicts a general view of the rule designer with some rule artifacts. As far as ODM is concerned, the executable data model discussed in Section 5.2.1 is called eXecutable OM (XOM) and the business layer that defines the vocabulary used by BRs is called Business OM (BOM). The XOM corresponds to the physical data model that references the application objects [3] and data, and is the base implementation of the BOM. ODM gives a flexible way to define the XOM from different data sources such as Java, .NET, XML or COBOL. Furthermore, the BOM is a model that is similar to the Java OM and whose elements resonate with those of the XOM. The latter represents the class model against which rules are run. The

[3]These objects are POJOs that the rule engine manipulates as WM elements

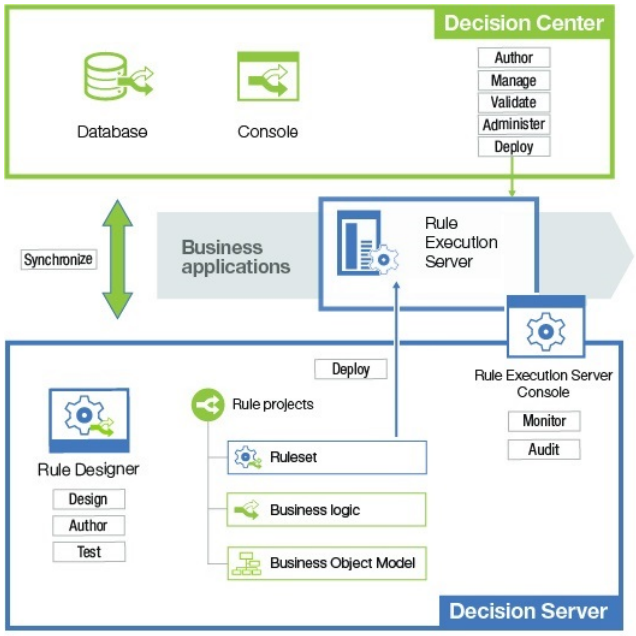


Figure 5.5: ODM Components View

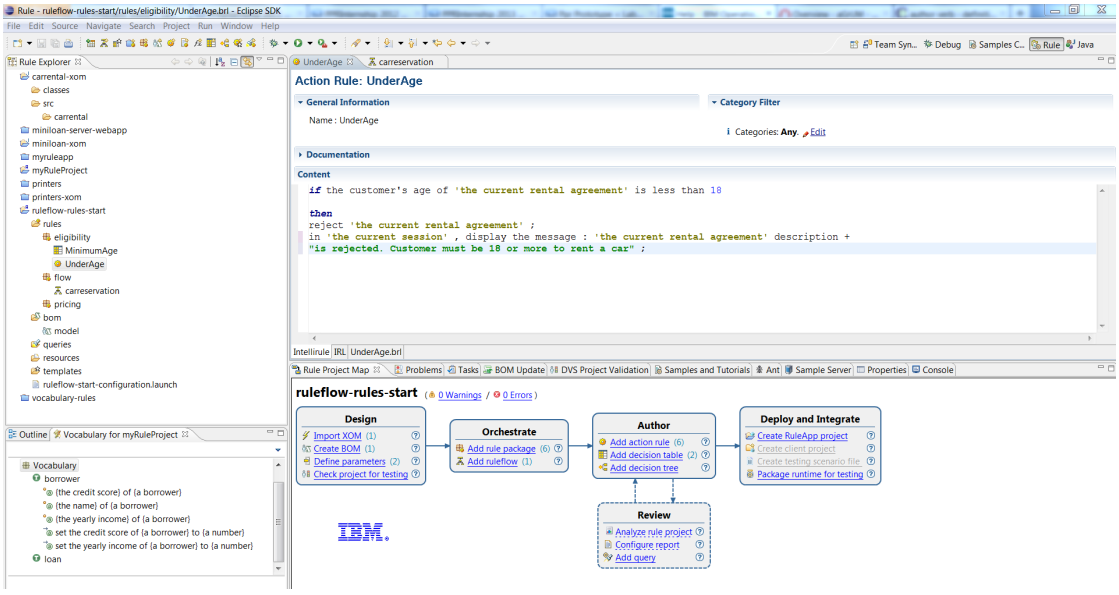


Figure 5.6: A screen view of IBM rule designer

high level language used by business professionals to edit rules is called Business Action Language (BAL). The rules that are defined using BAL are called *action*

rules and they are based on the well known `IF/THEN-ELSE` constructs. BAL is essentially designed to help business professionals to enter BRs in a more human readable format.

Knowledge in the rule engine is built using the ILOG Rule Language (IRL), a Java-like programming language, which is actually the rules technical and execution language in ODM. Therefore, every rule artifact should be translated into IRL as it is the only language the rule engine can understand. Rules that are written using the IRL are called *technical rules* and follow the the general syntax in Fig. 5.7. We would like to emphasize that ODM allows for a BOM-to-XOM mapping mechanism that is based on a specific XML schema. The latter must be provided

```
1  rule ruleName{
2   when{
3    // conditions:
4    <condition>;
5    [<condition>;]*
6    [evaluate (<then/else discrimination test>);]
7   }
8   then{
9    // action if 'when' is satisfied
10   [<action>;]*
11   }
12  [else{
13   //actions otherwise
14   [<action>;]*
15   }]
16 }
```

Figure 5.7: General form of IRL technical rules

at compilation for every rule artifact that is created directly from the BOM in order to be executed at runtime. In this way, ODM provides a semi-automated mapping that allows *action rules* to be translated during the compilation into *technical rules*. However, such an automation may require an IT specialist's insight and intervention to operate.

Now, let us summarize the most common IRL constructs that are used in the condition part, i.e., declared inside the **when** part, and let us give their meaning and a corresponding usage.

- **Class conditions** are the simplest form of conditions. They bind a variable to a pattern, which consists of a type name and, potentially, inside of parentheses, some constraints on the type attributes. For instance the following condition:

```
s: Subscriber(location=="Paris")
```

  binds a variable `s` to patterns from the WM that are matched by any `Subscriber` located in Paris.

- **exists**: this keyword checks the existence of a WME that satisfies the condition. The condition below is satisfied whenever a Subscriber WME located in Paris is found.

```
exists Subscriber(location=="Paris")
```

- **from** is used inside the rule to access ruleset parameters. These are a special type of parameters that are used to exchange data between the application and the ruleset. If we assume that `healthProfessional` is a ruleset input parameter, which is previously defined, then the following condition binds the variable `hp` to that parameter.

```
hp :HealthcareProfessional() from healthProfessional;
```

- **in** restricts the scope of a variable to a collection of values

```
sub  : Subscriber();
reim : Reimbursement() in sub.reimbursements;
```

- **not** statement returns true if there is no WME matching the condition. In order to express the negation of the **exists** operator, using the **not** operator (without **in/from**) is preferable to using **not exists**. For example, the following condition is satisfied when there is no subscriber in the WM located in Paris.

```
not Subscriber(location=="Paris")
```

- **Aggregate**s compute a value from a collection of values. Examples of aggregations are the average, sum, or maximum of a numeric collection. Here is an example of a condition that computes the total amount of all reimbursements. When we want to condition the result of the aggregate, the keyword **where** is used[4].

```
agg:aggregate{
    reimb:Reimbursement();
     }do{
      sum{reimb.price}
      } where{agg>20}
```

The BR engine is a central component in ODM as in every RBS. It manipulates state-full, history-aware Java objects and supports three execution algorithms, namely the Rete algorithm, the sequential algorithm and the Fastpath algorithm.

[4]Note that **exists** can be seen as a special case of aggregate.

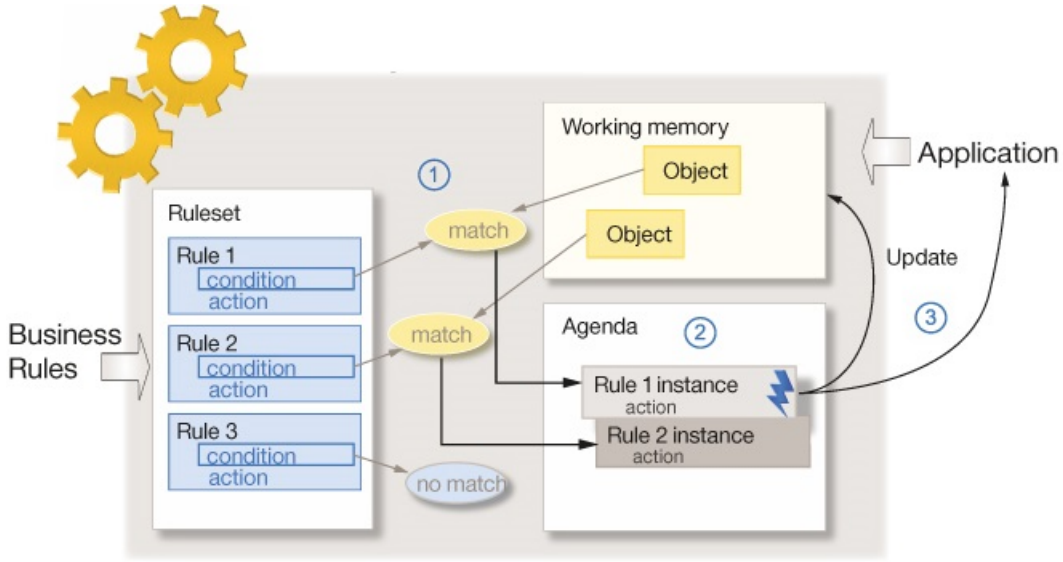


Figure 5.8: ODM rule engine in Rete mode

We give an overview of the first one, a description of the remaining algorithms might be found in [19].

As we already discussed in Section 2.3.2.2, the rule engine compiles the ruleset (RB) into a complex network that orders efficiently rules conditions and we called this network the Rete network. As it is shown in Fig. 5.8, WMEs are later evaluated against this network and if a match is found, an instance of the corresponding rule is triggered in the agenda. The latter is responsible for ordering, selecting and executing rules' instances. Algorithm 7 gives an overview of how the Rete works in ODM. Note that the execution of a rule instance may update the WM. Thus, it triggers a re-evaluation of the rules against the WM.

5.3.2 A Complex Compilation Process

Previous studies initiated the work to investigate how probabilities could be integrated into ODM and proved the feasibility of such an approach [7, 10, 105]. However, they were essentially based on BNs framework, where each class is asso-

Algorithm 7: ODM engine cycle execution in Rete mode

Input : WM, agenda, ruleset

Output: Rule Execution

```
1 repeat
2     Pattern-match condition rule parts with the WM
3     Create an agenda rule instance when there is a match
4     Select rule instance to be run using a conflict resolution strategy
5     Execute the condition part of the selected rule instances // leads to
          adding, removing or updating a WM element
6 until agenda contains no more rule instance;
```

ciated with a BN without no possible reference to probabilistic attributes outside. Also, the definition of the BN cannot be directly inferred from the class definition. Obviously, this is not a realistic use and rules conditions must query only variables of the object being handled. This means a strong correlation between the probabilistic model and the ruleset and this results in redundant BNs definitions. We already discussed some of the limitations facing BNs and we mainly related them to the lack of the object concept, see discussion in Section 3.3. Furthermore, probabilistic engines that are based on modeling only with BNs do not support reference mechanism, which is essential to objects management in BRMSs. Therefore, it is natural to investigate what PRMs can offer in our context.

To begin with, let us show the XOM classes obtained for `Subscriber` and the system of our running example. Consider Fig. 5.9 line 1, the annotation `@PrmClass` is a mark to tell the compiler it is handling a probabilistic class. The latter necessarily contains some probabilistic attributes, which are annotated with `@PrmAttribute`. They carry the information needed to describe their counterpart

PRM attributes. For instance, `age` in line 6 has no parents and a CPT describing whether the subscriber is under the age of 10 is provided. Note that `AgeType` at line 6 is an `Integer` restricted type. `@PrmAgg` marks the attribute as an aggregator. In line 11, the annotation specifies a list of the attribute parents and its CPT. In this example, we implemented the static mode. So, instances are specified as internal attributes of the system class that is annotated with `@PrmSystemClass` in line 15. Reference slots are set inside the class constructor at line 22. Finally, the relational skeleton in Fig. 5.3 is generated from this system class.

The compilation process is based on a series of model re-writings. This is a powerful tool that allows ODM not only to abstract instructions from their implementation, but also to conserve the rule paradigm. Practically, the IRL rules life-cycle is completely separated from that of BAL rules. As a consequence, changing the implementation is possible without altering every BR.

When BRs are entered using the BAL, they are first translated into IRL rules by a rewriting procedure. Second, the resulting ruleset is checked and parsed to obtain a ruleset semantic model as an Abstract Syntax Tree (AST). At this level, the result may undergo recursive rewritings, on top of which one has different types of plugins. Then, the ruleset AST is compiled while taking into account the chosen algorithm. Again, this phase can be parameterized by various plugins according to the algorithm to be used, e.g., Rete. The output at this stage is optimized and transformed to obtain the semantic OM. This latter is a powerful meta-model, it can be seen as an extension of the Java meta-model that allows compilation, sources processing and model definition. There is no longer semantic rulesets here, but, instead, an object model that encodes the rule semantics (condition and action) inside the generated classes and methods. Other operations may appear such as the BAL/IRL mapping and the linkage with outside applications via services mechanisms. The final result is persisted and jitted into an archive that can be

```
@PrmClass
 public class Subscriber{
  @PrmMultiReference
  public Reimbursement[] reimbs;
  @PrmAttribute(parents={},cpt={{.2},{.8}})
  public AgeType age;
  @PrmAgg(name="exists",attribute="reimbs.type",mod="lens")
  public boolean exists_t;
  @PrmAgg(name="exists",attribute="reimbs.price",mod="high")
  public boolean exists_p;
  @PrmAttribute(parents={"age","exists_t"},cpt
     ={{.2,.6,.5,.3},{.8,.4,.5,.7}})
  public boolean risk;
}

@PrmSystemClass
 public class System{
  public HealthcareProvider hp = new HealthcareProvider();
  public Reimbursement[] reimbs  = new Reimbursement[3];
  public Subscriber[] subs = new Subscriber[2];
  public System(){
   hp.subs=subs;
   sub[0].reimbs={reimbs[0],reimbs[1]};
   sub[1].reimbs={reimbs[2]};
   }
}
```

Figure 5.9: `Subscriber` and System classes

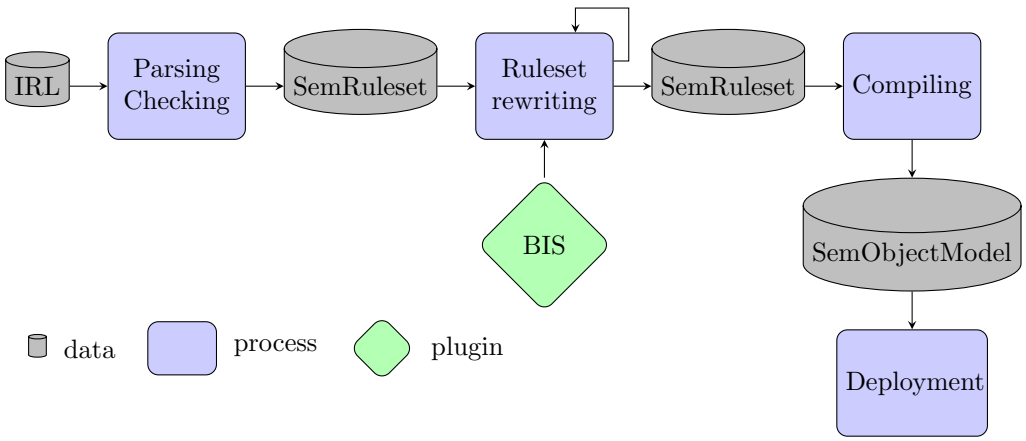


Figure 5.10: ODM compiling chain

deployed in the desired platform, e.g., Java, C# and Script. Note that this chain is executed in pipe-line and the order is controlled by the plugins execution in the chain. Our proposed prototype, called BIS for Bayesian Insight System, can be plugged on top of the rules compilation process as an additional rewriting of the ruleset. The plugging choice is motivated by our desire to take advantage of an existing compilation framework, rather than building such a process from scratch. Additionally, a plugging approach facilitates the conceptual and technical integration in the product architecture. Fig. 5.10 depicts an overall schema of the compilation process. The IRL-based Rule 5.4 illustrates the final results after BIS rewriting the Rule 5.3. It is at this level that we actually call the probabilistic engine.

