IMPROVING DECISION MODELING: ENHANCING MULTI-ENTITY DECISION GRAPH MODELING CAPABILITY AND SUPPORTING KNOWLEDGE REUSE

by

Mark A. Locher

A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Systems Engineering and Operations Research

Committee:

_____	Dr. Paulo C. G. Costa, Dissertation Director
_____	Dr. Kathryn B. Laskey, Committee Member
_____	Dr. Andrew Loerch, Committee Member
_____	Dr. Duminda Wijesekera, Committee Member
_____	Dr. John Shortle, Department Chair
_____	Dr. Kenneth S. Ball, Dean, Volgenau School of Engineering

Date:_____

Fall Semester 2019
George Mason University
Fairfax, VA

Improving Decision Modeling: Enhancing Multi-Entity Decision Graph Modeling Capability and Supporting Knowledge Reuse

A Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

by

Mark A. Locher
Master of Science
George Mason University, 2012
Master of Business Administration
Saint Mary's University, 1981
Bachelor of Science
United States Air Force Academy, 1979

Director: Paulo C. G. Costa, Associate Professor
Department of Systems Engineering and Operations Research

Fall Semester 2019
George Mason University
Fairfax, VA

DEDICATION

This is dedicated to my dear wife Peggy, who agreed in 2005 that my going back to school to update my skills would be a good idea, and then wondered if it would ever end.

ACKNOWLEDGEMENTS

I would like to thank the members of the faculty of the Department of System Engineering and Operations Research at George Mason University for guiding me during my time at George Mason. I am indebted to the late Dr. David Schum, who inspired me on my ultimate choice for a research area. I began my studies at George Mason intending to focus on system architecture. My second course at George Mason was a decision analysis course with Dr. Schum. His focus was on the evidential basis for reasoning under uncertainty, identifying several different approaches and methodologies. I was smitten and ended up taking four additional courses with him. I am deeply indebted to my dissertation director, Dr. Paulo Costa. He was free with his time, patient, and inspirational. I left every meeting with him uplifted and upbeat. He was all I could ask for in a director. I thank Dr. Kathryn Lasky for the grounding that she gave me in Bayesian approaches and reasoning, both in class and in separate discussions. It aided me immensely in my research efforts. I thank the other two members of my committee, Dr. Duminda Wijesekera and Dr. Andrew Loerch, for their time, advice and encouragement in completing this work. I appreciate the time spent in fruitful discussions with my classmate, Dr. Shou Matsumoto, who developed the initial implementation of Multi-Entity Decision Graphs, which this work expands upon. Experimenting with the early versions allowed me the opportunity to better understand how it works and to see where enhancements would fit in. My interactions with Dr. Andy Powell were a delight and I thank him for them. The attendees at the weekly Krypton seminar made the process enlightening and enjoyable.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

LIST OF ABBREVIATIONS

A-Type ..... Attribute Type
BN ..... Bayesian Network
CE ..... Collectively Exhaustive
CPT ..... Conditional Probability Table
CRN ..... Context Recognition Network
CRT ..... Compatibility Restriction Table
CSI ..... Context Specific Independence
CV ..... Context Variable
DG ..... Decision Graphs
DSS ..... Decision Support System
D ..... Decision Element
ECM ..... Electronic Countermeasure
F ..... Fusion Element
F ..... Functional relationship
F-Type ..... Functional Relationship Type
ICI ..... Independence of Causal Influences
IFF ..... Identify Friend or Foe
IR ..... Infrared
KR ..... Knowledge Representation
LPD ..... Local Probability Distribution
MDO ..... MEDG Decision Ontology
MEBN ..... Multi-Entity Bayesian Networks
MECE ..... Mutually Exclusive and Collectively Exhaustive
MEDG ..... Multi-Entity Decision Graph
MFrags ..... MEBN Fragments
MWR ..... Missile Warning Receiver
OOBN ..... Object Oriented Bayesian Networks
OPRML ..... Object-Oriented Probabilistic Relational Modeling Language
OV ..... Ordinary Variable
OWL ..... Web Ontology Language
PAS ..... Prototypical Analytic Structures
PC ..... Parent Configuration
PRM ..... Probabilistic Relational Models
PR-OWL ..... Probabilistic Web Ontology Language
PSMs. ..... Problem Structuring Methods
QPN ..... Qualitative Probabilistic Networks

R ..... Relationship
R-Type ..... Relationship Type
RDF..... Resource Description Framework
RV ..... Random Variable
RWR ..... Radar Warning Receiver
SA ..... Situational Awareness
SAM..... Surface to Air Missile
SCA..... Strategic Choice Approach
Soft OR ..... Soft Operations Research
SSBN..... Situation Specific Bayesian Network
SSDG ..... Situation Specific Decision Graph
SWOT ..... Strength / Weakness / Opportunity / Threat

ABSTRACT

IMPROVING DECISION MODELING: ENHANCING MULTI-ENTITY DECISION GRAPH MODELING CAPABILITY AND SUPPORTING KNOWLEDGE REUSE

Mark A. Locher, Ph.D.

George Mason University, 2019

Dissertation Director: Dr. Paulo C. G. Costa

Formal decision-making quality is affected by the capabilities of the decision- making tools and the availability of reusable decision knowledge. Decision graphs are a powerful decision modeling framework. Many decision graph elements are probabilistic, and each requires a Local Probability Distribution (LPD) specifying the probabilities of each state in that element. State-of-the-art implementations have first-order expressivity, where the computer can automatically create decision graphs with a varying number of entities (e.g. people, decisions, alternatives, etc.). Varying the entities creates different model configurations. For every possible model configuration, the modeling tool must define the LPD, using information provided by the modeler. Existing first-order expressive decision graph implementations have a critical limitation arising from a lack of research on the full range of capabilities required by a modeling tool to create these LPD. This lack restricts modeling of important types of decision problems. Compounding this limitation is a lack of de-

cision knowledge reuse capabilities, reducing decision modelers' ability to learn from other modelers' experiences on similar decision problems, increasing modeling time and the possibility of an incomplete model.

The research focused on eliminating modeling tool limitations and creating a knowledge reuse capability. First, it developed a new approach, dependency modeling, that identified a set of LPD behaviors that provide a robust capability to any first-order expressive modeling tool and includes ten new LPD development language capabilities that significantly extend the range of problems that can be modeled. Second, dependency modeling uncovered eight design patterns useful for modeling in any probabilistic first-order expressive framework. Third, it refined and integrated four factors that differentiate decision problems, identifying information reuse requirements and uncovering additional modeling tool needs. Fourth, it addressed the additional needs uncovered, developing algorithms to address decision problem asymmetry, developing a more efficient approach to model context variation, and defining modeling approaches for decision problems with varying entity counts. Fifth, it created decision templates as a decision knowledge reuse tool. An initial evaluation provided preliminary data that the template could reduce model development time by 50%, with 75% of participants assessing the template as useful or very useful in completing the experimental task.

CHAPTER 1
INTRODUCTION

1.1 Problem Introduction

This dissertation presents research results designed to improve decision-making support tools. The research focused on expanding modeling capabilities to address a wider range of significant decision problems and on developing knowledge reuse capabilities across problem sets to speed and enhance decision problem structuring and modeling.

Decision making is an important human activity, with consequences that ripple through the lives of the decision-maker, people affected by the decision, and the greater environment within which they live. A decision problem is a felt need by a decision-maker that there is a disconnect between the current situation and a desired future. This disconnect may be due to some obstacle to overcome or an opportunity to achieve a desirable outcome (Grünig and Kühn 2013). The primary formal approach to decision making is called decision analysis. It combines systems analysis, decision theory, epistemic probability, and cognitive psychology into a comprehensive approach to decision making. Decision analysis uses a decision problem structuring process that transforms a vaguely defined problem into a clearly articulated decision problem with achievable alternatives. A key product of the structuring process is a decision model, often graphical, that presents all available relevant information in an organized and understandable form that aids

the decision-maker. If the model is executable, the decision-maker can explore the ramifications of changes to model elements (R. A. Howard 1966).

The power of a decision model is affected by the capabilities of the tools used to create that model. The decision graph (DG) is a versatile graphical modeling tool for decision-making, able to visualize and manipulate the same information as several other graphical decision support tools, such as decision trees, objective hierarchies / value trees, event / fault trees, and ends-means diagrams. It is a derivative of the influence diagram (Ronald A. Howard and Matheson 1984) that extends a Bayesian Network (BN) (Pearl 1988) into the decision-making domain (Cooper 1988). Bayesian networks are a well-developed technology for modeling problems with probabilistic uncertainty (Finn Jensen 1996). A Bayesian network is a set of nodes representing random variables which model elements of a domain of interest. The directed arcs between nodes represent dependencies between nodes. The node from which the arc originates is called a parent of the node where the arc terminates. Decision graphs (DG) extend BNs by adding nodes that represent decision and utility elements. These allow a decision graph to compactly model the elements of a decision problem and visualize the dependencies between them (F. V. Jensen and Nielsen 2007). An important BN / DG feature is that every RV has a local probability distribution (LPD) that specifies the probability of each of its states given the states of its parent RVs.

Both BNs and decision graphs have a significant limitation – they can only model problems with a static structure. If the number of entities modeled by the problem changes, the model needs to be rebuilt. A modeling tool is first-order expressive if it can handle

varying entity counts. A first-order expressive modeling capability is sufficient to model anything executable by a computer (Sowa 2000). Multi-Entity Bayesian Networks (MEBN) are a first-order expressive extension of BNs. It provides a formal procedure to restructure a BN when information about new entities needs to be incorporated (Laskey 2008). Just as BNs were extended to decision graphs, MEBN has been extended to a Multi-Entity Decision Graph (MEDG) capability (Matsumoto 2019). Unfortunately, its initial implementation – UnBBayes-MEDG - falls short in providing decision-makers with a decision support tool that brings the advantages of first-order expressiveness. For instance, it has a limited ability to use random variables whose states are entities rather than fixed categorical values or Boolean. This limitation restricts the use of decisions where the alternatives are a varying list of entities rather than a fixed list. This limitation arises from a research gap in understanding the range of actions that a computer needs to take to develop each probabilistic node's LPD, given a possibly varying number of parent nodes. Additional limitations identified in this research are an inability to automatically handle certain types of decision model structures and inefficiencies in addressing decision context. These limitations severely hinder the use of the MEDG implementation in real-world decision problems. This dissertation's first research objective was to push the current state-of-the art of decision graph modeling capabilities so that decision graph tools can be applied as a first-order expressive decision tool across a broader range of decision problems. I identified the range of behaviors an LPD creation capability needed to create LPDs for any first-order expressive decision graph modeling framework. I also explored the effects that different decision problem characteristics had on the range of

modeling capabilities that any decision graph modeling tool needed to model decision problems with those varying characteristics.