Rule 5.4: detect invoice anomaly with probability

```
rule detectInvoiceAnomaly{
 when{
  hp:HealthcareProvider(ProbabilisticEngine.this.
     calculateProbability(this,"price_risk", "high") > 0.8);
     }
 then{
```

```
  raiseAlertExceededInvoicePrice(hp);
     }
 }
```

During the compilation of the extended IRL, annotations serve to extract PRM attributes, CPTs and relations from the XOM. When the checking is completed, the final model is written into a PRM text format, called *o3prm*, for Open Object Oriented PRM . The latter serves as input for the probabilistic engine to answer runtime probabilistic queries.

An alternative way to introduce the probability in rules using the IRL `evaluate` operator. Rule 5.5 evaluates the risk that a `Subscriber` is participating in a fraud.

Rule 5.5: evaluate subscriber risk with probability

```
rule evaluatSubscriberRisk{
 when{
  hp:HealthcareProvider();
  sub: Subscriber() in hp.subs;
  evaluate(prob(sub.risk==true)| hp.risk==true)>.8);
 }
 then{
  alertRiskedSubscriber(sub);
 }
}
```

The vertical bar in the rule above stands for "knowing that" and corresponds formally to the conditional probability of a random variable.

In the instruction `prob(sub.risk==true| hp.risk==true)`, the conditional context is explicitly mentioned and it refers to the fact that `hp.risk==true`. However, one might simplify the syntax by considering implicitly every fact in the WM. As a consequence, `prob` should be stateful to facilitate writing the rules without being

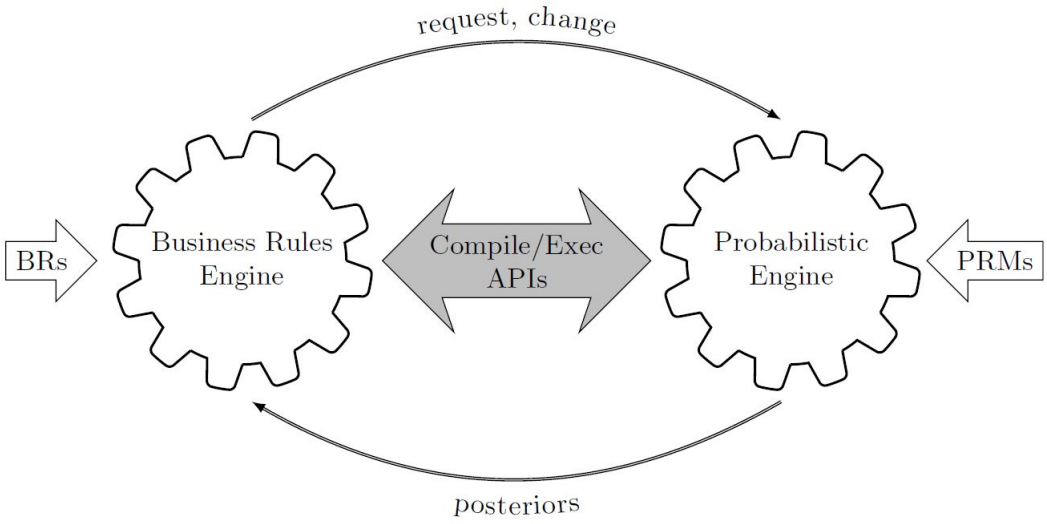


Figure 5.11: BIS coupling

obliged to specify the conditional context at every probabilistic query. Thus, the previous expression can be reduced to `prob(sub.risk==true)`, which is implicitly equivalent to `prob(sub.risk==true|WM)`, here WM is simply `"hp.risk==true"`[5].

5.3.3 A Loosely Coupling-based Execution

A complex PRM system can be easily generated from the rules by means of relating instances of probabilistic classes, which represent a pre-defined BN fragment template. This is an advantage over the classical BNs approach, where the BNs are repeatedly defined. In addition to the compilation API discussed previously, BIS is also endowed with an execution API. Both insure different services communicating following the schema shown in Fig. 5.11. This allows for a coupling between both BRs and probabilistic engines, which are implemented as services. Actually, our framework is not restricted to one implementation, but is open to other probabilistic libraries by means of an adaptive design. For instance, the cur-

[5]One can similarly manage the introduction of other operators such as `mutual information` or `entropy`.

rent work is using aGrUM that can deal with PRMs. We have also tested JSmile [15] as a probabilistic engine. However, we were limited by the lack of relations and object concepts in such a framework. Recall that the compilation part performs a rewriting from the rules semantic model, which encompasses probabilistic data, to runtime functions, which actually call the probabilistic engine[6]. In our case application, the PRM is generated by a XOM compilation and read by the probabilistic engine. It is also possible to read both models from external files. Furthermore, our architecture allows for a good inter-operability between both platforms. On one hand, rules execution can change the state of the WM and consequently the random variables in the PRM system through incremental operations, i.e., add/remove evidence, objects or relations. For instance, the action part of Rule 5.6 updates the WM by posting an evidence on the `risk` attribute.

Rule 5.6: detect invoice anomaly with probability

```
rule detectInvoiceAnomaly{
 when{
   hp:HealthcareProvider( prob(type_risk==high)>.8) );
   sub: Subscriber() in hp.subs;
 }
 then{
   update sub{risk=true;}
    }
}
```

On the other hand, when a rule triggers the probabilistic inference, the probabilistic engine computes the needed probability for the query and may also notify the WM to update some attributes values in order to re-evaluate rules.

In Fig. 5.12 engines are related via an observation mechanism and both are no-

[6] The current version of BIS does not support all the probabilistic engine functions.

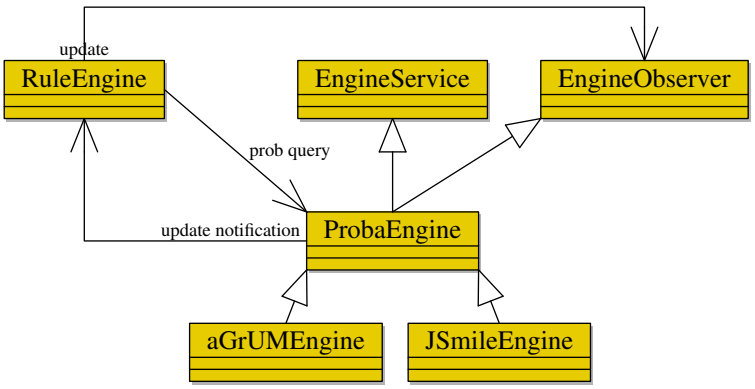


Figure 5.12: PRM plugin as a service

tified, through the observer, when any change occurs in the WM. Each time a WM incremental operation occurs, the underlying PRM is notified for the update. Therefore, the probabilistic engine should operate in a lazy mode, that is, it must record every incremental WM update until the next probabilistic query. Then, it accordingly updates the PRM system and performs inference. After converting the PRM system into a grounded BN, the IJTI algorithm 5 is used to optimize the corresponding JT re-evaluation by only recomputing what has been invalidated between two calls to the probabilistic engine.

5.4 Towards Advanced techniques

In this section, we provide foundations for an alternative approach to handle rules uncertainties using a global probabilistic operator. In the sequel, we intentionally ignore the action part, i.e., **then/else{..}** declarations, in rules examples as it is irrelevant to our discussion.

5.4.1 Preliminary

So far, the approach we implemented to express rules uncertainties is to replace every Boolean test over condition variables with a Boolean test over the proba-

bility of their corresponding random variables. In this scenario, every condition is associated with a non negative parameter α_i, as in the following rule, which is based on the model in 5.2.

```
rule findSubscribers{
 when{
  hp : HealthProfessional(probability(age>50)≥ α1);
  sub: Subscriber(probability(loc=="Paris")≥ α2);
 }
}
```

There are at least two reasons why this approach is limited. First, we actually hide the richness of rule constructs and operators and cut of their possible synergies with the probabilistic engine operators; we will detail further this point. Here, business users are prevented of expressing their confidence degree, more generally, on a joint of a subset or the whole of variables in the condition part such as in the following condition:

```
Subscriber(probability(location=="Paris" or "NY" )≥ α2);
```

Secondly, as the majority of business users are interested in decision-making, not in how probabilistic queries are done, they may be confused if they have to deal directly with programming or mathematical components. Transparency w.r.t. probabilistic inference becomes since then strongly desirable to hide the computational complexity that lies therein. Therefore, another interesting way to express probability in the rules is to parameterize the whole condition part, i.e., the **when{..}** declaration, by a probabilistic activation threshold. In practice, we introduce a general probability operator that governs rules eligibility by testing if the probability to satisfy all the rule conditions equals or exceeds a threshold α[7].

[7]Here we assume that α is given, otherwise it might be inferred or computed from rules context.

Rule 5.7: Find subscriber located in Paris for a certain professional

```
rule findSubscribers{
 probability ≥α ;// α is a non negative constant
 when{
  hp : HealthProfessional(age>50);
  sub: Subscriber(location=="Paris") in hp.subscribers;
}
```

For example, the condition in Rule 5.7 binds a variable `hp` to `HealthProfessional` WME whose age is more than 50 and a variable `sub` to a `Subscriber`, which is located in Paris and belongs to `hp`. When the probability of satisfying those two conditions exceeds α, then this rule is activated. The probability operator can be seen as a generalization of the Boolean case, for which rule actions are executed when the conjunction of the rule conditions is satisfied. In this case, the threshold is simply equal to 1 to express the fact that we are certain about the conditions. The probabilistic operator, which is now put on top the rule conditions, is not to be confused with certainty factors à la MYCIN, where each rules is associated with a factor to show the confidence one has in the rule. Indeed, here we only move probabilistic calls from inside the rules and put them on top of them. However, the probabilistic computations that we carry should be based on a probabilistic inference.

The difficult point lies in propagating this operator on the conditions, as it is moved to a higher level, and the impact of this move on finding a corresponding probabilistic query. Only the compilation process could tell the probabilistic engine how to manage variables that are mentioned in the rule conditions and how to regard non probabilistic ones. This is actually a challenging task that involves a complex compilation process including parsing probabilistic conditions,

extracting relevant variables and transforming results into an adequate probabilistic query. For example, in Rule 5.7, a joint probability distribution over `hp.age, sub.location` should be evaluated and the rule is equivalent to:

```
rule findSubscribers{
 when{
  hp : HealthProfessional();
  sub: Subscriber() in hp.subscribers;
  evaluate(p:probability(hp.age>50,sub.location=="Paris"),
           p≥ α);
  }
 }
```

In the following, we will try to shed light on the compilation challenges and give some directions to propagate the probability operator on rules conditions. Of course, the interpretation of the probabilistic operator will depend on the IRL constructs we use, e.g., class conditions, generators and aggregates, and the way they are combined together within rule conditions.

Now let us try to give a probabilistic interpretation of each construct alone, then see what happens when aggregation and other type of conditions are combined. We begin by emphasizing that since **in** and **from** generators restrict the condition evaluation scope, we grant them a higher precedence than the probability operator. This could be naturally done, as the rule engine is responsible to trigger the probabilistic inference and to specify the target variables from declaration of **from** and **in** generators.

A clause is a disjunction of logical atoms, atoms hereafter, or their negation. Where an atom is simply a predicate applied to a tuple of arguments. An atom is

characterized by the fact that it cannot be broken any further. As a consequence, a rule condition can be expressed in terms of a conjunction of one or more clauses. For example, the condition:

```
subscriber1.age<21 and subscriber1.location in {"Paris,NY"}
```

is composed of two clauses, where the first clause consists of one atom and the second consists of two atoms. More generally, we can use a CNF to represent the conditions of a particular rule.[8] Although this is not a necessary transformation, we use it to facilitate rule conditions representation and to simplify illustrations throughout this section. In this previous example, we have :

$$CNF_1 = a_{11}(age, 21) \wedge (a_{21}(location, Paris) \vee a_{22}(location, NewYork))$$

where :

- $a_{21}(location, Paris)$ = "location is Paris"
- $a_{22}(location, NewYork)$ = "location is New York"

More generally, if we drop atoms arguments, then: $CNF_i = \wedge_{j=1}^{n_i} \vee_{k=1}^{m_{ij}} a_{ijk}$, with n_i (resp. m_{ij}) is the number of clauses in the i^{th} condition (resp. of atoms in the j^{th} clause of CNF_i.

5.4.2 Probabilistic Propagation

In this section, we mainly base our discussion on the rules model in Fig. 5.13. To simplify our discussion, classes $\mathcal{X}$ and $\mathcal{Y}$ represent IRL classes and their corresponding PRMs at the same time. We omit probabilistic annotations and we provide a possible PRM classes dependency graph in Fig. 5.14. We begin with

[8] We emphasize that obtaining a CNF from a Boolean expression can lead to an exponential explosion of the formula and the conversion runs in exponential time in the worst case.

```
1  // probabilistic classes
2  class 𝒴 {
3          int Y1,Y2; // probabilistic attributes
4  }
5  class 𝒳 {
6          int X1,X2; // probabilistic attributes
7          𝒴 y;// a reference
8  }
9  // standard class
10 class B {
11         int b1;//non probabilistic attributes
12 }
```

Figure 5.13: A rule class model with (non) probabilistic attributes

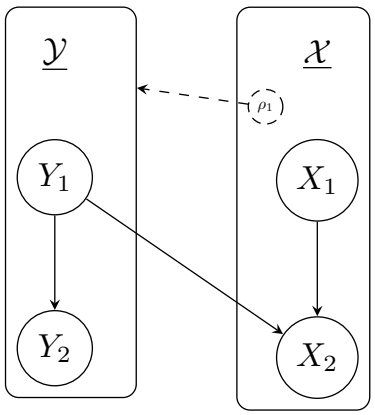


Figure 5.14: A possible dependency graph for PRM classes $\mathcal{X}$ and $\mathcal{Y}$

propagating the probability operator in the presence of standard rules and then we see what happens when other constructs such as aggregates are used. We use interchangeably $o.X_1$ to denote the probabilistic attribute X_1 of the object o and its corresponding random variable's instantiation. Once the PRM system is known,

i.e., objects are known and reference slots are correctly set, the ground BN[9] is not necessarily connected (isolated sub-graphs) and we emphasize that it contains random variables corresponding to attributes of different classes. At the end, two random variables obtained from the instantiation of the same class attribute are regarded as different BN nodes.

5.4.2.1 Simple conditions and a general approach

In this situation, only class conditions appear in **when {...}** parts. The propagation of probability operator is straightforward and the joint posterior $\mathbb{P}(\wedge_{i\geq 1} CNF_i)$ needs to be evaluated. However, we need to introduce some *functional nodes* to compute new random variables that result from applying predicate in a_{ijk}. A functional node may involve some algebraic operations such as random variables additions or multiplications. Consider Rule 5.8.

Rule 5.8: a rule with simple conditions and the probability operator

```
1 Rule r8 {
2  probability≥ α
3  when{
4   b:B();
5   y:Y (b.b1 × Y1 − Y2 ≥ 0);
6  }
7 }
```

Let $\beta = b.b_1$. In this example, we have $a_{211}(Y_1, Y_2, \beta) = \beta \times Y1 - Y2 \geq 0$ and the CPT of $Z = \beta \times Y_1 - Y_2$ should be computed. In particular, in order to check the condition in line 5, we have to evaluate the following:

$$\mathbb{P}(Z \geq 0) = \sum_{\substack{z_i \in Val(Z) \\ z_i \geq 0}} \mathbb{P}(Z = z_i)$$

[9] see Definition 3.3.9

In reality, it is frequent that the rule engine interacts with different sources of information, e.g., a database, which can supply the WM with non probabilistic objects. The rule's OM itself may contain non probabilistic classes, e.g., the B objects in the previous example. This is actually an issue since we loose the genericity when defining the new random variables resulting from functional nodes, e.g., every time the value of β changes, a new probabilistic query needs to be defined. To fix this issue, a solution may pre-compile the probabilistic query into parameterized functional nodes. The latter hold a memory for non probabilistic values and instantiate it at the execution[10].

Fig. 5.15 depicts a simplified general case where we have one CNF with two clauses and each clause has two atoms. Without loss of generality, we furthermore limited arguments of each atom predicate to two arguments, e.g., atom a_{111} has arguments t_1 and t_2 and results in a random variable that is also denoted by a_{111}. Each node in Fig. 5.15 represents a random variable, in particular, or_i (resp. and) is a (probabilistic) OR-gate that is obtained as a disjunction (resp. conjunction) of their parents a_{i1} and a_{i2} (resp. or_i), $i = 1, 2$. Here we have $CNF_1 = (a_{111}(t_1, t_2) \vee a_{112}(t_3, t_4)) \wedge (a_{121}(t_5, t_6) \vee a_{122}(t_7, t_8))$. Note that the BN fragment of Fig. 5.15 is actually a class level extension of the associated PRM and needs to be instantiated and linked to the PRM system as soon as we proceed new WMEs, i.e., for a new WME, the rule engine must trigger a new instantiation of the BN fragment in the PRM system and potentially drop the relations created for specific instances of previous evaluations. In the case of several CNFs, we need to concatenate fragments akin to those obtained in Fig. 5.15 using several AND-gates.

Based on the previous discussion, some issues arise regarding the introduction

[10]this is similar to a database *placeholder/wildcard*, when we prepare before executing a query that runs several times but with different values.

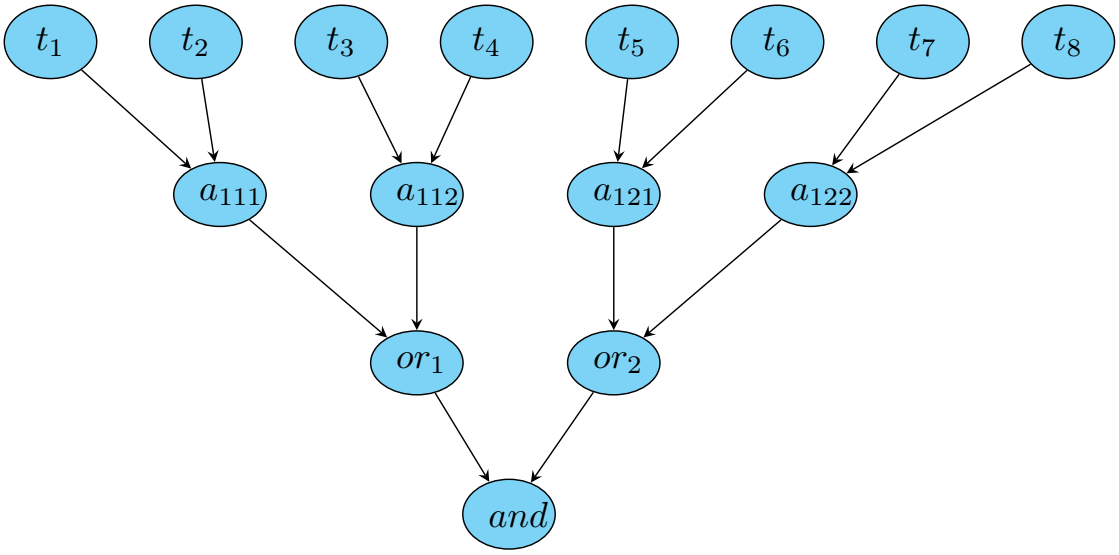


Figure 5.15: A BN fragment obtained from analyzing rule conditions

of functional nodes. First of all, the rule compiler must at least analyze each atom and compute the expression of the resulting random variables. Then the difficult task might be the generation of the corresponding CPTs, depending on the complexity of the expression in a_{ijk} and whether the rule compiler is responsible for such computations or the probabilistic engine. In this last case, the probabilistic engine must be extended to allow for an automated definition of functional nodes, which means to introduce new (parametrized) instructions and operators. Second, introducing functional nodes implies a new model definition and we need to extend the probabilistic engine to support an incremental/evolving PRM and its system. This means that we allow for an overlay that updates the PRM, as soon as we compile rule conditions, and its system, while executing rules actions. As we can see in Fig. 5.15, the posterior we are looking for, $\mathbb{P}(CNF_1)$, is obtained by querying the leaf node *and*.

We emphasize that the transformation process we carried out to obtain a leaf node, grows with the Cartesian product of the conditions and this may lead to scalability issues. A simplified example of this situation is the following:

```
1 Rule r2{
2  probability≥ α
3  when{
4   b:B();
5   y:𝒴(a_111(Y_1,Y_2,β));//β = b.b_1
6   x:𝒳(a_211(X_1,X_2));
7  }
8 }
```

Here we need to iterate the same process as before and construct the leaf target as the conjunction of leaves obtained from processing ling 5 and 6.

It remains to point out that even if we want to move probabilistic calls from inside a **when{...}** declaration, it is still possible to evaluate the probability of some particular clauses or atoms inside the condition part besides the global probability operator's use. Such a situation occurs when we want to constrain particular variables in addition to the global constraint we have on top of the condition part. This is naturally to be given a higher evaluation priority:

```
Rule r3{
 probability≥ α
 when{
  x: 𝒳(a_111(ℙ(X_1 ≤ 2),X_2));
 }
}
```

At the end, we apply the general probability operator to the atom $a_{111}(X_2, p)$, where p is the result of the probabilistic inference $\mathbb{P}(X_1 \leq 2)$.

5.4.2.2 Rules with Existence conditions and Aggregates

In this context, the extension we made in Section 5.2.1 becomes a specific case. There, the class condition corresponds to one atom with an equality predicate and the probabilistic aggregate **exists** was added to the model [11], e.g., a condition based on the model presented in 5.13 would be:

```
exists  𝒳(X1 == 2);
```

However, this is not always the case and class condition may be composed of a conjunction/disjunction of many atoms, which can be expressed using CNFs having general predicate expressions as shown in the previous section. Again we need to introduce new nodes and transformations, whose complexity depends on the complexity of the expressions at hand. Furthermore, to simplify this study, we suppose that we know exactly how many WMEs we have. However, in practice, this is often known at runtime and also evolves with incremental operations that affect the WM due to the execution of rules actions. Therefore, we need to be able to change incrementally the auxiliary nodes as long as we change the WM. This occurs especially when we deal with aggregates and we must be able to change parents of auxiliary variables nodes without recomputing everything from scratch, as we will see in **exists** handling. Thus, a solution for this issue is to define new types of probabilistic aggregate, such as **greaterThan**, which tells whether there exists a parent greater than a given value.
As far as the probabilistic inference is concerned, IJTI algorithm comes into play and allows for an inference that is adaptive to changes we made in the WM.

Based on the remark that the **exists** condition can be converted into a disjunction, we can follow a reasoning similar to what we held for standard rules and we

[11] Even in that simple case, we may notice a combinatorial explosion depending on the number of the aggregate parents, which of course depends on the size of the WM.

will be able to reduce **exists** and its negation **not** to computing some functional nodes. Let us illustrate this through an example based on the class model 5.13. Consider the simple Rule 5.9 with two clauses and one atom for each.

Rule 5.9: an IRL rule with probability operator and one *exist*

```
1 Rule r9{
2  probability≥ α
3  when{
4   exists  𝒳(X1 ≥ β1 ∧ X2 ≤ β2);
5   }
6 }
```

Assume for example that we have three $\mathcal{X}$ WMEs : x_1, x_2 and x_3. In order to evaluate the condition in line 4, the compiler must translate beforehand the expression into a query of a new target. Then, we need to add this latter, at runtime, into the PRM system. Again, we need some auxiliary random variables to define this target. For instance, $lt_2\beta_2$ is a random variable introduced to test whether its parent $(x_2.X_2)$'s value is less than or equal to β_2 and hence, it takes two values as we can see in Fig. 5.16.

Fig. 5.17 depicts the final network obtained. In particular, it shows that we need to query the leaf node *or* to see whether the condition in line 4 of Rule 5.9 is satisfied.

x2.X2		
0	1	2
0.4	0.0	0.6

	lt2Beta2	
x2.X2	false	true
0	0.0	1.0
1	0.0	1.0
2	1.0	0.0

Figure 5.16: CPTs of $x_2.X_2$ and $lt_2\beta_2$, with $\beta_2 = 1$ is given

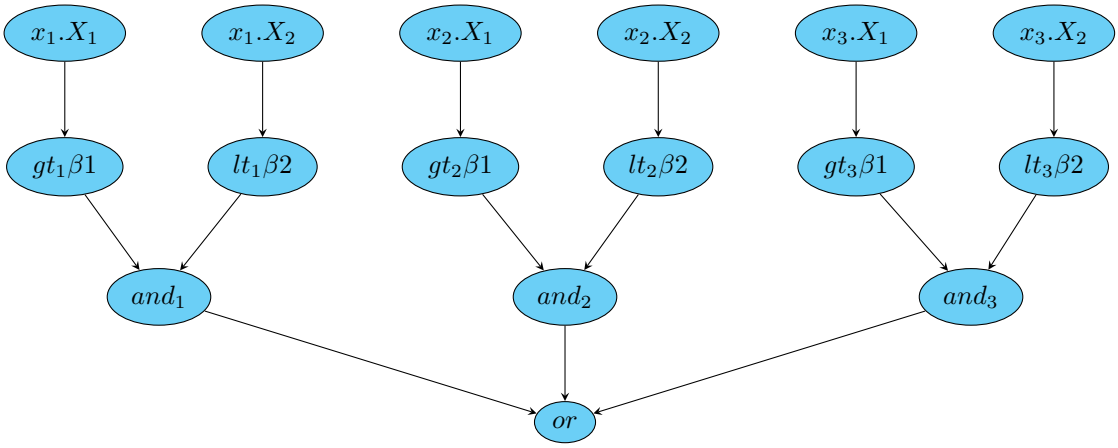


Figure 5.17: A runtime BN fragment obtained after compiling the existence condition in Rule 5.9

We would like to emphasize that after converting the **exists** condition into disjunctions, we are required to apply the same process we held for simple conditions in the previous paragraph. The difference here is that we deal directly with instances. As we can see through this simple example with one simple condition (two clauses) and three WMEs, compiling probabilistic query from a rule condition may result in some difficult transformations and computations that cannot be easily automated. The complexity increases more and more if the **when{...}** body involves a combination of class conditions with existence conditions, which may lead to intractable computations due to the size of the WM and the type of the conditions we use. In this case we need to convert all **exists** and combine the result with fragments obtained from processing other class conditions. The following rule shows an example of such a situation, where we have to construct a conjunction between the leaf obtained from processing line 5 and the one obtained from line 6.

```
1 Rule r5{
2  probability≥ α
3  when {
```

```
    b:B();
    x:𝒳(a111(X1, X2));
    exists 𝒴(a211(Y1, Y2, x.X1, b.b1));
  }
}
```

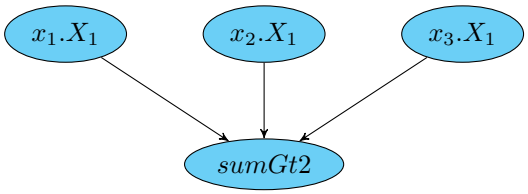
Note that in the Boolean case, one can halt the pattern matching search in an existence condition as soon as a WME is found. However in the probabilistic case, we need to construct a graph that covers all the WMEs to evaluate the **exists** condition and again, this may lead to a huge amount of computations.