The second research objective was to improve decision knowledge reuse in decision structuring, one of the most important processes in decision making (von Winterfeldt 1980). This process defines the decision model and develops all available relevant decision information. But structuring is considered an art form, with limited tool support to assist the process (von Winterfeldt and Edwards 2007). Mintzberg et al. identified the fundamental approach to decision problem structuring: decompose the problem to a level where the decision-maker either has knowledge about the elements of the decision problem or can identify what information must be developed to solve the decision problem. Then, the elements are recomposed into a decision model (Mintzberg, Raisinghani, and Théorêt 1976). This makes knowledge reuse central to this process.

Decision problems can be categorized by common characteristics. Individual problems may differ in detail and vary in model structure specifics. But knowing the general model structure and the types of information used for a class of decision problems speeds model development and focuses structuring resources to improve decision quality. In addition, different problem classes may have similarities that allow knowledge reuse at a higher level of generality. For instance, many decision problem classes use a form of sequential down selection, where selection criteria and objectives are successively refined, which narrows and refines the decision alternatives under consideration. Tools to support knowledge reuse need to address the varying level of granularity in the applicable knowledge being transferred from one decision problem to another.

Both MEBN and MEDG have two features that support information reuse. One is their use of a knowledge fragment-based approach. These knowledge fragments, called MEBN / MEDG Fragments (MFrags), represent probabilistic relationships among a conceptually meaningful group of uncertain attributes and relationships between entities (Laskey 2008). The second feature is the use of Probabilistic Web Ontology Language (PR-OWL) (Paulo Cesar G. Costa 2005; Rommel N. Carvalho, Laskey, and Costa 2017), an upper ontology for defining probabilistic ontologies expressed in the MEBN language. PR-OWL extends the Web Ontology Language (OWL) to allow probabilistic knowledge. OWL is a widely used standard for creating, sharing and reusing ontologies of various domains (Pascal Hitzler et al. 2012). PR-OWL can both create OWL-based ontologies and reuse existing OWL-based ontologies created by others, enhancing knowledge reuse capabilities.

MFrags can be shared among different problems that have common characteristics. They can also be mined for patterns and relationships for use in similar problems where the fragment itself is not an exact fit. In those cases, it is useful to access the key development information used to develop them. This provides important background information to understand how they can be adapted. A vehicle to capture that information would provide decision-makers useful material in solving new decision problems.

Von Winterfeldt suggested organizing knowledge about specific problem classes using Prototypical Analytic Structures (PAS) (von Winterfeldt 1980). Analytic structures are a set of graphical models (e.g., objective hierarchy / value tree, decision tree, ends-means diagram, influence diagram, event / fault tree, belief nets.) and tables (e.g., conse-

quence tables) that capture decision making information for a specific problem for use by the decision-maker. For each problem class, a set of PAS, filled with information relevant to that class, could be developed. While attractive, the PAS concept was never fleshed out. Decision graphs can represent most of these graphical forms. Having a knowledge source that identifies the information for different problem classes would be valuable. This dissertation's second research objective resulted in developing a decision structuring knowledge reuse tool.

1.2 Research Problem and Solution Approach

The decision graph is a versatile tool for modeling decision problems, and the Multi-Entity Decision Graph (MEDG) modeling framework provides a versatile capability to model decision problems where the number of entities modeled can vary between instances when a decision solution is needed. But the shortcomings in the initial MEDG implementation identify research shortfalls in understanding the capabilities a first-order expressive decision graph tool needs to address important problems. In addition, MEDG brings some valuable knowledge reuse capabilities. But there is a need for a more comprehensive decision-making focused toolset for improving knowledge reuse among classes of decision problems. The literature has proposed concepts for enhancing this, but, to date, none are known to have been implemented. In other words, a gap exists between the current state-of-the-art and the knowledge tools needed to support advanced decision modeling with consistent and coherent knowledge reuse.

To address this knowledge gap, four research efforts were pursued:

- Explore wide range of decision problems to identify differentiating characteristics. Identify impacts on a decision graph modeling tool's capabilities and decision structuring knowledge reuse requirements
- Conduct a comprehensive assessment of the LPD creation process to identify the capabilities needed in any first-order expressive probabilistic modeling capabilities. Develop specific enhancements for the MEDG implementation
- Use results from above two efforts to develop needed capabilities for decision graph modeling tools
- Use results from the first effort to develop a knowledge reuse-based decision structuring support tool.

These four research efforts resulted in five contributions to the state-of-the-art for decision-making support.

1.3 Dissertation Structure and Research Contributions

The structure of this dissertation follows the four research efforts and their contributions. First, Chapter 2 provides the necessary background information to understand the contributions. Then, Chapter 3 explores the characteristics that differentiates decision problems. Understanding these characteristics sets the stage for Chapters 5 and 6. This exploration makes two contributions to the state-of-the-art. *The first contribution is a new characterization of decision problem differentiators*. Four important differentiators were found: context, problem type, decision pattern, and uncertainty.

Understanding a decision's context is acknowledged to be important to structuring a problem but is lightly discussed in the decision analysis literature. Several useful concepts were found in the larger artificial intelligence literature that was adapted to refine the concept of decision context. The key idea is that context has both internal decision-maker and external situation factors that interact to define the context. The context differentiator identifies four major ways in which the decision context affects the characterization of a specific decision problem. This provides a basis for modeling context and identifies decision context information that should be in in a decision knowledge reuse tool.

The second differentiator is problem type. Five basic problem types were identified, which provide key insight into the knowledge required in the structuring process. Between the problem types, there are differences in problem objective, starting point for structuring, focus of the structuring process, and core processes used in resolving the problem. These are key items of information for a knowledge reuse tool.

The third differentiator is the decision pattern, the organization and sequencing of decisions within a decision problem. Thirteen decision patterns are identified, which collectively cover the major examples of decision problems found in the literature. The key distinction in these patterns is their emphasis on whether information learning occurs within the decision model and the basic flow structure through the decision model. These decision patterns help identify needed modeling enhancements and are themselves useful information in guiding a structuring process.

The fourth differentiator is the magnitude of uncertainty. Uncertainty shapes the structuring approach and the resulting decision model. As uncertainty increases, the pro-

cess increases its emphasis on mitigating the effects of uncertainty. Information on how to deal with uncertainty is a useful element for a knowledge reuse tool. Collectively, these differentiators identified needed decision graph modeling tool enhancements and knowledge reuse tool requirements and are a contribution to the decision-making state-of-the-art.

Chapter 4 identifies the limitations of the currently implemented local probability distribution (LPD) capabilities and then designs solutions that enhance implementation of first-order expressive probabilistic modeling. This research created dependency modeling to explore the range of interactions between parent RVs in influencing the child RV's local probability distribution (LPD). Chance RVs in any first-order expressive probabilistic modeling framework are based on knowledge representation / engineering concepts. They explicitly model an entity's attributes, relationships, and functions. Each has a distinct characteristic and interact differently with other types. These concepts are widely used to model knowledge in many domains. Exploring at this level means the results are potentially applicable to those domains. For each parent / child combination type, a scenario was developed and modeled. The LPD in the model was analyzed to determine the behaviors required to create that LPD. The process is also described in detail in Appendix A. These models were then mined to find a robust set of LPD creation behaviors. The existing UnBBayes-MEDG LPD language was used as a baseline to identify which of these behaviors required new language capabilities. *The research's second contribution are a set of LPD behaviors that provide a robust capability to any first-order expressive modeling tool and includes ten new LPD development language capabilities that significantly*

extend the range of problems that can be modeled. They are first described behaviorally, and then are given a MEBN/MEDG LPD language implementation form to assist in understanding the required behavior. The behavioral description allows developer of other first-order expressive modeling systems to understand what the specific capability is doing and then implement in their system-specific language. The model scenarios also provide a set of implementation-independent test cases for validating the LPD language enhancements. *The third contribution from this research are eight design patterns for first-order expressive modeling that have not been comprehensively documented in the literature*. In going through each of the dependency model cases, eight structural patterns became apparent, in addition to LPD behavior. Two of the patterns have been described in the literature, the other six were not. These patterns are general to first-order expressive modeling and useful as guides for in that modeling. The results are also fully documented in Appendix B.

Chapter 5 describes this research's *fourth contribution, a set of first-order expressive decision graph modeling enhancements*. One problem addressed is how a decision graph modeling tool can effectively model and solve problems that have significant asymmetry. Asymmetry occurs when previous decisions or learned information alter the allowable paths or states in the network. The enhancements implement automated support for asymmetry handling for decision problems with varying numbers of involved entities. These enhancements affect the model design, model visualization, structuring algorithms and solution process. An example problem is used to demonstrate the viability of the enhancements.

A second decision graph tool enhancement is an efficient approach to modeling problems with significant context variation, where a change in context changes the desirable decision attributes, the consequences / criteria used to evaluate the alternatives, and the value placed on those criteria. Having a model that encompasses the full range of variation can result in large models. This enhancement allows only those model elements relevant to a specific context to be instantiated in the model, reducing computer resource demands. The third enhancement is to define needed modeler actions in modeling of problems with varying entity counts, either as states in an RV or as objects of a decision.

Chapter 6 addresses knowledge reuse and develops a decision-making focused knowledge reuse tool. It would greatly assist the human modeler if knowledge relevant to a problem were packaged in a form that clearly showed what was already known about a problem, and what information the modeler needs to develop/elicit. *The fifth contribution is the decision template,* a tool that integrates the knowledge needs identified by the differentiator analysis, the decision problem elements ontology, and the results of a decision structuring decomposition process into a coherent knowledge reuse format. It also incorporated the various decision problem decomposition products used in the field. The decision template is adaptive. It begins with a general top-level template that is a shell for potential information content. It can be developed out to support a problem class. Finally, one can develop it into a detailed support template for a specific decision problem.

To test the effectiveness of the concept, a simplified decision template package was developed to support a class project in a heterogeneous data fusion class at George Mason University. The project was to develop a data fusion / decision support system for

a fighter aircraft self-defense system. The students in the class formed six teams, and each team developed its own version of the system. The system architecture naturally divided into a fusion element and a decision support element. Based on the benchmark model, the two elements were roughly equivalent in model complexity. Two decision templates were created, one for the fusion element and one for the decision support element. Three teams were given one template, while the other three were given the other. Each student was asked to provide data on how much time they worked on different aspects of the project, and to complete a post-project evaluation. Twelve of 13 students participated. 75% indicated the decision template was a useful tool. The collected time data had significant limitations, but it provided some evidence that the decision template provided 50 – 70% time reduction in developing the described model element versus the model element for which they did not receive a template.

Chapter 7 identifies avenues for further development and summarizes the effort.

CHAPTER 2
BACKGROUND

The relevant background supporting this work will be covered in five sections. The first discusses the general background for formal decision-making. The next examines problem structuring concepts and approaches within the literature. The third covers graphical modeling approaches to visualize and analyze the results of problem structuring. The fourth section discusses knowledge representation and reuse concepts. The final section describes Multi-Entity Bayesian Networks (MEBN) and Multi-Entity Decision Graphs (MEDG).

2.1 Decision-making

Decision-making has been examined as an important human activity for a long time (Buchanan and O'Connell 2006). But the discussion was qualitative until the 18th century, when Bernoulli published his "Exposition of a New Theory on the Measurement of Risk" (Bernoulli 1738). This paper presented the idea of quantifiable utility. In 1926, Ramsey wrote the first paper linking subjective probability and utility (Ramsey 1931). Modern approaches to formal decision-making trace to Von Neumann and Morgenstern's normative theory of decision-making, laying out a set of axioms for decision-making under conditions of uncertain knowledge (von Neumann and Morgenstern 1947).

Decision analysis is the normative approach to decision-making. It is built upon Von Neumann and Morgenstern's work. The mathematical foundation is decision theory,

developed by Savage (1954) and fully codified by 1964 (Pratt, Raiffa, and Schlaifer 1964). Decision analysis is more than decision theory. Howard, who coined the term "decision analysis" in 1966, defined it as

> *"… a logical procedure for the balancing of the factors that influence a decision. The procedure incorporates uncertainties, values, and preferences in a basic structure that models the decision. ……. The essence of the procedure is the construction of a structural model of the decision in a form suitable for computation and manipulation; the realization of this model is often a set of computer programs*. (R. A. Howard 1966).

Numerous researchers made key contributions to decision analysis. Keeney and Raiffa produced the standard reference for multi-criteria decision-making (Keeney and Raiffa 1976). An early summary of the decision analysis process is the Handbook for Decision Analysis (Barclay et al. 1977). Von Winderfeldt and Edwards provided a comprehensive description of organizational and psychological considerations (Von Winterfeldt and Edwards 1986). Keeney provided an extensive development of user objective analysis (Keeney 1992). Standard works include Watson and Buede (1987), Raiffa (1997), Peterson (2009) and Clemen and Reilly (2014).

All decision problems occur within a decision environment, sometimes called a decision context (Clemen and Reilly 2014) or a decision situation (Zsambok 1996). Within this environment, they have four characteristics:

- A disconnect between the current state and a desired state, where "desired" is defined by the decision-maker in terms of one or more fundamental objectives
- Two or more viable alternatives that might achieve the desired state
- Each alternative has a set of consequences, either positive or negative, that affect the fundamental objectives

- One or more of these consequences vary between alternatives
- One or more criteria, mapped to the consequences, by which to evaluate those alternatives
- A degree of uncertainty about the above characteristics. Uncertainty may range from none to complete ignorance (Grünig and Kühn 2013).

A formal decision process also requires:

- The consequences of each alternative under each chance outcome must not only be known, but the value to the decision-maker must be definable in some quantitative form (ordinal or cardinal)
- A well-defined rule or set of rules that identifies what the best decision is
- A specific formal procedure available by which to do this (White 1969).

A decision problem meeting these characteristics is considered well-structured.

While decision analysis is normally taught as proceeding to a decision recommendation, it may stop short of that. Instead, it may provide a decision-maker a:

- Complete model of the problem, including available decisions alternatives, but makes no recommendation
- Downselected set of alternatives for consideration
- Sorted list of decision alternatives into meaningful categories for evaluation (e.g. “Implementable now”, “Needs modification”, “Not recommended”)
- Ranked order list of alternatives, without a recommendation (Roy 2005).

2.2 Decision Problem Structuring

Decision problem structuring has been defined as "…the process by which a decision situation is transformed into a form enabling choice" (Dillon 2002). The structuring process should deliver a decision problem model that address the decision-maker's information needs and be of such a quality that further analysis is unlikely to give significant new insights. Phillips terms this a requisite decision model (Phillips 1984).

An important question is what the basic structuring process is. Mintzberg et al. provided a key insight over 40 years ago – decompose the problem into known elements:

> *"This research indicates that, when faced with a complex, unprogrammed situation, the decision-maker seeks to reduce the decision into subdecisions to which he applies general purpose interchangeable sets of procedures and routines. In other words, the decision-maker deals with unstructured situations by factoring them into familiar structurable elements" (Mintzberg, Raisinghani, and Théorêt 1976).*

This observation highlights the importance of knowledge reuse in the decision-making process. But how to perform problem structuring is an open question. In 1980 and again in 2007, von Winterfeldt said

> *"Structuring decision problems into a formally acceptable and manageable format is probably the most important step of decision analysis. Since presently no sound methodology for structuring exists, this step is still an art left to the intuition and craftsmanship of the individual analyst" (von Winterfeldt 1980, von Winterfeldt and Edwards 2007).*

Belton and Stewart place problem structuring within an iterative five step solution process. It includes:

- Problem identification
- Problem structuring: Determine stakeholders, values, goals, external environment, constraints, key issues, initial alternatives, uncertainties

- Model building: Refine alternatives, define criteria, elicit detailed values
- Use model to inform and challenge thinking: Synthesize information, challenge intuition, create new alternatives and conduct robustness and sensitivity analyses
- Develop action plan (Belton and Stewart 2002).

The literature outlines several structuring approaches. One is a basic intuitive approach, guided by the decision analyst's experience and implicit knowledge. Another approach is to use a taxonomic approach, in which relevant problem characteristics are mapped against prototypical decision types, and the analytic structure for the most appropriate type match used (Vári and Vecsenyi 1984). Two different ways of defining this taxonomy have been presented. Brown and Ulvila published a detailed taxonomy structure, matching problem structure to analytic characteristics (Brown and Ulvila 1976). Brennan et al. proposed a similar taxonomic approach for modeling health technology options (Brennan, Chick, and Davies 2006). Von Winterfeldt proposed an alternative approach, that the taxonomy be based on substantive differences in the problem content. He suggested that each problem type, such as site selection or regulatory development, had distinctive characteristics. He identified several examples, but never fully developed the approach (von Winterfeldt and Edwards 2007). A third structuring approach is to use prepackaged modeling software to structure the problem using the features of the modeling system. This approach is called a Decision Support System (DSS) generator (Bonczek, Holsapple, and Whinston 1981).

Decision problem structuring research in the last 15 years has focused on multicriteria decision problems with multiple stakeholders (Scheubrein and Zionts 2006; Fran-

co and Montibelller 2009; Belton and Stewart 2010). Integrating the objectives, criteria and values of multiple stakeholders drove the structuring focus to the use of Soft Operations Research (Soft OR) techniques / Problem Structuring Methods (PSMs). Focused on supporting decision problem knowledge elicitation, these include:

- Strength / Weakness / Opportunity / Threat (SWOT) analysis (Sorenson and Vidal 1999)
- Strategic Choice Approach (SCA) (Friend and Hickling 2005)
- Scenario methodology (O'Brien 2004; Montibeller, Gummer, and Tumidei 2006; Stewart, French, and Rios 2013).

The predominant approach today is the DSS generator. A DSS generator provides tools and capabilities for creating a problem-specific DSS, including structuring the decision problem (Power and Sharda 2007). Examples of DSS generators include Analytica®, Decision Programming Language®, Expert Choice®, Logical Decisions® and Precision Tree®. A limitation of these generators is they provide very limited, if any, tools for uncovering and reusing previous decision knowledge.

Not all problems can be well-structured. Rittel and Webber identified the characteristics of what they termed a "wicked problem" - those problems for which there is no definitive formulation of what the problem is, or decision-maker agreement on the elements of the problem. In addition, there is often no agreement on the decision-making rule (Rittel and Webber 1973). Ackoff identified a similar concept, that he called "messy problems" (Ackoff 1974). Many social problems that involve multiple groups tend to be wicked or messy problems. Snowden extended the idea of a wicked problem with his

Cynefin model, which highlights the importance of being able to determine and understand cause and effect relationships. In those cases where cause and effect can only be known after the fact (post decision), or where the situation is so new that cause and effect are not discernable, decision-making becomes an iterative "probe and evaluate" activity (Snowden and Boone 2007). Lipshitz and Strauss identified a different type of ill-structured problem, one in which the decision-maker did not have enough information to make a sound decision. In that case, they noted a decision-maker often used several techniques to stall making a decision (Lipshitz and Strauss 1997).

2.3 Graphical Models

The outcome of a structuring process is a decision model. There are two desirable characteristics in such a model:

- It presents information in a human understandable form
- It is executable. The model has an underlying mathematical / logical engine that allows one to draw various inferences from the model, and to examine effects of changes in model structure or parameters.

Human beings have a natural ability to grasp information presented visually. This makes graphical modeling an attractive approach. Such models highlight the key relationships between the model elements. Decision analysis has developed or adopted several graphical modeling approaches. They can be binned into four categories. The first focus on modeling situational factors whose presence / absence affects the consequences of decisions. Commonly used models include belief networks (Pearl 1988), fault trees (Ericson 1999), and event trees (American Institute of Chemical Engineers 2008). The second cat-

egory map decision alternatives to the utility (or value) they provide the decision-maker. These include means-ends diagrams (Manheim and Hall 1968), decision trees (Terek 2005), and decision graphs / influence diagrams (Ronald A. Howard and Matheson 1984; F. V. Jensen and Nielsen 2007). The third category focuses on objectives / values presentation, highlighting the factors important to the decision-maker. Objective hierarchies (Granger 1964) / Value Trees (Von Winterfeldt and Edwards 1986) and consequence tables (Gregory et al. 2012) are included here. The fourth are supporting tools such as mind maps (Buzan and Buzan 1996) and Strength / Weakness / Opportunity / Threat (SWOT) analysis charts (Sorenson and Vidal 1999).