Now, we discuss how rules aggregates could be processed and let us begin with a simple example as in Rule 5.10.

Rule 5.10: An IRL rule with aggregate **sum**

```
Rule r10{
 probability≥ α
 when{
  agg:aggregate{
    x:𝒳();
  }do{
    sum{x.X1};
  } where(agg ≥ 2);
 }
}
```

A straightforward approach consists of parsing the rule and detecting the aggregate keyword and the sum operator. Then it enriches the PRM system at the execution with a functional node that computes the sum and compares it to **where** condition. Assume that we have three $\mathcal{X}$ WMEs and, only for the purpose of the



(a) creation of a function node $sumGt2$

			sumGt2	
x1.X1	x2.X1	x3.X1	false	true
0	0	0	1.00	0.00
1			1.00	0.00
0	1		1.00	0.00
1			0.00	1.00
0	0	1	1.00	0.00
1			0.00	1.00
0	1		0.00	1.00
1			0.00	1.00

(b) $sumGt2$'s CPT

Figure 5.18: A BN fragment obtained when compiling Rule 5.10

example, the variable X_1 takes only two values. Then Rule 5.10 is equivalent to query random variable $sumGt2$ in Fig. 5.18a, where $sumGt2$ takes the true value when the sum on its inputs is greater than or equal to 2 and false otherwise, and whose parents are all $\mathcal{X}$ WMEs. We give the CPT of $sumGt2$ in Fig. 5.18b. Again, it would be an interesting enhancement if the rule compiler or the probabilistic engine could also automatically generate such a CPT, because it will be impracticable to build it manually as the number of parents grows. Just to see what happens if we want to actually evaluate the probability, we assume that the BN fragment in 5.18a is isolated and we have an evidence $x_2.X_1 = 1$, then the condition in line 8 is satisfied when $\mathbb{P}(sumGt2 == true | x_2.X_1 = 1) = 0.5295 \geq \alpha$, as shown in Fig. 5.19. Therefore, we can see through this simple example that finding an equivalent probabilistic query might not be an easy task. First, we need to check whether the aggregate is supported by the probabilistic engine or not. In the latter case, we need to implement it based on the compiler analysis of the rule conditions, e.g., $sumGt2$ is obtained by parsing the **aggregate** declaration and

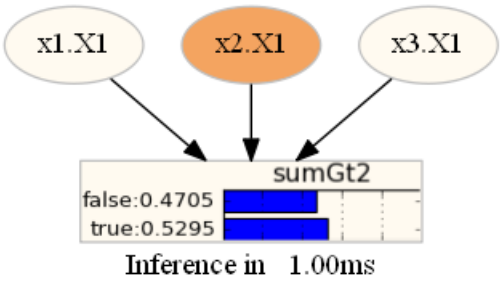


Figure 5.19: Probabilistic inference result, with evidence $x_2.X_1 = 1$

the **where** condition. Second, the analysis of clauses in the condition will simplify this task or make it hard because we may introduce some new nodes as we did in the case of standard rules.

In the presence of many aggregates each one will result in a complex node to be combined with nodes from other conditions. Having this discussion in mind, rules such as Rule 5.11, will result in complex computations including parsing rules, defining functional nodes and putting all together to finally find an equivalent probabilistic query.

Rule 5.11: A rule with two aggregates

```
1  Rule r11 {
2   probability≥ α
3   when {
4    b:B();
5    agg1:aggregate {
6        x : X(a111(X1, b.b1));
7        }
8    do{
9        sum{x.X2};
10       }
```

```
11    where(agg1 ≤ 2 );
12    agg2:aggregate{
13        y : 𝒴(a211(y.Y1, b.b2));
14        }
15    do{
16        sum{y.Y2 + agg1}
17        };
18    where(agg2 ≥ 5);
19   }
20 }
```

In the presence of nested aggregates, one solution would be to evaluate the most enclosed one and combine the output with the superior aggregate block and so on. We want to conclude this discussion by emphasizing that there exists some supported PRM aggregates such as **min, max, forall and, average** that can be mapped directly to their counterparts in the rules after computing functional nodes when needed.

5.5 Conclusion

This chapter introduced an effective approach to integrate probabilistic reasoning into modern BRMSs. The solution we proposed is based on a loosely coupling between BRMSs and PRMs. We also highlighted the natural mapping between both paradigms and gave an operational method to assure it. Then, after describing the ODM BRMS, we proposed a general architecture of the coupling platform and described our technical contribution, which is implemented as a prototype on top of ODM.

We concluded this chapter with a discussion on generalizing the use of the probability operator. The point is to move any technical probabilistic (mathematical or programming) construct from the condition definition and put it on top of the rule. Some issues arise regarding the propagation of such an operator. Thus, we propose a general approach to cope with this issue. In particular, we highlighted through many examples, that the PRM engine has to support functional nodes, including generic probabilistic aggregates and dynamic PRM systems. The rule compiler has also to be extended to parse correctly probabilistic rules. The main challenge here is to produce the runtime probabilistic query. We saw that the output of this task varies from one rule construct to another and its complexity depends on the complexity of the condition expression and the combination of the rule constructs, such as generators, aggregates and different class conditions.

Finally, we know that the execution of a rule action may update, insert or remove WMEs. This clearly has an impact on the corresponding PRM as it may cause its inconsistency. For example, the removal of an object from the WM may unset a reference slot, hence, the PRM becomes ill-defined. This is an issue because, if the PRM is queried at this time the inference answer is clearly unsound. To tackle this problem, we need to impose a transactional mode when the rules engine declares the PRM system or updates WMEs or forbid querying the PRM before completing its correct definition.

Chapter 6

Discussion and Future Works

We conclude this manuscript by summarizing the main contributions and discussing some directions for future works.

Summary

In the last decades, BRMSs emerged as complex platforms that help organizations to effectively manage the rules that drive and guide the business practice. The main idea of such systems is to externalize the business logic, expressed in terms of Business Rules, and to allow business users to be essential and main actors in their life-cycle management. Such an approach gains more and more acceptance among organizations as it provides an agile development of business applications. In Chapter 1, we discuss the business rules approach. If this approach advocates the use of Business Rules Management Systems, it does not provide explicit guidelines about handling uncertainties in the domain. In addition, there is no satisfactory method implemented in current BRMSs to cope with this question neither. However, uncertainty is a pervasive property of data and a good uncertainty representation coupled with a mechanism for reasoning

under uncertainty will render business applications more realistic. This remark motivated our research and our choice for PGMs to cope with the issue of uncertainties in BRMS. In particular we use PRMs, an OO version of directed PGMs, to model uncertainties and answer probabilistic queries. Coupling both frameworks has many advantages. First, they provide a declarative and expressive representation of the knowledge, which is separated from the reasoning process. Second, conceptual similarities (classes, objects and relations) facilitate the mapping of the object model in BRMSs with the probabilistic model in PRMs. In Chapter 3, we present PRMs and provide a discussion about incremental probabilistic inference. Actually, the working memory in BRMSs evolves incrementally and, to ensure an effective coupling with a probabilistic engine, we need to extend probabilistic inference algorithms to take incrementality into account. In Chapter 4, we present, IJTI, a new inference algorithm that deals exactly with such an issue. The key idea of the proposed algorithm is taking into account previous computations to optimize the current inference by computing only what is exactly needed. We then highlight its effectiveness through experimental tests on real and randomly generated data. In Chapter 5, we present a method to couple BRMSs with PRMs exploiting shared similarities. Then we apply our approach to IBM ODM, which is a leading industrial BRMS. We also present the architectural basis and show how our prototype allows for an inter-operability between BRMSs and PRMs.

Future work

The research presented throughout this manuscript opens many other research perspectives. In particular, the IJTI algorithm can be easily adapted to the PRM inference. For this purpose, we need, first, to ensure a transactional mode that guarantees the coherence of the model during updates caused by the working

memory. One should have an automatic consistency checking before probabilistic inference is engaged. Second, the probabilistic inference itself could be optimized by taking into account structural properties encoded in the PRM and lift a part of inference to the PRM class level. This would optimize inference at runtime as we can (re)use class computations at instance level.

Moreover, as soon as we deal with aggregators/generators in the rules, e.g, `from,in`, it becomes necessary to automatically generate their PRM counterpart as and when we compile the rules. This immediately opens the issue of PRMs non-supported operators and the idea of extending this model. Another difficult compilation aspect, yet more interesting, is the use of several operators within the same rule and how this impacts the PRM construction.

After each working memory update, the Rete algorithm tries to optimize the expression re-evaluations, while taking into account the previous state. For example the Rete network shares nodes in the compiled network whenever shared conditions appear in the rules. It would be interesting to develop a two-sided optimization that supervises PRM and working memory updates, e.g, taking into consideration the PRM variables independence. Or better yet, to try to optimize updates propagations as part of a tight coupling but to the cost of algorithmic complexity and finding a trade-off between implementation and performance. Finally, this work also opens doors to introduce uncertain reasoning in rules temporal expressions to process complex events over uncertain data or events.

Appendix A

Proofs

Proof of Proposition 4.3.4: Note that $\mathcal{V}_{\text{-}j}(i) = \{i\} \cup \bigcup_{k \in \text{Adj}_{\text{-}j}(i)} \mathcal{V}_{\text{-}i}(k)$ and, for $k \in \text{Adj}_{\text{-}j}(i)$, $l' \in \mathcal{V}_{\text{-}i}(k)$, we have $Adj_j(l') = Adj_i(l')$. Using Definition 4.3.3, one can thus rewrite $\mu_{i \to j}$ into:

$$\mu_{i\to j} = \bigcup_{\substack{l' \in \mathcal{V}_{\text{-}j}(i) \\ \{l\} = Adj_j(l')}} \lambda_{l' \to l} = \lambda_{i\to j} \cup \bigcup_{k \in \text{Adj}_{\text{-}j}(i)} \overbrace{\bigcup_{\substack{l' \in \mathcal{V}_{\text{-}i}(k) \\ \{l\} = Adj_j(l')}} \lambda_{l' \to l}}^{\mu_{k \to i}} = \lambda_{i\to j} \cup \bigcup_{k \in \text{Adj}_{\text{-}j}(i)} \mu_{k \to i} \qquad \blacksquare$$

Proof of Theorem 4.3.5 – mutual exclusivity: if property 1) is satisfied, then $\mathcal{T}$ contains no edge, therefore properties b) and c) cannot be satisfied.

Now, assume that there exist r_1, r_1' such that $\mu_{r_1' \to r_1} = \mu_{r_1 \to r_1'} = T\epsilon$ (property 2). Let r_2 be any clique in $\mathcal{V}(\mathcal{T})$. Without loss of generality, assume that r_1 lies on the path $i_1 = r_2, i_2, \ldots, i_p = r_1'$ between r_2 and r_1'. Then, by Proposition 4.3.4, $\mu_{i_2 \to r_2} \supseteq \mu_{i_3 \to i_2} \supseteq \cdots \supseteq \mu_{r_1' \to r_1} = T\epsilon$. Therefore, properties b) and c) cannot hold simultaneously. $\blacksquare$

Proof of Theorem 4.3.5 – r's existence: if $\mathcal{A}(\mathcal{T}) = \emptyset$, then property a) holds and r is the unique node of $\mathcal{T}$. Now, assume that $\mathcal{A}(\mathcal{T}) \neq \emptyset$. If there exists an edge $(i, j) \in \mathcal{E}(\mathcal{T})$ such that $\mu_{i \to j} = \mu_{j \to i} = T\epsilon$, then $r = i$ satifies property b). Otherwise, neither properties a) nor b) hold. Assume that property c) neither

holds. Then, for all edges (i,j), exactly one of $\mu_{i\to j}$ or $\mu_{j\to i}$ is equal to $T\epsilon$ and the other one belongs to $\{\emptyset, \epsilon, T\}$. Let (i_0, j_0) be such that $\mu_{i_0\to j_0} = T\epsilon$ and $\mu_{j_0\to i_0} \neq T\epsilon$. Then, if $|\text{Adj}(i_0)| = 1$, clique i_0 satisfies property c), a contradiction. As we assume that property b) neither holds, there exists $i_1 \in \text{Adj}(i_0)$ such that $\mu_{i_1\to i_0} = T\epsilon$ and $\mu_{i_0\to i_1} \neq T\epsilon$. The same reasoning holds for i_1, hence either i_1 is a leaf, which contradicts property c) or i_1 has another neighbor i_2 such that $\mu_{i_2\to i_1} = T\epsilon$ and $\mu_{i_1\to i_2} \neq T\epsilon$. By induction, we create a path $i_1, \ldots, i_n$ of maximal size. This path is necessarily finite since $\mathcal{T}$ is a finite tree, hence clique i_n is a leaf which, therefore, satisfies property c), a contradiction. Consequently, when properties a) and b) do not hold, property c) holds. ■

One can now prove separately the optimality for each property of Theorem 4.3.5, since these properties are mutually exclusive:

Proof of Theorem 4.3.5 – property 1's optimality: r is the only node in $\mathcal{T}$. Choosing it as a root is therefore optimal. ■

Lemma A.0.1. *Let $i, j \in \mathcal{V}(\mathcal{T})$ be such that $\epsilon \in \mu_{j\to i}$ and $\mu_{i\to j} = \emptyset$, then $\forall l \in \mathcal{V}_{-j}(i) : \delta(l) = \delta(j) + len(l-j)$.*

Proof. Note that when $\epsilon \notin \mu_{j\to i}$, $\mathcal{T}$ is up-to-date in the current inference and there is no need to perform any computation. The proof is achieved by induction on $n = len(l-j)$. For $n = 1$, we have $l = i$, so by Equation (4.2) and the fact that $\epsilon \in \mu_{j\to i}$ and $i \in Adj_i(j)$, we get $\delta_{j\to i}(i) = 1$. As a consequence, $\delta(i) = \sum_{(k',k)\in\mathcal{A}(\mathcal{T})\setminus\{(j,i)\}} \delta_{k'\to k}(i) + 1$. Yet, as $T \notin \mu_{i\to j}$ we have $\delta_{j\to i}(j) = 0$; so $\delta(j) = \sum_{(k',k)\in\mathcal{A}(\mathcal{T}))\setminus\{(j,i)\}} \delta_{k'\to k}(j)$. Since $\epsilon \notin \mu_{i\to j}$, $\delta_{i\to j}(i) = \delta_{i\to j}(j) = 0$. For $(k',k) \neq (i,j),(j,i)$, we have $Adj_i(k) = Adj_j(k)$ and $Adj_i(k') = Adj_j(k')$. In this case, it follows that $\delta_{k'\to k}(i) = \delta_{k'\to k}(j)$. We conclude that $\delta(i) = \delta(j) + 1$.

Now suppose this property is satisfied for $n - 1 > 1$, let us prove that it remains true for n. Let l be such that $len(l-j) = n - 1$. Let $\{p\} = Adj_i(l)$.

Then $\delta(l) = 1 + \sum_{(k',k)\in\mathcal{A}(\mathcal{T})\setminus\{(p,l)\}} \delta_{k'\to k}(l)$ because $\delta_{p\to l}(l) = 1$ (since $\epsilon \in \mu_{p\to l}$ and $\{l\} = Adj_l(p)$). Knowing that $T \notin \mu_{l\to p}$, we get $\delta_{p\to l}(p) = 0$, it follows that $\delta(p) = \sum_{(k',k)\in\mathcal{A}(\mathcal{T})\setminus\{(p,l)\}} \delta_{k'\to k}(p)$. Now using the same reasoning as in the case $n = 1$ and by remarking $\delta_{l\to p}(p) = \delta_{l\to p}(l) = 0$ because $\epsilon \notin \mu_{l\to p}$, we conclude that $\delta(l) = 1 + \sum_{(k',k)\in\mathcal{A}(\mathcal{T})\setminus\{(p,l)\}} \delta_{k'\to k}(l) = 1 + \sum_{(k',k)\in\mathcal{A}(\mathcal{T})\setminus\{(p,l)\}} \delta_{k'\to k}(p) = 1 + \delta(p)$. By applying the induction hypothesis on l, where $len(l-j) = n-1$, we obtain : $\delta(l) = 1 + \delta(p) = 1 + n - 1 + \delta(j) = \delta(j) + n$. ■

Lemma A.0.2. *Let $\mathcal{V}_1 = \{r \in \mathcal{V}(\mathcal{T}) : \exists k \in \mathrm{Adj}(r), \mu_{r\to k} = \mu_{k\to r} = T\epsilon\}$, then for any r, r' in $\mathcal{V}_1$ we have $\delta(r) = \delta(r')$.*

Proof. Assume that $|\mathcal{V}_1| > 1$. By Proposition 4.3.4, the nodes in $\mathcal{V}_1$ form a connected subgraph. Let $r, r' \in \mathcal{V}_1$ be such that $(r, r') \in \mathcal{E}(\mathcal{T})$. Finally, let $(k', k) \in \mathcal{A}(\mathcal{T}) \setminus \{(r, r'), (r', r)\}$. If $k' \notin \{r, r'\}$, then either $k = r$, $k = r'$ or $k \notin \{r, r'\}$ and in all these cases we have: $Adj_r(k') = Adj_{r'}(k')$, hence $\delta_{k'\to k}(r) = \delta_{k'\to k}(r')$. Otherwise, let $k' = r'$ then $k \neq r$ and we have also [1] $Adj_r(k) = Adj_{r'}(k)$ and again $\delta_{k'\to k}(r) = \delta_{k'\to k}(r')$. As a consequence: $\sum_{(k',k)\in\mathcal{A}(\mathcal{T})\setminus\{(r,r'),(r',r)\}} \delta_{k'\to k}(r) = \sum_{(k',k)\in\mathcal{A}(\mathcal{T})\setminus\{(r,r'),(r',r)\}} \delta_{k'\to k}(r')$. By Equation (4.2), we get:
$\delta_{r\to r'}(r) + \delta_{r'\to r}(r) = \delta_{r\to r'}(r') + \delta_{r'\to r}(r') = 2$. We conclude that
$\delta(r) - \sum_{(k',k)\in\mathcal{A}(\mathcal{T})\setminus\{(r,r'),(r',r)\}} \delta_{k'\to k}(r) = \delta(r') - \sum_{(k',k)\in\mathcal{A}(\mathcal{T})\setminus\{(r,r'),(r',r)\}} \delta_{k'\to k}(r')$. Hence
$\delta(r) = \delta(r')$. ■

Proof of Theorem 4.3.5 – property 2's optimality: Under the notations of property 2), it is sufficient to prove that for any i not in $\mathcal{V}_1, \delta(r) \leq \delta(i)$ [2]. Without loss of generality, assume that $i \in \mathcal{V}_{\text{-}r'}(r)$. Let $(k, k') \in \mathcal{A}(i-r)$, where $\mathcal{A}(i-r)$ is the set of arcs induced from $i-r$. We either have $\{k'\} = Adj_r(k)$ or $\{k\} = Adj_r(k')$. Assume for instance that $\{k'\} = Adj_r(k), k \neq r$,

[1] if $k' = r$ then $k \neq r'$ and the equality also verified.

[2] all the nodes are computationally equivalent if $\forall i \in \mathcal{V}(\mathcal{T}), i \in \mathcal{V}_1$ since $\mathcal{V}(\mathcal{T}) = \mathcal{V}_1$

the second case should be treated similarly. Then $\mu_{k' \to k} = T\epsilon$ and by applying Equation 4.2, Tab. A.1 summarizes the results we obtain. we conclude that

Table A.1: Computed messages number

$\mu_{k \to k'}$	$\delta_{k \to k'}(i) + \delta_{k' \to k}(i)$	$\delta_{k \to k'}(r) + \delta_{k' \to k}(r)$
$\emptyset$	1	0
T	1	1
ϵ	2	1

$\sum_{(k',k) \in \mathcal{A}(i-r)} \delta_{k' \to k}(r) \leq \sum_{(k',k) \in \mathcal{A}(i-r)} \delta_{k' \to k}(i)$.(1)
Now for $(k, k') \notin \mathcal{A}(i-r)$ it is easy to see that $\delta_{k \to k'}(i) = \delta_{k \to k'}(r)$ and hence :
$\sum_{(k,k') \in \mathcal{A}(\mathcal{T}) \setminus \mathcal{A}(i-r)} \delta_{k' \to k}(r) = \sum_{(k,k') \in \mathcal{A}(\mathcal{T}) \setminus \mathcal{A}(i-r)} \delta_{k' \to k}(i)$. (2).
By comparing (1) and (2) we get that $\delta(r) \leq \delta(i)$ for $i \notin \mathcal{V}_1$. So far, we obtain, by Lemma A.0.2, for any i in $\mathcal{V}_1$, $\delta(r) = \delta(i)$ and for any i not in $\mathcal{V}_1, \delta(r) \leq \delta(i)$, therefore we have $r \in Argmin_{i \in \mathcal{V}(\mathcal{T})}\ \delta(i)$. ■

Proof of Theorem 4.3.5 – property 3's optimality: Let i in $\mathcal{V}(\mathcal{T})$ s.t. $i \neq r$.

first case: $\mu_{Adj_i(r) \to r} = \emptyset$. Assume that $T, \epsilon \in \mathcal{V}_{\text{-}i}(r)$, because otherwise there is no need to perform any computation, as either there is no query or no modification in $\mathcal{T}$; so by Lemma A.0.1 we have $\delta(i) = \delta(r) + len(i-r)$ because $i \in \mathcal{V}_{\text{-}r}(Adj_i(r))$. Hence $\delta(r) < \delta(i)$.

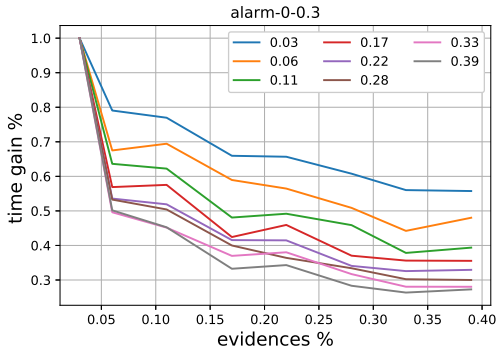
second case: we omit the case $\mu_{Adj_i(r) \to r} \in \{T, \epsilon\}$, but one should use the same methodology as in property 2)'s proof and the fact that for any k, k' in $i-r$ s.t $\{k'\} = Adj_r(k) : \mu_{k \to k'} = \mu_{i \to Adj_r(i)}$ and examine $\delta_{k' \to k}(r)$ and $\delta_{k' \to k}(i)$. ■

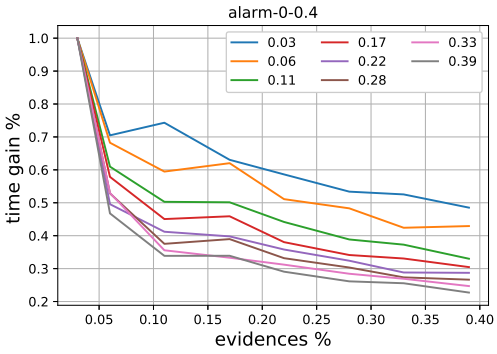
Proof of Proposition 4.3.6 : Given a root r, $\delta_{i \to j}(r)$ corresponds, by construction, to the fact that $\psi_{i \to j}$ is necessary during the current inference and was

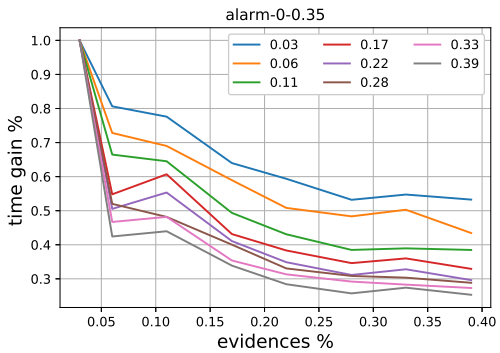
invalidated in the previous one. As a consequence, the current inference needs to recompute only such a message for any i, j in $\mathcal{V}(\mathcal{T})$. ■

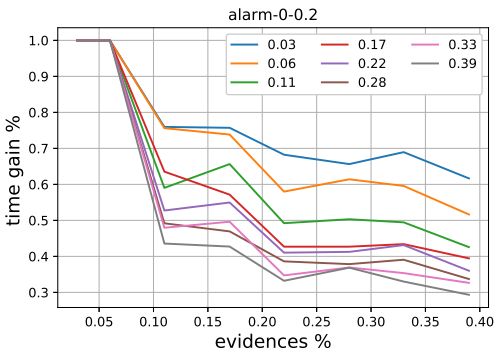
Appendix B

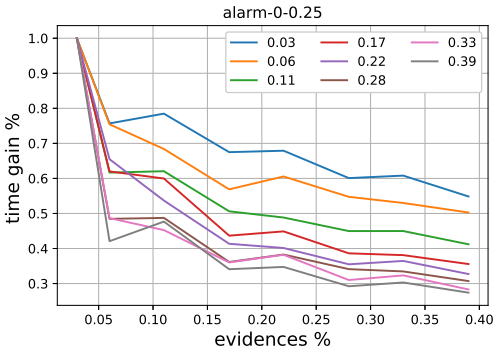
Other experimental results

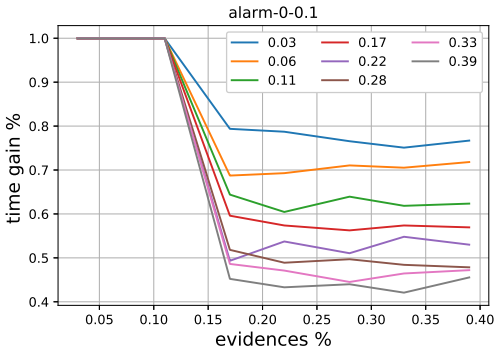


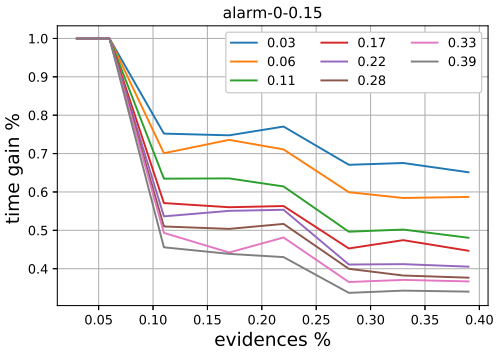


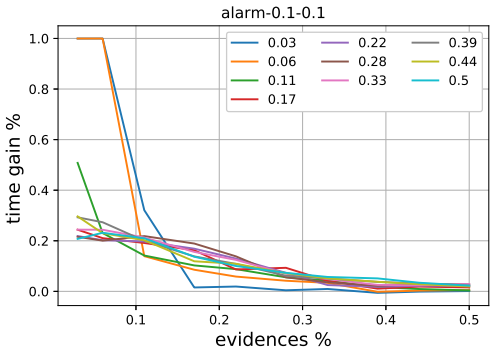


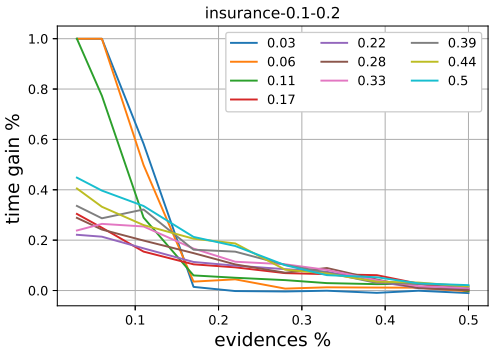


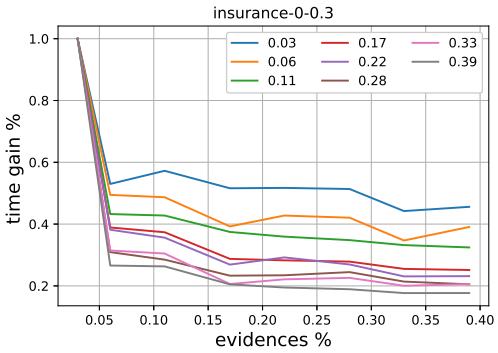

alarm-0-0.25
time gain %
evidences %
0.03
0.06
0.11
0.17
0.22
0.28
0.33
0.39

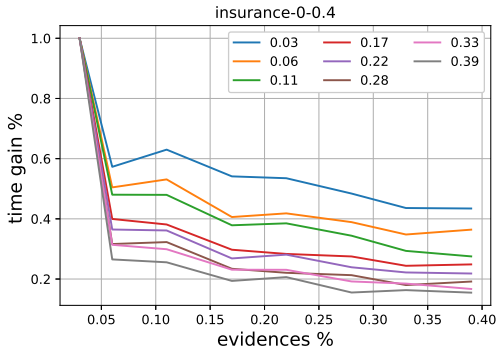

alarm-0-0.1
time gain %
evidences %
0.03
0.06
0.11
0.17
0.22
0.28
0.33
0.39

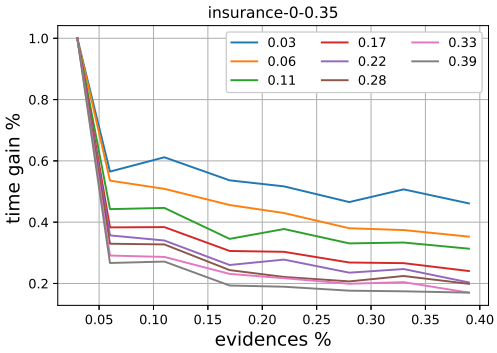

alarm-0-0.15
time gain %
evidences %
0.03
0.06
0.11
0.17
0.22
0.28
0.33
0.39

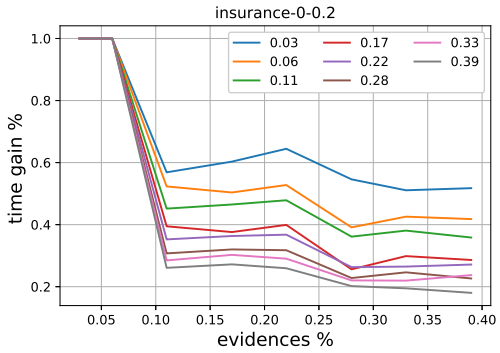

alarm-0.1-0.1
time gain %
evidences %
0.03
0.06
0.11
0.17
0.22
0.28
0.33
0.39
0.44
0.5