Historically, decision trees have been widely used, but they become unwieldy for complex problems. Other than decision trees, the various graphical models list above in the first two categories are types of network diagrams. These diagrams model key attributes and relationships as nodes in a network, with the arcs between nodes representing dependencies between nodes. Network diagrams provide decision information more compactly than decision trees for almost all decision problems. Two important types of network diagrams are Bayesian networks and decision graphs.

2.3.1 Bayesian Networks

Bayesian networks (BN) are an efficient tool for reasoning about information with probabilistic uncertainty (Pearl 1988). A Bayesian network B = (N, V, E, PN) provides a factorized joint probability distribution on a set of random variables N. The network has a directed acyclic graph (V, E), where each vertex $v_i \in V$ represents a specific random variable $n_i \in N$. A directed edge $e_j \in E$ represents a conditional dependency between two

random variables. Each random variable n_i has two or more mutually exclusive and comprehensively exhaustive states. For each random variable n_i, there is a set of zero or more random variables called the *parent set* of n_i, $Pa(n_i)$, each having a directed edge to n_i. A *parent configuration* (PC) is the assignment of a specific state to each random variable in the parent set. Finally, each RV n_i has a set P_N that has a probability function that assigns a conditional probability $P(n_i | Pa(n_i))$ to each state of n_i for every PC in the parent set. The conditional probabilities collectively are called a local probability distribution (LPD). For any RV n_i that has a finite number of states and a finite PC, it can be represented in tabular form, normally called a conditional probability table (CPT). The joint probability distribution for a set of specific random variable states in (N, P_N) is

$$\prod_{n_i \in N} P((n_i)|Pa(n_i))$$

Equation 1: Bayesian network joint probability distribution

Bayesian networks can reason probabilistically, logically (set probabilities to 0/1) or qualitatively (set probabilities to - / 0 /+) (Wellman 1990). This makes them a powerful tool for modeling acyclical problems.

2.3.2 Decision Graphs

Decision graphs (DG) provide a compact means of visualizing a decision problem. They are a merger of influence diagram visualization concepts with Bayesian network probability management concepts. Influence diagrams model dependencies between elements of a decision problem. They provide a broad range of capabilities, being able to

represent deterministic, probabilistic and functional cases (Ronald A. Howard and Matheson 1984). Influence diagrams predate Bayesian networks and did not have the full set of tools needed to correctly model probabilistic dependencies or to propagate evidential impacts through a network (Pearl 2005). Later work by a number of researchers provided the theoretical underpinnings to merge the two, resulting in decision graphs (F. V. Jensen and Nielsen 2007). Figure 1 shows the elements of a decision graph. They have three node types: chance nodes (identical to Bayesian network nodes); decision nodes, whose states are the decision alternatives; and value nodes (also called utility nodes), which assign utility values to each parent configuration. The arcs between the nodes identify dependencies between nodes. Probabilities and value information that are explicitly included in a decision tree are documented in tables or as functional equations associated

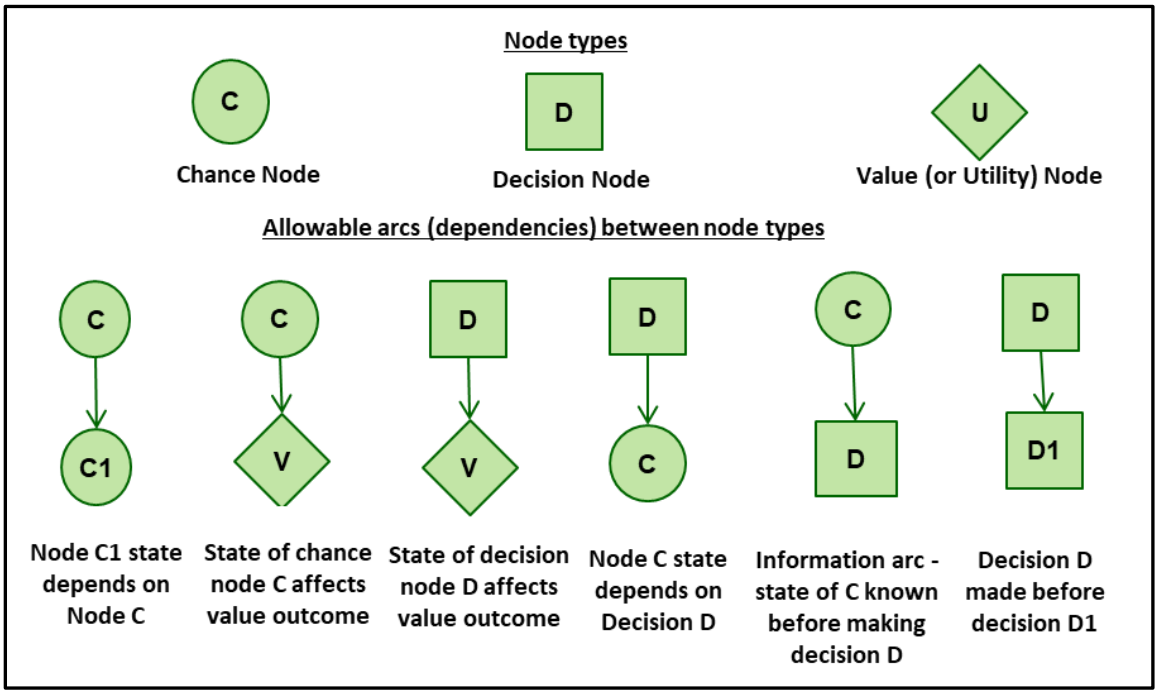


Figure 1: Decision graph elements

with each node and are not normally displayed on a decision graph. One can propagate evidence in a decision graph, using new evidence to change one's understanding of the various probabilities on the graph. Evidence propagation in a decision graph can use the same exact or approximate algorithms as Bayesian networks. When propagating evidence, utility nodes and arcs into decision nodes are ignored (Jensen and Nielsen 2013).

No-forgetting is commonly assumed in decision graphs that have multiple decisions. This means that for every decision, the decision-maker remembers both the choices made in all previous decisions in the model and all information provided via information arcs to previous decisions. The sequence in which they occur is called the decision past. For any decision, decision past items before that decision may influence that decision. However, they may not all do so. Those that could influence the decision are called the required past for that decision. Required past for a decision node is analogous to parents for a chance node. For each possible combination of the required past states, there is an optimal decision (Nielsen and Jensen 2004). The collection of all optimal decisions for all required past states is the optimal policy for that node.

A decision problem is symmetric if, in the decision tree model, every decision past element is included in the same order on every path through the tree. Problems without this characteristic are called asymmetric. In asymmetric problems, the decision sequence through the graph can be dynamic, with learned information or previous decisions changing the path. A decision graph then is a representation of possible decisions and learned information, some of which may not occur in the model's execution. Such a decision graph provides a plan of what could occur and what the appropriate responses are.

There are three forms of decision asymmetry. Functional asymmetry occurs when a previous decision or learned information disallows one or more alternatives in a follow-on decision. Structural asymmetry occurs when a previous decision or learned information nullifies the effects of one or more decision graph nodes in the decision problem under consideration. Under structural asymmetry, one can say that there are multiple versions of the decision graph, guided by the previous decisions and learned information. In order asymmetry, one has a sequence of decisions, with learning occurring between subsets of decisions. This learning can affect the value of follow-on decisions, and the order in which decisions should be made is itself a decision problem.

For a symmetric decision graph, finding the optimal policy is straightforward. The original solution algorithm is the Shachter algorithm, and like solving a decision tree, it uses a roll-back algorithm (Shachter 1986). Alternate approaches to solving an influence diagram include variations on the junction tree algorithm (Shachter and Bhattacharjya 2010; Frank Jensen, Jensen, and Dittmer 1994; Madsen and Nilsson 2001), game trees (Shenoy 1998) and valuation networks (Shenoy 1992). If the problem being modeled is asymmetric, then either the problem must be made symmetric by adding dummy nodes, states or arcs to the graph, or by using a solution algorithm designed to cope with asymmetry. These approaches are discussed in Chapter 5.

2.4 Knowledge Reuse

Knowledge reuse is desirable in decision-making, as it reduces the cost of making the decision, and provides opportunities to improve decision quality. Most often, a complete body of knowledge from one decision is not reusable in toto, but elements of it are.

Potentially reuseable elements of knowledge are called knowledge fragments. The original concept of a knowledge fragment was that it encapsulates either a relationship on a set of entities or define one or more attributes of an entity. A Resource Description Framework (RDF) triplet is one of the most elementary knowledge fragments (Herman et al. 2015). A more complex knowledge fragment concept is a rule of the form "If A, then B" (S. J. Russell and Norvig 2018). The definition of knowledge fragment used in this dissertation is derived from the network fragment (Laskey and Mahoney 1997). For some collection of entities, a knowledge fragment is a set of related entity attributes and relationships together with the dependencies (possibly probabilistic) among them, which captures both concepts expressed above. The idea of developing and integrating knowledge fragments is widespread. For instance, it is used in failure modes and effects analysis (Teoh and Case 2005), military planning (Ryan 2007), medical product development (Mohan, Jain, and Ramesh 2007), social network analysis (Santos et al. 2008), system engineering requirements (López, Astudillo, and Cysneiros 2008), and Information Technology security (Fenz, Parkin, and van Moorsel 2011).

Knowledge fragments can be formed into a knowledge base about some domain. Following Brachman and Levesque, five basic concepts define a knowledge base that describes some domain. The first is *entity*, or named individual. These are the domain elements. Each entity is *typed*, using a unary predicate that assigns each entity to a class. The features of an entity are described by *attributes* (also called properties or data type relationships), each one assigned by a unary predicate. *Relationships* (also called role or object type relationship) between entities are described by n-ary predicates. Finally,

functions are a special form of relationship, in which the object of the relationship is a specific entity assigned from a mutually exclusive collectively exhaustive class of entities (Brachman and Levesque 2004).