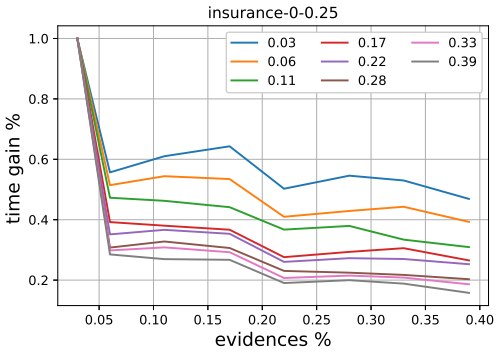

insurance-0.1-0.2
time gain %
evidences %
0.03
0.06
0.11
0.17
0.22
0.28
0.33
0.39
0.44
0.5

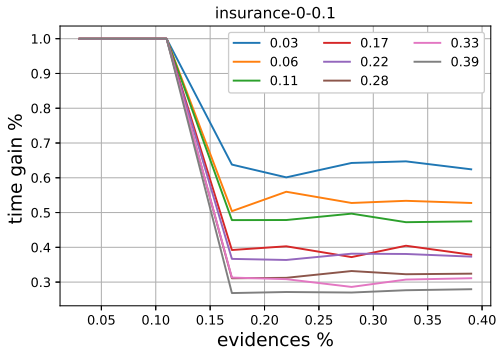

insurance-0-0.3
time gain %
evidences %
0.03
0.06
0.11
0.17
0.22
0.28
0.33
0.39

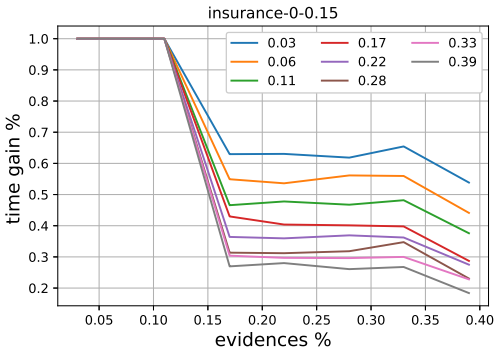

insurance-0-0.4
time gain %
evidences %
0.03
0.06
0.11
0.17
0.22
0.28
0.33
0.39

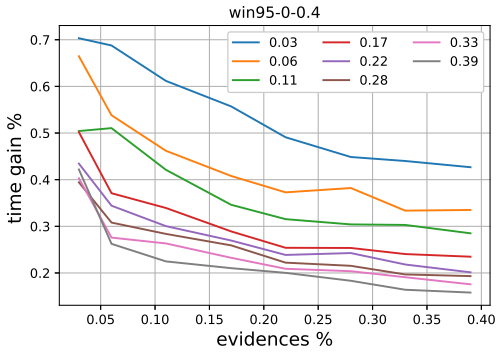

insurance-0-0.35
time gain %
evidences %
0.03
0.06
0.11
0.17
0.22
0.28
0.33
0.39

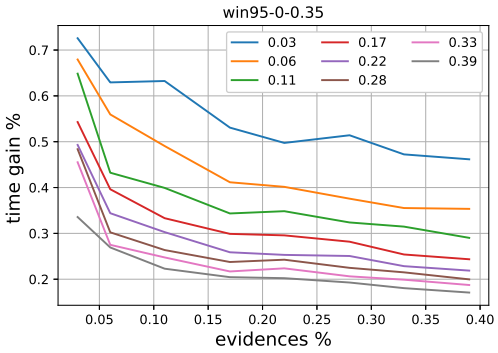

insurance-0-0.2
time gain %
evidences %
0.03
0.06
0.11
0.17
0.22
0.28
0.33
0.39

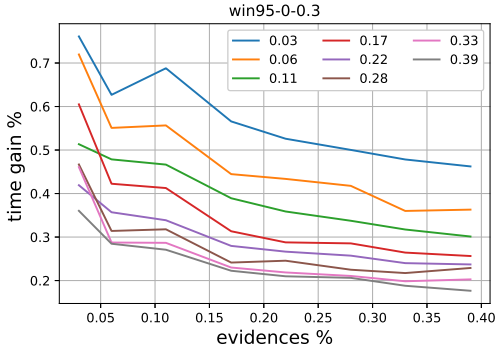

insurance-0-0.25
time gain %
evidences %
0.03
0.06
0.11
0.17
0.22
0.28
0.33
0.39

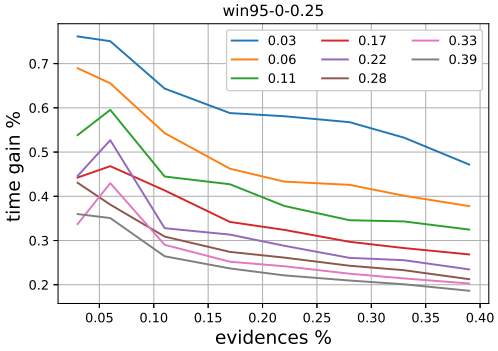

insurance-0-0.1
time gain %
evidences %
0.03
0.06
0.11
0.17
0.22
0.28
0.33
0.39

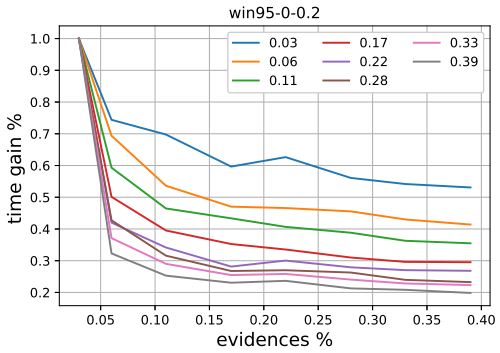

insurance-0-0.15
time gain %
evidences %
0.03
0.06
0.11
0.17
0.22
0.28
0.33
0.39

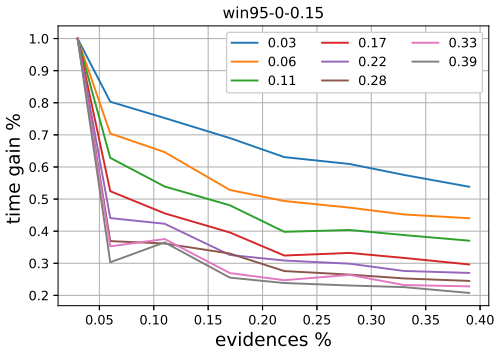

win95-0-0.4
time gain %
evidences %
0.03
0.06
0.11
0.17
0.22
0.28
0.33
0.39

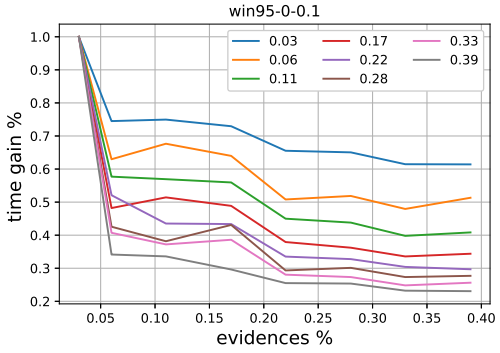

win95-0-0.35
time gain %
evidences %
0.03
0.06
0.11
0.17
0.22
0.28
0.33
0.39

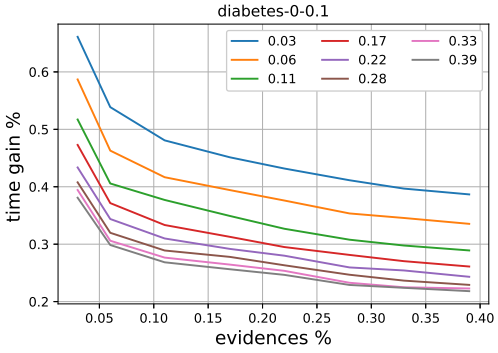

win95-0-0.3
time gain %
evidences %
0.03
0.06
0.11
0.17
0.22
0.28
0.33
0.39

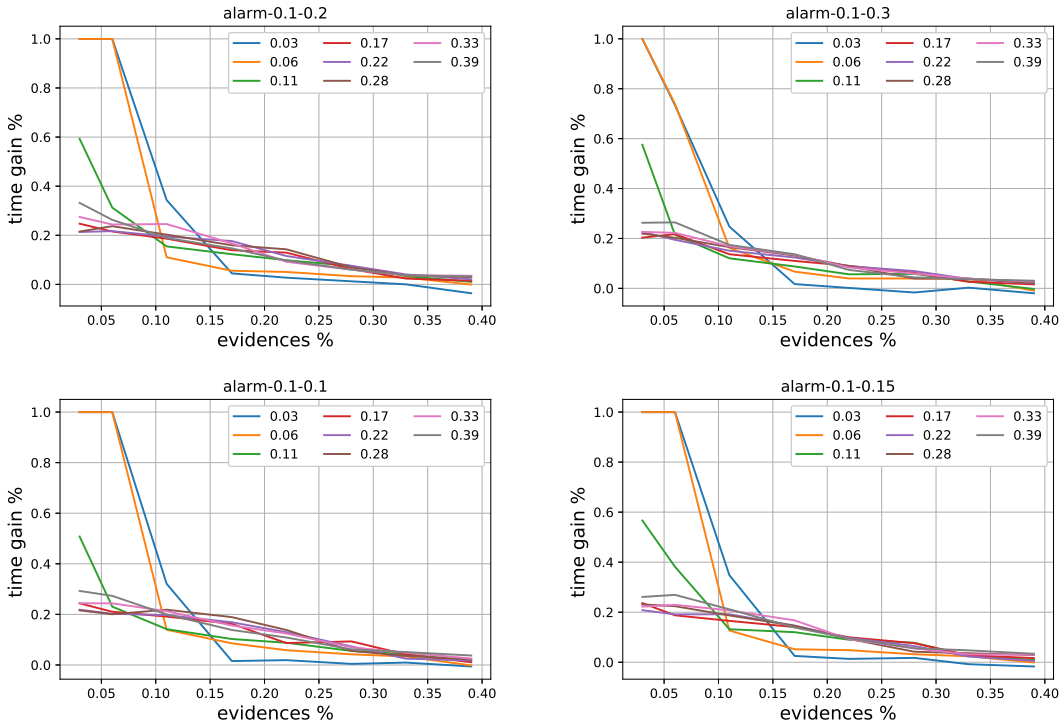

win95-0-0.25
time gain %
evidences %
0.03
0.06
0.11
0.17
0.22
0.28
0.33
0.39

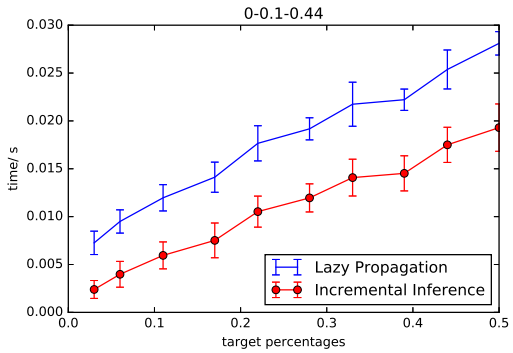

win95-0-0.2
time gain %
evidences %
0.03
0.06
0.11
0.17
0.22
0.28
0.33
0.39

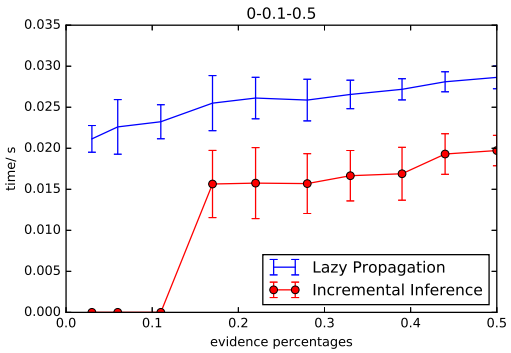

win95-0-0.15
time gain %
evidences %
0.03
0.06
0.11
0.17
0.22
0.28
0.33
0.39

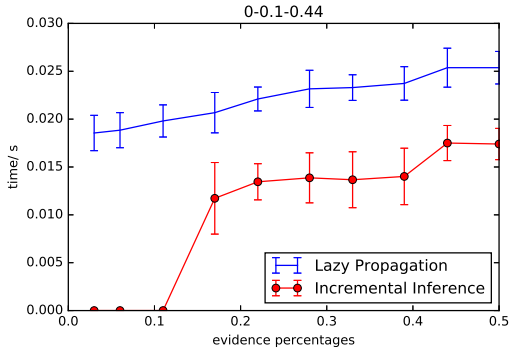

win95-0-0.1
time gain %
evidences %
0.03
0.06
0.11
0.17
0.22
0.28
0.33
0.39

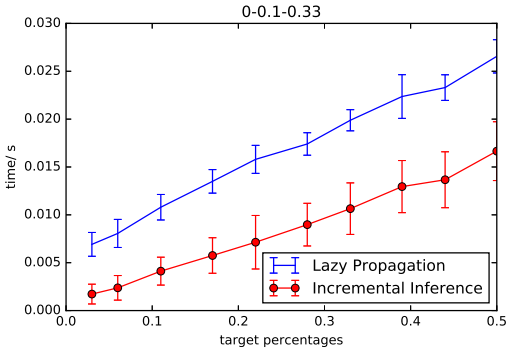

diabetes-0-0.1
time gain %
evidences %
0.03
0.06
0.11
0.17
0.22
0.28
0.33
0.39

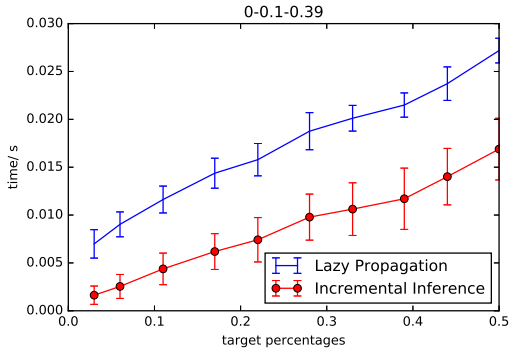

alarm-0.1-0.2
alarm-0.1-0.3
alarm-0.1-0.1
alarm-0.1-0.15
0.03
0.06
0.11
0.17
0.22
0.28
0.33
0.39
time gain %
evidences %
0.0
0.2
0.4
0.6
0.8
1.0
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40

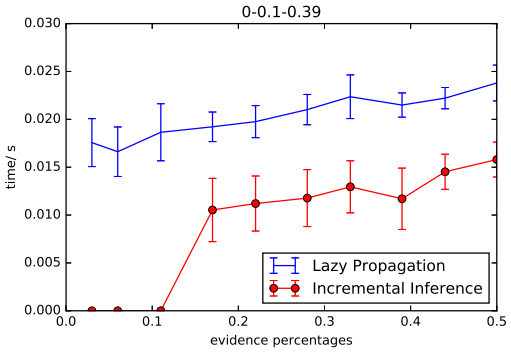

0-0.1-0.44
time/ s
target percentages
Lazy Propagation
Incremental Inference

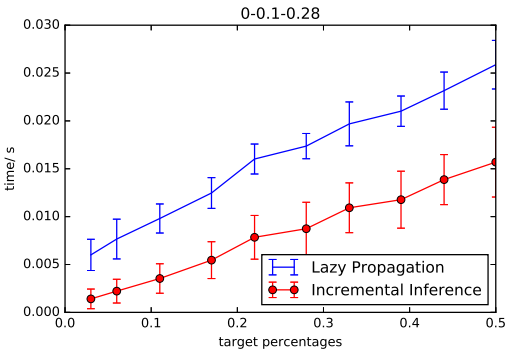

0-0.1-0.5
time/ s
evidence percentages
Lazy Propagation
Incremental Inference

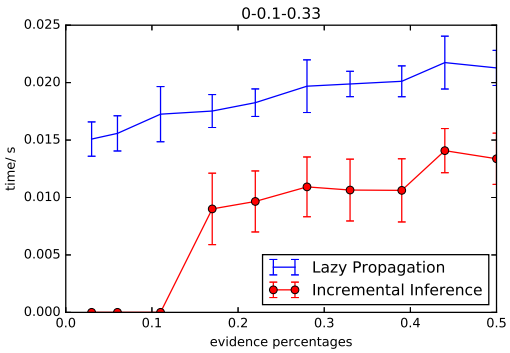

0-0.1-0.44
time/ s
evidence percentages
Lazy Propagation
Incremental Inference

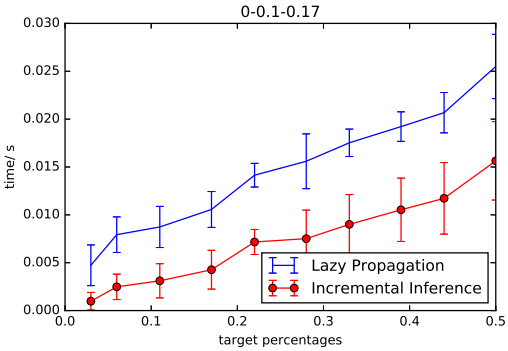

0-0.1-0.33
time/ s
target percentages
Lazy Propagation
Incremental Inference

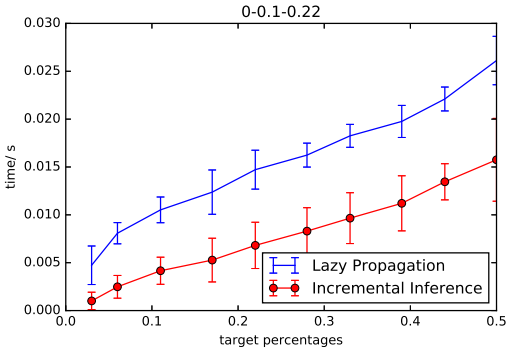

0-0.1-0.39
time/ s
target percentages
Lazy Propagation
Incremental Inference

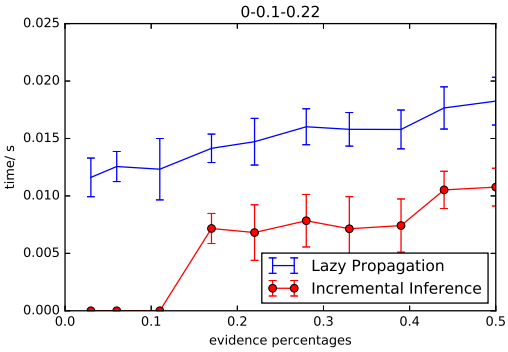

0-0.1-0.39
time/ s
evidence percentages
Lazy Propagation
Incremental Inference


0-0.1-0.28
time/ s
target percentages
Lazy Propagation
Incremental Inference


0-0.1-0.33
time/ s
evidence percentages
Lazy Propagation
Incremental Inference


0-0.1-0.17
time/ s
target percentages
Lazy Propagation
Incremental Inference


0-0.1-0.22
time/ s
target percentages
Lazy Propagation
Incremental Inference


0-0.1-0.22
time/ s
evidence percentages
Lazy Propagation
Incremental Inference

Bibliography

[1] Agli, H., Bonnard, P., Gonzales, C., Wuillemin, P.H.: Uncertain Reasoning for Business Rules. In: CEUR Proceedings of the 8th International Web Rule Symposium Rule Doctoral Consortium. RuleML (2014)

[2] Agli, H., Bonnard, P., Gonzales, C., Wuillemin, P.H.: Business Rules Uncertainty Management with Probabilistic Relational Models. In: Proceedings of the 10th International Web Rule Symposium. pp. 53–67. RuleML (2016)

[3] Agli, H., Bonnard, P., Gonzales, C., Wuillemin, P.H.: Incremental junction tree inference. In: Proceedings of the 16th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems. pp. 326–337. IPMU (2016)

[4] Agli, H., Bonnard, P., Gonzales, C., Wuillemin, P.H.: Un algorithme d'arbre de jonction incrémental. In: Proceedings of 8èmes Journées Francophones des Réseaux Bayésiens, JFRB (2016)

[5] Agli, H., Bonnard, P., Gonzales, C., Wuillemin, P.H.: Modèles Probabilistes Relationnels et leur inférence incrémentale pour les systèmes à base de règles orientés objet. Revue d'Intelligence Artificielle (2017), submitted

[6] Agli, H., Bonnard, P., Wuillemin, P.H., Perez, K.: Des règles métier pour la gestion de l'incertain. In: Proceedings of 7èmes Journées Francophones des Réseaux Bayésiens, JFRB (2014)

[7] Aït-Kaci, H., Bonnard, P.: Probabilistic production rules. Tech. rep., International Business Machine (2011)

[8] Allen, D., Darwiche, A.: New Advances in Inference by Recursive Condi-

tioning. In: Proceedings of the 9th Conference on Uncertainty in Artificial Intelligence. UAI (2003)

[9] Appleton, D.S.: Business rules: The missing link. Datamation, 30 pp. 145–150 (1984)

[10] Arru, M.: Introduction of probabilistic reasoning in JRules. Master's thesis, Polytech Nantes (2011)

[11] Bahar, R., Frohm, E., Gaona, C., Hachtel, G., Macii, E., Pardo, A., Somenzi, F.: Algebric decision diagrams and their applications. Formal Methods in System Design 10, 171–206 (1997)

[12] Bangsø, O., Wuillemin, P.H.: Object oriented Bayesian networks: A framework for topdown specification of large Bayesian networks and repetitive structures. Tech. rep., Aalborg University, Denmark (2000)

[13] Barker, V.E., O'Connor, D.E., Bachant, J., Soloway, E.: Expert systems for configuration at digital: Xcon and beyond. Communications ACM 32, 298–318 (1989)

[14] Barzilay, R., McCullough, D., Rambow, O., DeCristofaro, J., Korelsky, T., Lavoie, B.: A new approach to expert system explanations. In: Proceedings of the 9th International Workshop on Natural Language Generation. pp. 78–87 (1998)

[15] BayesFusion, LLC: SMILE : Structural Modeling, Inference, and Learning Engine, https://www.bayesfusion.com/smile-engine

[16] Bobek, S., Nalepa, G.J.: Compact representation of conditional probability for rule-based mobile context-aware systems. In: Proceedings of 9th International Symposium. pp. 83–96. RuleML (2015)

[17] Bodlaender, H.L.: A tourist guide through treewidth. Acta Cybernetica 11, 1–23 (1993)

[18] BOSCH: Visual Rules, https://www.bosch-si.com/bpm-and-brm/visual-rules/business-rules-management.html

[19] Boyer, J., Mili, H.: Agile Business Rule Development - Process, Architecture, and JRules Examples. Springer (2011)

[20] Brownston, L., Farrell, R., Kant, E., Martin, N.: Programming Expert Systems in OPS5: An Introduction to Rule-based Programming. Addison-Wesley Longman Publishing Co., Inc. (1985)

[21] Buchanan, B., Feigenbaum, E., Lederberg, J.: Heuristic DENDRAL: a program for generating explanatory hypotheses in organic chemistry. In: Proceedings of the 1st Hawaii International Conference on System Sciences . pp. 482–485. HICSS (1968)

[22] Buchanan, B.G., Shortliffe, E.H.: Rule Based Expert Systems: The Mycin Experiments of the Stanford Heuristic Programming Project. Addison-Wesley (1984)

[23] Butz, C.J., dos Santos, A.E., Oliveira, J.S., Gonzales, C.: A simple method for testing independencies in bayesian networks. In: Proceedings of the 29th Canadian Conference on Artificial Intelligence. pp. 213–223 (2016)

[24] Cooper, G.F.: The computational complexity of probabilistic inference using Bayesian belief networks. Artificial Intelligence 42, 393–405 (1990)

[25] Cowell, R.G., Dawid, A.P., Lauritzen, S.L., Spiegelhalter, D.J.: Probabilistic Networks and Expert Systems: Exact Computational Methods for Bayesian Networks. Springer (2007)

[26] Dagum, P., Luby, M.: Approximating probabilistic inference in Bayesian belief networks is NP-hard. Artificial Intelligence 60, 141–153 (1993)

[27] D'Ambrosio, B.: Incremental probabilistic inference. In: Proceedings of the 9th International Conference on Uncertainty in Artificial Intelligence. pp. 301–308. UAI (1993)

[28] Darlington, K.: Effectiveness of explanation facilities for intelligent systems. Ph.D. thesis, London South Bank University, UK (2014)

[29] Darwiche, A.: Dynamic Join Trees. In: Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence. pp. 97–104. UAI (1998)

[30] Darwiche, A.: A Logical Approach to Factoring Belief Networks. In: Proceedings of the 8th International Conference on Principles and Knowledge Representation and Reasoning. pp. 409–420. KR (2002)

[31] Darwiche, A.: Modeling and Reasoning with Bayesian Networks. Cambridge University Press (2009)

[32] Darwiche, A.: Inference in Bayesian networks: A historical perspective. In: R. Decther, H.G., Halpern, J. (eds.) Heuristics, Probability, and Causality: a Tribute to Judea Pearl, pp. 105–120. College Publications (2010)

[33] Date, C.: What Not How: The Business Rules Approach to Application Development. Addison-Wesley (2000)

[34] Dawid, A.P.: Conditional independence in statistical theory. Journal of the Royal Statistical Society. Series B (Methodological) 41, 1–31 (1979)

[35] De Raedt, L., Kimmig, A.: Probabilistic (logic) programming concepts. Machine Learning 100, 5–47 (2015)

[36] Dechter, R.: Bucket Elimination: A Unifying Framework for Probabilistic Inference. In: Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence. pp. 211–219. UAI (1996)

[37] Diaz, O., Iturrioz, J., Piattini, M.G.: Promoting business policies in object-oriented methods. Journal of Systems and Software. 41, 105–115 (1998)

[38] Duda, R., Shortliffe, E.: Expert systems research. Science pp. 261–268 (1983)

[39] Durkin, J.: Expert Systems: a view of the field. IEEE Expert 11, 56–63 (1996)

[40] Elkan, C.: The paradoxical success of fuzzy logic. In: Proceedings of the 11th National Conference on Artificial Intelligence. pp. 698–703 (1993)

[41] Erl, T.: SOA Design Patterns. Prentice Hall (2008)

[42] Feigenbaum, E., McCorduck, P.: The fifth generation: Artificial Intelligence and Japan's computer challenge to the world. Addison-Wesley (1983)

[43] FICO: FICO Blaze Advisor Decision Rules Management System, http://www.fico.com/en/products/fico-blaze-advisor-decision-rules-management-system#overview

[44] Flores, M.J., Gámez, J.A., Olesen, K.G.: Incremental Compilation of Bayesian Networks. In: Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence. pp. 233–240. UAI (2003)

[45] Forgy, C.: Ops5 user's manual. Tech. rep., CMU-CS-81-135, Carnegie-Mellon University, Pittsburgh (1981)

[46] Forgy, C.: Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. Artificial Intelligence 19, 17–37 (1982)

[47] Geiger, D.: Graphoids: A Qualitative Framework for Probabilistic Inference. Ph.D. thesis, University of California at Berkeley (1990)

[48] Getoor, L., Taskar, B.: Introduction to Statistical Relational Learning, Adaptive Computation and Machine Learning. MIT Press (2007)

[49] Gonzales, C., Torti, L., Wuillemin, P.H.: aGrUM: a Graphical Universal Model framework. In: Proceedings of the 30th International Conference on Industrial Engineering, Other Applications of Applied Intelligent Systems (2017)