There are two basic approaches to developing knowledge fragments: ontologies and rules. An ontology focuses on a common understanding about a domain of interest between agents (human or machine). It is a formal vocabulary of terms for a specific domain of interest. It provides a rigorous descriptive model that defines a term by describing its relationships with other terms in the ontology, using explicit semantic representations and logic-based formalisms (Tetlow et al. 2006). An ontology is a specification of domain knowledge, defining the technical means for working with knowledge. Ontology uses include information integration, information retrieval, semantically enhancing content management, knowledge management and support for community portals (Grimm 2010). The Web Ontology Language (OWL) is a standard and widely used language for specifying ontologies (Consortium 2012). A rule-based approach describes what actions or inferences may be drawn given a set of rules and facts. Rules are a collection of if-then statements, while facts are assertions about entities in the domain of interest (Grosan and Abraham 2011). Bayesian networks and decision graphs may be seen as an extension of a rule-based system that requires completeness of all the rules and incorporates uncertainty about the rules.

2.5 First Order Expressive Graphical Models

2.5.1 Background

Both BNs and DGs are propositionally-logic expressive – both the number of nodes (chance, decision and utility) and the parent set of each node are fixed. This limits their applicability because they cannot model most dependencies between a variable number of domain entities (Milch and Russell 2010). The case where the number of entities can vary between model instantiation is known as first-order expressivity. Development of first-order expressive Bayesian networks has been part of a larger effort to develop first-order expressive probabilistic logics. These logics merge first-order logic (or a fragment of first-order logic) with uncertainty reasoning (Poole 2011; S. Russell 2015). Modeling capabilities for first-order expressive probabilistic logics developed along three tracks: probabilistic description logics, probabilistic logic programming and probabilistic graphical models.

Description logics are fragments of first-order logic with an emphasis on computational tractability and decidability. Description logics are strongly associated with knowledge representation and reasoning efforts (Baader et al. 2003). They provide a semantic and reasoning basis for variants of OWL (McGuinness and Van Harmelen 2004; Consortium 2012). Early work in extending them probabilistically started with P-Classic (Koller, Levy, and Pfeffer 1997). Probabilistic versions of the most expressive description logic languages used in OWL have been developed, called P-SHOQ and P-SHOIN (Lukasiewicz 2008). The logic programming track extended the concepts of logical programming languages like Prolog, which have a first-order expressivity for models with-

out uncertainty, by adding the ability to address probabilistic uncertainty. Efforts included Independent Choice Logic (Poole 1997), Bayesian Logic (Milch et al. 2007), and Church (Goodman et al. 2012). The probabilistic graphical models track extended graphical models from a fixed-entity or limited variable entity count propositional logic level to a first-order level. The probabilistic graphical models track itself has two subtracks: Markov (undirected) networks and Bayesian (directed) networks. Markov networks use undirected networks, allowing them to model cycles. This makes them useful in domains such as visual recognition, social network analysis, and object identification. The downside of Markov networks is that the potential functions in such networks must be specified globally. Change to any part of the network requires a respecification of all the potentials assigned to every node in the network (Koller and Friedman 2009). In addition, their lack of direction means they cannot model causality, only correlation. Markov networks have a propositional expressivity. The extension of Markov networks to first-order logic is known as Markov Logic Networks (Richardson and Domingos 2006; Domingos and Lowd 2009). For Bayesian networks, the emphasis has been on extending Bayesian network capabilities from a fixed set of entities (established by the model builder) to a capability that could model problems where the number of entities vary between problem instantiations. This active research area resulted in such capabilities as Object Oriented Bayesian Networks (OOBN) (Koller and Pfeffer 1997), Probabilistic Relational Models (PRM) (Getoor et al. 2007), Probabilistic Entity-Relationship Models (Heckerman, Meek, and Koller 2007), Object-Oriented Probabilistic Relational Modeling Language (OPRML) (C. Howard 2010) and MEBN (Laskey 2008).

First-order Bayesian network capabilities have four primary characteristics beyond a Bayesian network. First, they have some form of template structure that identifies the types of RVs and the dependencies between RV types in a model. There is a basic requirement that the set of templates be logically consistent. Second, they require a knowledge representation structure. There has been significant efforts to integrate OWL with first-order Bayesian network tools (Paulo Cesar G. Costa 2005; Matsumoto, Carvalho, Costa, et al. 2011; Rommel N. Carvalho, Laskey, and Costa 2017). Third, there is a database that identifies the entities in a specific instance of the model and has findings (evidence) on zero or more RVs of specific entities. Fourth, it has a specification for how the local probability distribution is created for each RV, given a varying number of parent entities for each RV. In addition, a first-order Bayesian network requires an adjustment to its inferencing process. The older approach is to create a *grounded Bayesian network* for a given template structure, knowledge base, and LPD specification. The grounded Bayesian network is an ordinary Bayesian network that has the specific entities, attributes and relationship captured in the knowledge base. It can be queried and updated using standard Bayesian network techniques (Mahoney and Laskey 1998; Pfeffer 1999). Grounded networks can be very large and possibly be computationally intractable. A significant research effort is developing lifted inferencing, which limits the degree of grounded modeling required (Kersting 2012; Kimmig, Mihalkova, and Getoor 2015).

2.5.2 Multi-Entity Bayesian Networks

A recent review of implementations of first-order Bayesian network modeling found three viable implementations: PRM, MEBN and OPRML (C. Howard and

Stumptner 2014). A review of probabilistic reasoning methods for use in automated driving situational awareness identified MEBN, with a fuzzy logic extension, as the most broadly capable approach for probabilistic first order logic modeling in this domain (Golestan et al. 2016).

MEBN represents the world as a collection of inter-related entities, with relationships between them. Entity and relationship knowledge are modeled as a collection of reusable knowledge elements, called MEBN Fragments (MFrags). A MEBN Theory (MTheory) is a set of well-defined MFrags that collectively satisfy first-order logical constraints ensuring a unique joint probability distribution (P. C. G. Costa and Laskey 2005). Formally, an MFrag ***F*** = ***(C, I, R, G, D)*** consists of a finite set ***C*** of *context* value assignment terms; a finite set ***I*** of *input* random variable terms; a finite set ***R*** of *resident* random variable terms; a *fragment graph* ***G***; and a set ***D*** of *local probability distributions*, one for each member of ***R***. The sets ***C, I***, and ***R*** are pairwise disjoint. The fragment graph ***G*** is an acyclic directed graph whose nodes are in one-to-one correspondence with the random variables in ***I*** ∪***R,*** such that random variables in ***I*** correspond to root nodes in ***G***. Local probability distributions in ***D*** specify the conditional probability distributions for the resident random variables (Laskey 2008).

Figure 2 shows an example MFrag. This simplified example models the informal statement that a student's performance in a course depends (partially) on a professor's teaching ability. In MEBN, *ordinary variables* (OV) are placeholders for entities from some domain. The pentagonal nodes are context nodes. They represent the conditions that must be met for the MFrag to be applicable. In this example, the two IsA context nodes

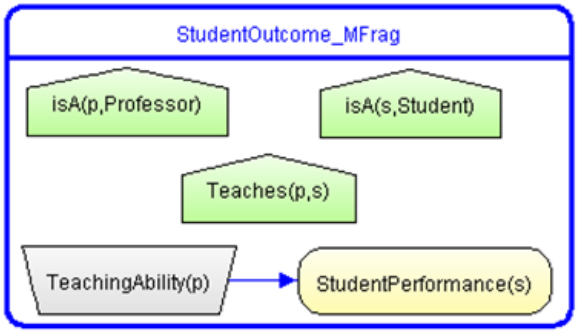


Figure 2: MFrag example

identify the domain class from which an OV takes entities. If there are no entities in a domain, this MFrag is not used.

The context node *Teaches(p, s*) specifies the condition that a specific professor entity must teach a specific student entity for this MFrag to have meaning. *Teaches(p, s)* is an RV defined in a separate MFrag. The rounded node is a resident node. Each RV is a resident node on one MFrag, which defines its states and provides a local probability distribution defining the probability of each state depending on the state(s) of its parent(s). The trapezoidal node is an input node, signifying an RV that is a parent RV that is defined in another MFrag. The arrow between the input and resident nodes indicates there is a probabilistic dependency between them. An MFrag must have one resident node. It can have as many resident, input and context nodes as a designer wants to incorporate. However, small MFrag sizes facilitate knowledge reuse. To represent a domain or problem, the modeler creates one or more logically consistent MFrags that define the domain. There is also a list (possibly varying) of entities in the domain. Finally, there may be finding MFrags, which provide observed evidence about RV states for specific entities. In the example above, the entities could be “Kathy is a professor”, “Paul is a student”, and the

finding MFrags be “Kathy teaches Paul”, and “Kathy has good teaching abilities”. An introduction to MEBN can be found in P. C. G. Costa and Laskey (2005).

The MFrags are the templates for the grounded Bayesian networks, called Situation Specific Bayesian Networks (SSBN). An SSBN is a standard BN and any BN solution algorithm can be used on it. One current MEBN implementation is UnBBayes-MEBN (Matsumoto, Carvalho, Ladeira, et al. 2011), which includes a Knowledge Representation (KR) capability to model the entity class structure and key relationships between entities. It uses the Probabilistic Web Ontology Language (PR-OWL), a probabilistic extension of OWL. The latest PR-OWL version uses Protégé® as its knowledge representation tool (Rommel Novaes Carvalho 2011). It also includes an LPD script language for creating scripts that define the LPD for each RV state given the possible variations in the RV’s parent configurations.

2.5.3 Multi-Entity Decision Graphs

The Multi-Entity Decision Graph (MEDG) is a first-order expressive decision modeling framework. It extends MEBN by adding the elements of a decision graph and incorporating decision theoretic reasoning. A MEDG Theory is defined by one or more MEDG Fragments (called MFrags when the reference to MEDG vice MEBN is clear) along with the associated list of entities and finding MFrags. Three types of nodes can be resident on a MEDG Fragment: probabilistic, decision, and utility. The interpretations of the nodes and the arcs between them follow the decision graph conventions (Section 2.3.2 / Figure 1 on page 22). The only known MEDG implementation is

UnBBayes-MEDG[1]. This implementation uses an extended version of the MEBN Local Probability Distribution script language (shortened to LPD language when the reference to MEDG or MEBN is clear) that allows it to specify a real-number valued utility distribution for utility nodes (MEDG decision nodes have no local probability distribution). The implementation uses PR-OWL-Decision (PROWL-D), a decision problem focused framework for building and reasoning about probabilistic ontologies. It is an OWL-based ontology language that syntactically and semantically supports near first-order expressiveness[2] and uncertainty, with the explicit representation of decisions and utilities of a decision maker. Documents written in PROWL-D can be understood to be computer-readable representations of MEDG Theories. (Matsumoto 2019). A Situation Specific Decision Graph (also called a Situation Specific Influence Diagram) is a grounded decision graph model instantiated from the MTheory and associated set of entities and finding MFrags.