[50] Graham, I.: Business Rules Management and Service Oriented Architecture: A Pattern Language. Wiley (2007)

[51] Graham, I.: Service Oriented Business Rules Management Systems. Tech. rep., TriReme (2005)

[52] Hall, C., Harmon, P.: The 2006 BPTrends Report on Business Rules Products. Tech. rep., Business Process Trends (2006)

[53] Halle, B.V.: Business Rules Applied: Building Better Systems Using the Business Rules Approach. John Wiley & Sons (2001)

[54] Halle, B.V., Goldberg, L.: Business Rule Revolution: Running Business the Right Way. Happy About (2006)

[55] Halpern, J.Y.: Reasoning about uncertainty. MIT Press (2003)

[56] Hanson, E., Hasan, M.S.: Gator: An optimized discrimination network for active database rule condition testing. Tech. rep., TR93-036, University of Florida (1993)

[57] Hart, P.E., Duda, R.O., Einaudi, M.T.: PROSPECTOR—a computer-based consultation system for mineral exploration. Journal of the International Association for Mathematical Geology 10, 589–610 (1977)

[58] Hay, D.C., Healy, K.A.: Defining business rules what are they really? Tech. rep., Business Rules Group (July 2000)

[59] Heckerman, D., Breese, J.S.: Causal independence for probability assessment and inference using Bayesian networks. IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans 26, 826–831 (1996)

[60] Heckerman, D., Meek, C., Koller, D.: Probabilistic Entity-Relationship Models, PRMs and Plate Models. In: Getoor, L., Taskar, B. (eds.) Introduction to Statistical Relational Learning, pp. 201–239. MIT Press (2004)

[61] Heckerman, D.E., Shortliffe, E.H.: From certainty factors to belief networks. Artificial Intelligence in Medicine 4, 35–52 (1992)

[62] Hilwa, A.: Worldwide business rules management systems forecast, 2015-2019: Gazing into the cloud. Tech. rep., International Data Corporation (2015)

[63] IBM: Operational Decision Manager, https://www.ibm.com/support/knowledgecenter/SSQP76_8.9.0/welcome/kc_welcome_odmV.html

[64] Infoholic: Global Business Rules Management System (BRMS) Market, Trends & Forecast: 2015-2020. Tech. rep., Infoholic Research (2015)

[65] InRule Technology, Inc.: Inrules, http://www.inrule.com/products/

[66] Jaeger, M.: Relational Bayesian Networks. In: Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence. pp. 266–273. UAI (1997)

[67] Jordan, M.I.: Graphical models. Statistical Science 19, 140–155 (2004)

[68] Kersting, K., De Raedt, L., Kramer, S.: Interpreting Bayesian logic programs. In: Working Notes of the AAAI-2000 workshop on learning statistical models from relational data. pp. 29–35 (2000)

[69] Kim, J.H., Pearl, J.: A Computational Model for Causal and Diagnostic Reasoning in Inference Systems. In: Proceedings of the 8th International Joint Conference on Artificial Intelligence. IJCAI (1983)

[70] Kjærulff, U.: Triangulation of graphs – algorithms giving small total state space. Tech. rep., University of Aalborg, Institute for Electronic Systems, Department of Mathematics and Computer Science (1990)

[71] Knolmayer, G., Herbst, H., Schlesinger, M.: Enforcing Business Rules by the Application of Trigger Concepts. In: Proceedings of the Priority Programme Informatics Research, Information Conference Module 1. pp. 24–30 (1994)

[72] Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques. MIT Press (2009)

[73] Koller, D., Pfeffer, A.: Object-Oriented Bayesian Networks. In: Proceedings of the 13th Annual Conference on Uncertainty in Artificial Intelligence. pp. 302–313. UAI (1997)

[74] Korver, M., Lucas, P.J.F.: Converting a rule-based expert system into a belief network. Medical Informatics 18, 219–241 (1993)

[75] Laird, J.E., Newell, A., Rosenbloom, P.S.: SOAR: An architecture for general intelligence. Artificial Intelligence 33, 1 – 64 (1987)

[76] Laskey, K.B.: MEBN: A language for first-order Bayesian knowledge bases. Artificial Intelligence 172, 140 – 178 (2008)

[77] Lauritzen, S., Dawid, A.P., Larsen, B.N., Leimer, H.G.: Independence properties of directed Markov fields. Networks 20, 491–505 (1990)

[78] Lauritzen, S., Spiegelhalter, D.J.: Local computations with probabilities on graphical structures and their applications to expert systems. Journal of the Royal Statistical Society 50, 157–224 (1988)

[79] Leith, P.: The rise and fall of the legal expert system. European Journal of Law and Technology 1, 1–16 (2010)

[80] Li, W., van Beek, P., Poupart, P.: Performing Incremental Bayesian Inference by Dynamic Model Counting. In: Proceedings of the National Conference on Artificial Intelligence. pp. 1173–1179 (2006)

[81] Lin, Y., Druzdzel, M.J.: Relevance-Based Sequential Evidence Processing in Bayesian Networks. In: Proceedings of the 11th International Florida Artificial Intelligence Research Society Conference. pp. 446–450. FLAIRS (1998)

[82] Lucas, P., van der Gaag, L.: Principles of Expert Systems. Addison-Wesley Longman Publishing Co., Inc. (1991)

[83] Madsen, A.L., Jensen, F.V.: Lazy propagation: A junction tree inference algorithm based on lazy evaluation. Artificial Intelligence 113, 203–245 (1999)

[84] Mahoney, S.M., Laskey, K.B.: Network engineering for complex belief networks. In: Proceedings of the 12th International Conference on Uncertainty in Artificial Intelligence. pp. 389–396. UAI (1996)

[85] Melle, W.: A domain-independent production-rule system for consultation programs. In: Proceedings of the 6th International Joint Conference on Artificial Intelligence. pp. 923–925. IJCAI (1979)

[86] Microsoft: BizTalk Server, https://www.microsoft.com/france/serveur-cloud/biztalk/default.aspx

[87] Miranker, D.P.: TREAT: A New and Efficient Match Algorithm for AI Production Systems. Ph.D. thesis, Columbia University (1987)

[88] Moore, J., Swartout, W.: Explanation in expert systems: a survey. University of Southern California, Information Sciences Institute (1988)

[89] Moore, J.D.: Participating in Explanatory Dialogues: Interpreting and Responding to Questions in Context. MIT Press (1994)

[90] Morgan, T.: Business Rules and Information Systems: Aligning IT with Business Goals. Addison-Wesley Professional (2002)

[91] Newell, A.: Human Problem Solving. Prentice-Hall, Inc. (1972)

[92] Ng, K.C., Abramson, B.: Uncertainty management in expert systems. IEEE Expert: Intelligent Systems and Their Applications 5, 29–48 (1990)

[93] O.M.G Specifications: Semantics of business vocabulary and business rules. Tech. rep., Object Management Group (2015)

[94] Onisko, A., Lucas, P.J.F., Druzdzel, M.J.: Comparison of rule-based and Bayesian network approaches in medical diagnostic systems. In: Proceedings of the 8th Conference on Artificial Intelligence in Medicine in Europe. LNCS, vol. 2101, pp. 283–292 (2001)

[95] OpenRules, Inc.: Open Source Business Decision Management System, http://openrules.com/

[96] Oracle Business Rules: http://www.oracle.com/technetwork/middleware/business-rules/overview/index.html

[97] Pearl, J., Paz, A.: Graphoids: Graph-Based Logic for Reasoning about Relevance Relations or When would x tell you more about y if you already know z? In: Proceedings of the 7th European Conference on Artificial Intelligence. pp. 357–363. ECAI (1986)

[98] Pearl, J.: Reverend Bayes on Inference Engines: A Distributed Hierarchical Approach. In: Proceedings of the Second AAAI Conference on Artificial Intelligence. pp. 133–136. AAAI (1982)

[99] Pearl, J.: A Constraint-Propagation Approach to Probabilistic Reasoning. In: Proceedings of Workshop on Uncertainty and Probability in Artificial Intelligence. pp. 31–42 (1985)

[100] Pearl, J.: Fusion, propagation, and structuring in belief networks. Artificial Intelligence 29, 241–288 (1986)

[101] Pearl, J.: Evidential reasoning using stochastic simulation of causal models. Artificial Intelligence 32, 245–257 (1987)

[102] Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann (1988)

[103] Pearl, J.: Belief networks revisited. Artificial Intelligence 59, 49 – 56 (1993)

[104] Pegasystems Business Rules Platform : https://www.pega.com/business-rules-platform

[105] Perez, K.: Probabilistic Production Rules. Master's thesis, École des Mines de Saint-Etienne (2013)

[106] Pfeffer, A.J.: Probabilistic Reasoning for Complex Systems. Ph.D. thesis, Stanford University (2000)

[107] Poo, D.C.C.: Implementing an evolutionary structural software model. Journal of Systems and Software 22, 81–90 (1993)

[108] Poo, D.C.C., Lee, S.: Tartan: Interweaving objects with rules in information systems development. Journal of Systems and Software 33, 3–14 (1996)

[109] Progress Software Corporation: Progress Corticon, https://www.progress.com/corticon

[110] Red Hat: JBoss BRMS, https://www.redhat.com/en/technologies/jboss-middleware/business-rules

[111] Robert, C., Casella, G.: Monte Carlo Statistical Methods. Springer-Verlag New York (2004)

[112] Ross, R.G.: Business Rules Manifesto. Tech. rep., The Business Rules Group (2003)

[113] Ross, R.G.: Principles of the Business Rule Approach. Addison-Wesley (2003)

[114] Ross, R.G., Lam, G.S.: Building Business Solutions: Business Analysis with Business Rules. Business Rule Solutions (2011)

[115] Russell, S., Norvig, P.: Artificial Intelligence, a modern approach. Prentice Hall (2003)

[116] Sang, T., Bearne, P., Kautz, H.: Performing Bayesian Inference by Weighted Model Counting. In: Proceedings of the 20th National Conference on Artificial Intelligence. pp. 475–481. AAAI (2005)

[117] SAP: NetWeaver Business Rules Management, https://archive.sap.com/documents/docs/DOC-26748

[118] Shachter, R.D., Peot, M.A.: Simulation approaches to general probabilistic inference on belief networks. In: Proceedings of the 5th Annual Conference on Uncertainty in Artificial Intelligence. pp. 221–234. UAI (1990)

[119] Shafer, G., Pearl, J.: Readings in Uncertain Reasoning. Morgan Kaufman (1990)

[120] Shenoy, P., Shafer, G.: Axioms for Probability and Belief-Function Propagation. In: Yager, R.R., Liu, L. (eds.) Classic Works of the Dempster-Shafer Theory of Belief Functions, pp. 499–528. Springer Berlin Heidelberg (2008)

[121] Smets, P.: Imperfect information: Imprecision and uncertainty. In: Motro, A., Smets, P. (eds.) Uncertainty Management in Information Systems: From Needs to Solutions, pp. 225–254. Springer US (1997)

[122] Sottara, D.: Integration of symbolic and connectionist AI techniques in the development of Decision Support Systems applied to biochemical processes. Ph.D. thesis, University of Bologna (2010)

[123] Standish: The CHAOS report. Tech. rep., The Standish Group International Inc (1995)

[124] Standish: The CHAOS report. Tech. rep., The Standish Group International Inc (2004)

[125] Swartout, W., Moore, J.: Second generation expert systems. In: David, J.M., Krivine, J.P., Simmons, R. (eds.) Explanation in Second Generation Expert Systems, pp. 543–585. Springer-Verlag New York, Inc. (1993)

[126] Swartout, W., Paris, C., Moore, J.: Explanations in knowledge systems: design for explainable expert systems. IEEE Expert 6, 58–64 (1991)

[127] Torti, L.: Inférence probabiliste structurée dans les modèles graphiques probabilistes orientés-objet. Ph.D. thesis, Pierre et Marie Curie University (2012)

[128] Tsalgatidou, A., Loucopoulos, P.: An object-oriented rule-based approach to the dynamic modelling of information systems. In: Proceedings of the International Working Conference on Dynamic Modelling of Information Systems . pp. 165–188 (1991)

[129] Verma, T., Pearl, J.: Causal Networks: Semantics and Expressiveness. In: Proceedings of the 4th Annual Conference on Uncertainty in Artificial Intelligence. pp. 69–78. UAI (1988)

[130] Vlasselaer, J., Van den Broeck, G., Kimmig, A., Meert, W., Raedt, L.D.: Tp-compilation for inference in probabilistic logic programs. International Journal of Approximate Reasoning 78, 15 – 32 (2016)

[131] Weske, M.: Business Process Management. Springer-Verlag Berlin Heidelberg (2012)

[132] Wick, M.R., Slagle, J.R.: An explanation facility for today's expert systems. IEEE Expert 4, 26–36 (1989)

[133] Wick, M.R.: Second generation expert system explanation. In: David, J.M., Krivine, J.P., Simmons, R. (eds.) Second Generation Expert Systems, pp. 614–640. Springer Berlin Heidelberg (1993)

[134] Wiegerinck, W., Kappen, B., Burgers, W.: Bayesian networks for expert systems: Theory and practical applications. In: Babuška, R., Groen, F.C.A. (eds.) Interactive Collaborative Information Systems, pp. 547–578. Springer Berlin Heidelberg (2010)

[135] Ye, L.R., Johnson, P.E.: The impact of explanation facilities on user acceptance of expert systems advice. MIS Quarterly 19, 157–172 (1995)

[136] Zadeh, L.A.: Fuzzy sets. Information and Control 8, 338–353 (1965)

[137] Zhang, N.L., Poole, D.: A simple approach to Bayesian network computations. In: Proceedings of the 10th Biennial Canadian Conference on Artificial Intelligence. pp. 171–178 (1994)

Notations and Acronyms

$\mathbb{P}$	a probability distribution
DAG	Directed acyclic graph
$\mathcal{G} = (\mathcal{V}, \mathcal{A})$	a directed graph over nodes (random variables) $\mathcal{V}$ and arcs $\mathcal{A}$
CPT	Conditional Probability Table

CNF	Conjunctive Normal Form
PGM	Probabilistic Graphical Model
BN	Bayesian Network
BR	Business Rule
RBS	Rules-based System
BRMS	Business Rules Management Systems
KB	Knowledge Base
WM	Working Memory
WME	Working Memory Element
IT	Information Technology
POJO	Plain Old Java Object
IDE	Integrated Development Environment
OM	Object Model
AI	Artificial Intelligence

Subject Index