The UnBBayes-MEDG implementation has a significant limitation – it can correctly structure decision problems where the decision alternatives are varying set of entities from a class (rather than being a fixed list), but it cannot provide a solution recommendation. This is caused by a shortcoming in the LPD language – it has very limited abilities to create LPDs for parent configurations that have parents whose states are enti-

---

1 https://sourceforge.net/projects/unbbayes/files/UnBBayes%20Plugin%20Framework/Plugins/Probabilistic%20Networks/MEDG/. Using this plug-in requires loading the UnBBayes framework and several other plugins.

2 It specifically use OWL-DL, which uses a description logic fragment of first-order logic and does not have full first-order expressiveness.

ties. This limitation affects both MEBN and MEDG. The discovery of this limitation raised questions as to the range of capabilities an LPD language for first-order expressive modeling should have. A literature review found no comprehensive discussion on this topic. There were discussions on handling of individual types of first-order uncertainties (e.g. attribute uncertainty (Pfeffer 1999), reference and property uncertainty (Getoor et al. 2007), identity uncertainty (Pasula et al. 2002), and number uncertainty (Milch et al. 2007)) and discussions about LPD approaches that simplify the workload of (e.g. independence of causal influences ((Dıez and Druzdzel 2006), aggregators ((Kazemi et al. 2017)). But nothing discussed the range of behaviors required to create LPDs in a first-order expressive modeling tool. Identifying the behaviors an LPD language needs to define a broad scope of possible LPDs is a driving requirement.

CHAPTER 3
DECISION PROBLEM DIFFERENTIATORS

Chapter 2 outlined the common characteristics of decision problems and the features of decision graphs that model those characteristics. Closing research gaps that limit effective decision graph implementations and knowledge reuse tool development also require an understanding of the differentiators between different decision problems. These differentiators establish the range of capabilities needed in decision graph implementations and the knowledge categories needed in a decision knowledge reuse tool.

The literature identifies a significant number of different classes of decision problems. A partial list is in Figure 3. Each class has common characteristics in each class that are common among specific instances of a problem class that also differentiate the class from other decision problems. As an example, Von Winterfeldt observed that facility siting has the following characteristics:

- Sequential screening from candidate areas to possible sites, to a preferred set, to final site-specific evaluations
- Multi-objective nature with emphasis on generic classes of objectives: investment and operating cost, economic benefits, environmental impacts, social impacts, and political considerations
- The process of organizing, collecting, and evaluating information is similar in many problems (von Winterfeldt 1980).

- Budget allocation
- Choice selection
- Condition / action
- Contingency planning / management
- Diagnostics / Repair
- Energy
- Facility siting
- Financing
- Go / No Go
- Legal decisions
- Manufacturing & Services
- Marketing
- Medical diagnostic problems
- Medical management
- Military Planning
- New Problem Resolution
- Personnel
- Private investment decisions
- Procedural management
- Procurement
- Product development
- Product mix selection
- Public Policy / regulation
- Reorganizations
- Resource development
- Scheduling / sequencing
- Services offering
- Situation Recognition
- Social conflict resolution
- System / Process controls
- Technology investment
- Workflow

Sources: Orasanu and Fischer 1996, Nutt 2001, Keefer, Kirkwood and Corner 2004, von Winterfeldt and Edwards 2007

Figure 3: Decision problem class examples

But not all of these characteristics are unique to facility siting. With different details, they are characteristic of many types of decision problems. It includes choice problems (e.g. choosing a college, buying a new personal computer) and procurements for which many bidders are anticipated. These three characteristics are examples of possible differentiators. Sequential screening is an example of a decision pattern. The multi-objective nature with specific classes of objectives are a specific example of the contents of a decision model. The information management process is a specific case of a core structuring process.

This chapter identifies four significant differentiators developed from a literature review of decision problem characteristics. They are context, basic problem type, decision patterns, and degree of uncertainty. In addition, this research organized the elements of a decision problem into a decision problem element schema. This schema does three things: First, it identifies the elements of a decision problem. Second, it defines the key

relationships between the problem elements, which is needed when modeling multi-entity decision problems. Third, it establishes a core modeling pattern applicable to every decision graph-based model and identifies where decision problem structuring results are mapped to the decision model. Collectively, the differentiators and the decision problem elements schema contribute to advancing the decision modeling state-of-the-art by identifying needed capability enhancements in decision graph modeling tools and knowledge reuse tool content.

3.1 Decision Problem Characteristics: Literature Review

The decision-making literature describes multiple decision problem characteristics, and researchers have developed taxonomies based on these characteristics. In most cases, these focus on one or two factors or on a specific domain. Degree of uncertainty is one key characteristic. For example, Scherpereel developed a three-class taxonomy, based on the degree of uncertainty and dynamics in the problem (certain / static properties, probabilistic uncertainty / defined dynamic properties and genuine uncertainty / complex dynamics), which is a common division in the literature (Scherpereel 2006). Snowden's Cynefin model focused on the uncertainty in understanding the cause-and-effect relationships in a decision problem. This model classifies problems into four types based on the degree of cause / effect uncertainty (known, probabilistically known, vaguely and incompletely known, and completely unknown) (Snowden and Boone 2007).

Another characteristic is general problem types. Some authors divided problems into binary classes, such as a basic difference between choice problems, where the alternatives are known at the start of the decision structuring process, and design problems,

where the alternatives have to be unearthed or created (Simon 1967). Others divided them into threat versus opportunity problems (Grünig and Kühn 2013), or planning versus control decisions (Walther 2018). Alternate approaches classify problem types into specific areas, such as Nutt's classification of eight decision areas in industry (technology, reorganizations, controls, marketing, products and services, personnel, financing inputs, and location) (Nutt 1993) or a noncomprehensive breakout of application areas (Keefer, Kirkwood, and Corner 2004). Additional decision problem characteristics include management level / problem scope (strategic, operational, tactical) (A. J. Rowe 1962) or problem solving approach (Nickols 2012).

Several authors organized several characteristics into a structure. Thrall (1984) suggested that decision problems have at least three significant characteristics: the number of decision-makers, number of decision criteria, and number of decisions. Kleindorfer, Kunreuther, and Schoemaker (1993) identified context, decision-maker structure and eleven decision problem characteristics as a structure for differentiating decision problems. Reich and Kapeliuk (2005) suggest that there are four dimensions for organizing the decision problem space: the task to be solved, available technologies, organizations, and the people involved. Grünig and Kühn (2013) organized multiple characteristics to differentiate decision problems.

In reviewing the literature for effects on decision graph modeling tool capabilities and knowledge reuse, four differentiators became evident. First, context was identified as having broad effects on decision-making. Second, defining the specific decision problem is an important early structuring activity. Having a prototypical set of problems types aids

this activity. Third, decision problems differentiate themselves based on the pattern of decisions in the model. Fourth, the degree of uncertainty in the decision problem drives the structuring process. In addition, an understanding of how the decision problem elements relate to one another is useful for developing a knowledge reuse capability and in understanding how to do decision graph modeling.

3.2 Context

Understanding the decision context is considered key to decision-making (Brézillon and Brézillon 2008; Clemen and Reilly 2014) and is a part of the decision problem structuring process (Von Winterfeldt and Edwards 1986; Watson and Buede 1987). But the definition of decision context varies significantly. The Oxford English Dictionary defines context as "The circumstances that form the setting for an event, statement, or idea, and in terms of which it can be fully understood" (OED 2017). This is a very broad and flexible definition. Within the decision-making literature, one finds two primary interpretations.

The dominant view of decision context is that it is closely related to situation or environment (Clemen 1997). Within naturalistic decision-making, context is very strongly identified with situation. Context (as situation) has a leading role, in that a key decision is to decide what the specific context is. Once the context is identified, fashioning an appropriate response tends to be rapid, as described in models such as the recognition-primed decision model (G. A. Klein, Calderwood, and Clinton-Cirocco 1988) and situational awareness for decision-making (Endsley 1995).

On the other hand, some researchers in strategic organizational decision-making adopted a very expansive view of context. Here, it encompasses decision-maker characteristics and background, decision problem characteristics, organization structure and characteristics, and the external environment (Dean, Sharfman, and Ford 1991; Sharfman and Dean 1997; Shepherd and Rudd 2014).

The artificial intelligence community has explored context as well, often as part of developing decision-making applications. An early summary is in Brézillon (1999). Much of the artificial intelligence literature focuses on the situation aspect of context. For example, in designing context-aware systems, Dey states that "Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves" (Dey 2001). In recent years, mobile and pervasive computing has spurred significant context research (Bettini et al. 2010). The current state-of-the art in context-aware system development is in Alegre, Augusto, and Clark (2016).

Öztürk and Aamodt (1997) provide an interesting and useful set of insights. They divide context into internal (decision-maker specific) and external (situation) elements. Ozturk (1999) identifies two specific features of context that are relevant to decision-making:

- Context focuses attention on specific problem aspects
- Context changes shift the focus of attention.

This has been pithily summarized by Gundersen's observation that "what context is changes with its context" (Gundersen 2013). Recommender systems include user preferences, usually in the form of ratings, to identify choices that share similar characteristics (Adomavicius and Tuzhilin 2015). In their work on context in information fusion systems, Snidaro et al. split context into internal and external factors, and identified a mutual interaction between the two (Snidaro, García, and Llinas 2015).

Two key observations about context shape the decision structuring / knowledge reuse process. First, the full context is not often known by the decision marker at the time a need for a decision is triggered. The triggering need may simply be a sense that something is wrong or that an improvement is needed. Therefore, a significant part of the structuring process is to identify what the relevant context is, how much of it the decision-maker already knows, and what must be developed during the structuring process. For some decision problems, very little is known about the relevant context and that the primary decision problem is to establish what the context is. Second, context may be dynamic. The degree of time stability of context information can drive the decision process, including its stability from decision process start to decision point, and from decision point to outcomes. The degree of stability can affect both the decision choices being considered and the available time to execute the decision process.

There have been several approaches to formally modeling context in support of decision processes. This includes conceptual context graphs (Brézillon and Brézillon 2008), Contextors (Coutaz and Rey 2002), HyCoRE (Beamon and Kumar 2010), and Context Recognition Network (CRN) Toolbox (Bannach, Amft, and Lukowicz 2008).

These approaches collectively are limited to situation context only (no decision-maker element considerations) and in some cases are application-domain specific (e.g. wearable IT) or have a limited ability to manage context uncertainty. Examining the products of a number of these approaches, it became obvious that BN/DGs were inherently capable of perform context modeling of situations.

3.2.1 The Decision Context Model

Odd Gundersen built a context model (Figure 4) using the internal / external insights from Öztürk and Aamodt (Gundersen 2013). He considers context to have both an internal and an external aspect vis-à-vis a decision-maker, and he casts it within the decision-maker's need to assess the situation within which the decision-maker must operate. The external elements all relate to the situation in which the decision problem is embedded. The internal elements are the key decision-maker factors that shape how the decision-maker views the decision problem.

Gunderson uses Endsley's situation awareness model to define the elements of a situation. Endsley developed a widely used model of how a decision-maker understands a situation. The model includes the perception of the elements of a situation, the comprehension of what the elements and their relationships mean to a decision-maker, and the

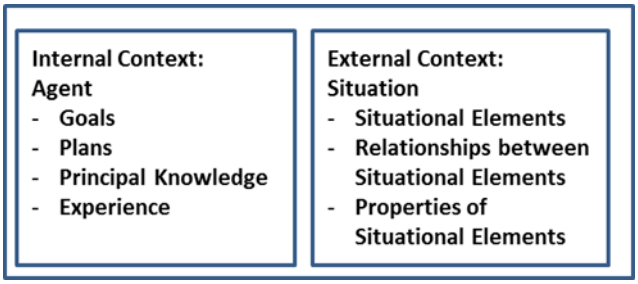


Figure 4: Elements of Gunderson's context model

projection of the future states of a situation (Endsley 1995). Gunderson used situation semantics (Devlin 2008) to provide a modeling framework for the elements of a situation. Finally, Ozturk and Aamodt's knowledge-level context model provides, within a decision-making environment, a basis for integrating the decision-maker's elements with the situational elements (Öztürk and Aamodt 1997).

The Decision Context Model (DCM) builds upon Gundersen's work. It starts with the key insights he incorporates:

- Decision-making context arises out of the interactions between a specific set of external elements (the situation) that a decision-maker faces and the decision-maker's internal elements
- Context focuses attention on aspects of the situation and internal elements, not all of them
- Context changes as situation or decision-maker internal factors change.

The model is extended in several dimensions, resulting in the model shown in Figure 5. A key modification is to recognize multiple parties in a decision context, called stakeholders. Stakeholders fall into one of two categories: decision-maker and affected party. Decision-maker is the party (or parties) that make the decision. Affected party represents individuals or groups affected by a decision, who have some input into the decision-making process, but do not make the decision. There may be several different affected parties, each with their own internal context factors The DCM does not define the structure of a decision-maker. This can be a single individual, a group with a decision-making rule (e.g. majority voting) or a multi-party negotiation process. The research

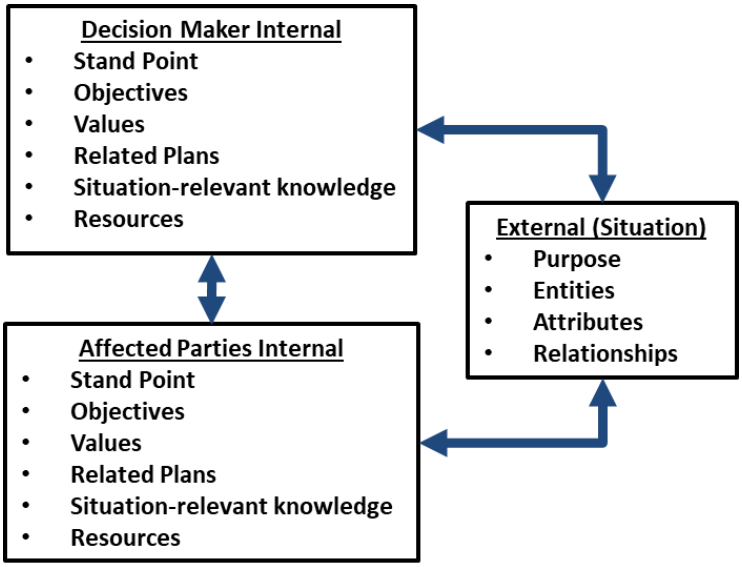


Figure 5: Elements of the Decision Context Model (DCM)

below assumes a unary decision-maker. The Decision Context Model structure can support future research on the different effects of multiple-decision-maker structures.

3.2.1.1 External (Situation) Elements

The external elements are part of a single situation, applicable to all the stakeholders. The DCM recognizes that different stakeholders can have a different view of a situation. This difference of situation knowledge is included as a stakeholder internal element (see next section below).

To model the situation, the DCM uses Sowa's entity ontology in lieu of Endsley's model. Endsley's model has a process orientation, how situational awareness is achieved, rather than a structural orientation, what a situation is. Sowa's ontology structure fits nicely with the MEDG knowledge representation scheme. *Entities* can either be anything existing in the physical world or be abstract concepts. They may be either time-stable (objects / concepts that are recognizably the same or similar over some time scale) or

time-varying (such as an event or process). Most important, entities can be composed of other entities. Sowa uses entity relationship at three distinct levels as a differentiator. The first is in isolation, where the entity in question is treated as having no internal structure. One looks only at entity's *attributes,* or inherent characteristics. *Relationships* between entities are addressed in the second and third levels. The second level focuses on specific relationships among a small set of entities, which is ideal for a Template implementation. The third level is the situation level, a set of multiple entities (either objects or events) with multiple attributes interacting in multiple relationships (Sowa 2000).

A situation has a *purpose*, established by some agent that "determines why the interaction of entities in the situation are significant." That agent often is not the decision-maker, and does not even have to be human (Sowa 2000). Purpose ties the entities into a coherent whole and is the key characteristic that distinguishes one situation from another. One comprehends a situation when one determines its purpose. As an example, consider a military commander who must assess and react to a significant increase in military activity in a neighboring adversary country. Knowing whether the purpose of the activity is to attack or defend against another state, suppress domestic unrest, conduct a large-scale exercise, or execute a coup d'état will be a key factor guiding the decision-maker's actions. In this case, the purposive agent is the military commander of the involved adversary forces. Comprehending a situation also provides one an understanding of what the elements of the situation should be and allows one to project how a situation could unfold.

3.2.1.2 Internal Elements

For each stakeholder, there are six internal elements that affect the decision-making process.

Standpoint summarizes the decision-maker's overall perspective, analogously as purpose captures an overall situation. Standpoint captures the stakeholder's position and viewpoint within the problem space (Schum 1994). The standpoint of a corporate CEO is different than the standpoint of a government regulator. It helps to define the decision-maker's values and objectives (as noted in the adage "where you stand depends on where you sit"), but other influences can affect those for a specific stakeholder.

Objectives are the outcomes a decision-maker expects to achieve by making the decision. These are also called ends (Von Winterfeldt and Edwards 1986) or fundamental objectives (Keeney 1992).

Values represent fundamental principles and priorities the decision-maker will abide by in making decisions. Values shape the objectives, delineate acceptable alternatives, and guide the relative weighting between multiple objectives (Thomsen 2004; Jones 2014). The importance of both stakeholder's objectives and values to a decision problem is widely discussed in the decision making literature (e.g., Watson and Buede 1987; Keeney 1992; Grünig and Kühn 2013; Clemen and Reilly 2014).

Related Plans are other on-going or planned activities of interest to the stakeholder that could be affected by one or more decision options (Öztürk and Aamodt 1997). While the effect of a decision alternative on these other activities may be incorporated

into the objectives, it is important to identify that decision-making includes understanding what outside of the immediate decision problem is affected by the decision.

Situation-relevant knowledge includes everything a stakeholder understands about a decision problem, and the knowledge background through which a situation is interpreted. It merges Gundersen's Principal Knowledge and Experience. As noted in section 3.2.1.1, there is only one situation. But each stakeholder's perception of the situation depends on the stakeholder's knowledge about the situation. Differences in knowledge, along with differences in values, mean different stakeholders may focus on different situation elements.

Resources are assets available to the stakeholder that shape the possible decision options. These resources include physical assets, knowledge, time and financial assets. Resources affect the range of viable alternatives and may be items requiring tracking in a decision model.

3.2.2 Context Differentiator Impact on Decision Graph Modeling and Knowledge Reuse

Decision context identifies four major differences between decision problems, as shown in Table 1. These differences become important items to identify in a knowledge reuse tool. In addition, they drive a decision graph capability requirement for modeling context variation.

The first difference is problem focus - whether the decision context is dominated by the decision-maker's internal elements or the situation. This corresponds roughly to Keeney's division of problem structuring approaches into value-focused versus

Table 1: Context-derived decision problem differences

Focus	Objectives / Values		Situation	
Context variation	Single	Some	Major	
Context dynamics	Static - no significant change until decision implementation	Changing by decision implementation	Rapidly changing – by planned decision time	
Decision-maker / stakeholder structure	Single decision maker, limited stakeholders	Group - formal procedure (e.g. majority voting)	Single / group decision making - significant stakeholder input	Negotiated decisions

alternative-focused approaches (Keeney 1992). He argues that for many decision problems, rather than focusing on the specific decision situation and decision alternatives, it is more appropriate to begin with a thorough understanding of the decision-maker's objectives and values. Once those are understood, then the decision-maker is in a better position to frame the decision, or multiple decisions if this enhanced understanding identifies that there are several areas in which decision problems exist.

On the other hand, a significant number of decision problems are situation-driven, and in some cases, recognizing the correct situation is the primary decision-making problem. As noted by Klein and other naturalistic decision-making researchers, once a decision-maker is able to correctly identify a situation, the appropriate decision alternative is often readily identifiable (G. Klein, Calderwood, and Clinton-Cirocco 2010). If the decision context is significantly time-constrained, immediately implementing a viable option can be the most rational decision a decision-maker can make. Situation and objective / values are both important to all decision problems. But one may be well understood while

determining the other is a key structuring activity. Context focus identifies a key knowledge reuse element.

The second context difference is the degree of context variation in a specific problem. If the problem is to select a restaurant, the selection criteria and the relative weights between them can vary significantly between a romantic dinner and a business dinner. The context can also be changed by information learned while executing the decision-making model. Decision problems with a high degree of context variation require more extensive modeling, and benefit from modeling features that enable a specific context variant to be modeled as appropriate. The degree of context variation is also a knowledge reuse content element.

The third way decision context difference is context dynamics. This is the degree of change in the decision context over time. The issue here is not the effects of context learning, but whether what is known / learned about the context is outdated by significant changes in the decision context. Highly dynamic decision contexts shorten the available decision-making time and require a more adaptive problem structuring process. There are two important intervals. The first is the degree of change expected prior to making the decision. The second is the degree of change expected between the time decision structuring is initiated to the time a decision alternative can be implemented. Highly dynamic decision context requires a shorter decision cycle, and this knowledge needs to be captured in a decision knowledge reuse tool.

The fourth decision context difference is in stakeholder considerations. Decision problems with significant stakeholder divergence require a different problem structuring

approach. This places significant emphasis on understanding the stakeholder environment. The stakeholders include both the decision-maker and affected parties that have an influence on the decision process. A rough division of stakeholder organization is to divide it into four categories. The first is where there is a single decision-maker and a limited or no role for any affected parties. The second is where the decision-maker is a group that makes decisions via some formal rule, such as majority voting. In this case, all affected parties are represented in the group. The third is where there is a decision-maker (either single or formal group) where affected parties can provide inputs to the decision process. The final option is where there is no formal decision-maker. Rather, the decision is a negotiated agreement between affected parties. A decision problem knowledge reuse tool should capture key decision-making structure information and problem class specific information on working with the stakeholders.

3.3 Decision Problem Type

Framing a decision is an important early decision action. It defines a precise statement of the decision to be made. Framing is aided by selecting the appropriate decision problem type. This is the basic orientation of the decision problem. Nickols suggests there are three basic types. Consider the decision framing choices of an owner of a failed industrial process machine. She could repair the machine, which is a restoration problem type. Here, one primarily wants to return to a previous acceptable state. The second is to replace the machine with one that offers major improvements over the existing machine. This is a refinement problem type. The third would be to redesign the workflow to eliminate the need for the machine. Designing out the need for the machine or combining

functions with a new type of machine would be the outcome here. This is a creative problem type, focused on creating substantial, major changes that result in dramatically improved results (Nickols 2012).

There are at least two additional problem types. The first is that some decision problems involve determining a state. Determination problems are widespread. For example, a criminal jury must determine whether the facts and law presented to them sufficiently support a finding of guilty. Many information fusion problems have this requirement to determine a state. The determination often is probabilistic, that there is a high likelihood that x is the case, but for decision purposes, it will be treated as if x is actually the case. This occurs from a tradeoff that one is better off treating x as the case even if it is not, rather than ignoring it and accepting the risk that it is the case. The second type is when a decision-maker must respond to a determination from another source. In some cases, a single predetermined choice is appropriate. In other cases, a decision-maker must select from a set of choices. For example, consider an aircraft pilot who learns that the weather at the planned destination is no longer acceptable and the plane must now divert to an alternate airport. If there are several acceptable alternatives, the pilot must decide. Table 2, modified from Nickols (2012), identifies the key characteristics of each basic decision problem type. A problem type has four key characteristics. First is the general objective, guiding and focusing the decision-making actions. The starting point specifies where the structuring process begins. The focal point identifies what the primary emphasis of the structuring process is. Core processes are the primary processes used to define the contents of the decision model. All of these are content for a knowledge reuse tool.

Table 2: Decision problem type characteristics and their structuring process effects

Characteristic	Determination	Response	Restoration	Refinement	Creation
Objective	Establish specific state	React to state change	Return something to a predefined state	Improve upon a predefined state	Create new arrangement much superior to present
Starting Point	Context / Background	State change effect	Triggering condition (e.g. symptoms)	Existing system / structure / arrangement	Required / desired end-state
Focal Point	Information fusion	Options	Causes and corrective measures	Constraints, restraints, and modifications	Required structure to achieve end state
Core Processes	Information collection / evaluation	Option analysis; determination acceptance	Trouble shooting; cause assessment	Analysis – what are limits	Design – what are possibilities

3.4 Decision Patterns

The third differentiator is the decision pattern applicable to the decision problem category. A decision problem may have multiple decisions that are chained together in various ways. Figure 6 below gives a decision pattern hierarchy. The top-level branch is decision instances. A decision instance is a set of one or more decisions to be made. If there is more than one decision in an instance, there is no learning between decisions. That is, there is no information node between two or more decisions that is triggered by one decision and the information passed to a second decision. For instance, a doctor must decide on whether to order any of three possible tests. If the doctor decides on all three prior to receiving any results, this is a single decision instance. If the doctor decides them one at a time and awaits results before deciding the next, there are three decision instances. In that case, the solution process that establishes the alternative values may be complex.

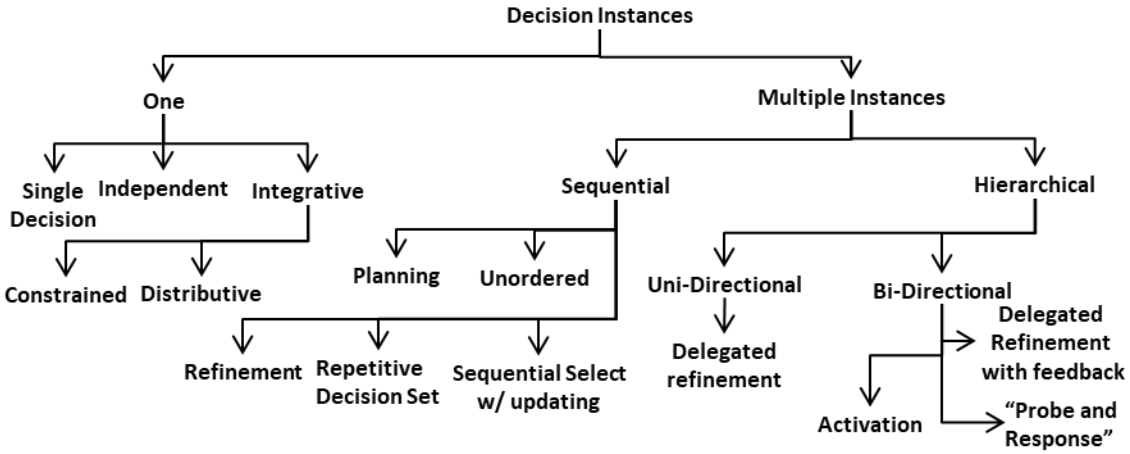


Figure 6: Hierarchy of decision patterns

The figure identifies thirteen patterns. Four are single instance patterns, and nine are multiple instance patterns, subdivided into sequential and hierarchical patterns. The simplest pattern is a single decision. An example is a jury determining guilt or innocence in a trial. Observe that while the decision pattern is simple, there can be a complex information fusion element that supports that single decision.

Slightly more complex is when there are multiple independent decisions that need to be made. This pattern occurs when there is a common source of information supporting multiple decisions, but the decisions do not affect one another. For instance, a breeder discovers that one of her horses has a genetic defect. There is then a set of decisions about whether each living ancestor, sibling, and cousin should continue to be bred. The decision on each is based on the risk of a foal receiving the recessive gene from both parents.

The third single instance pattern is the integrative pattern. Here, there is a set of decisions that are made jointly (e.g. there is no learning between the decisions in the set),

but collectively have constraints that must be met. Integrative patterns all exhibit functional asymmetry. The pattern has two distinct subclasses. The first is the constrained decision pattern. One decision alternative is not compatible with some of the alternatives of another decision that must be made simultaneously. For instance, a fighter aircraft facing an enemy aircraft has multiple decisions to make, such as making a defensive maneuver and launching a missile. But some defensive maneuvers are violent; a missile cannot be launched if they are chosen. The second integrative decision subclass is the distributive pattern. One is distributing a resource among the possible decision alternatives, and the total set of alternatives cannot be selected. For instance, a portfolio decision problem or any resource allocation problem requires that the sum of a resource type applied to possible choices does not exceed some limit. Sequencing or scheduling problems also have this pattern if all decisions must be made up front. If learning is allowed between some or all decisions, this becomes an unordered decision.

For multiple decision instances, there are two basic approaches: sequential and hierarchical. This follows from Saaty and Shih's (2009) distinction between network and hierarchical structures of influence between network elements. If the flow of influence is among elements without a logical grouping of a hierarchy of elements, a sequential pattern exists. One pattern is the planning pattern, where there is a defined ordering of the decisions. An example is an information gathering decision followed by an action decision, which uses the result of the information gathering as part of its decision. While the pattern is relatively simple, multiple items may be connected to make complex models. In addition, the structure may have functional or structural asymmetry, if information gath-

ering results activate different decisions. For example, a model may have a contingency track of decisions and actions activated only if some distinct state is detected. In the unordered pattern, the ordering of the decisions is itself part of the decision problem. This pattern exhibits order asymmetry. Diagnostic / repair problems are an example of the unordered pattern. The sequencing of the diagnostic activities is the primary decision problem here. The key characteristic here is that between every decision there is learning and under some conditions, the need to continue making decisions is not required.

A sequential refinement problem differs from most other patterns in that the decision model is rebuilt several times, adding detail and complexity as it is rebuilt. The decisions and the scope of the problem also become more focused. The facility siting problem is an example of this decision pattern. Here, there is a progressive down select of candidate locations, with an initial broad set of criteria to guide the initial down select, followed by refining the criteria to guide the next round of down selection. The repetitive decision pattern is any problem where the structure of the decision is repeated multiple times. A partially observable Markov decision problem is an example of this pattern. The final sequential pattern is the sequential select with updating pattern. Here there is a sequencing activity in which there is learning that can cause a restructuring of the sequence. This is an extension of the ordering problem described above. There are ordering decisions to be made, but the order may be reset based on new information gained along the way. This can include repeating actions that were done in earlier decisions. Real-time problems in contingency or medical management can follow this pattern, where new information forces the decision-maker to restructure a sequence of actions.

In the hierarchical patterns, decisions are made at different levels in a hierarchy and can influence other levels. A key differentiator for the hierarchical design pattern is each level has its own value structure for determining its decisions. The only path from decisions at one level to another level go through decisions at the other level. This form makes these patterns structurally asymmetric. The hierarchical nature often comes by segregating decisions by their scope and the time required to make them. A common form is organizational level of differences. Top level management may provide broad strategic direction, with a requirement that lower levels determine how to implement the direction at their level. Hierarchical decisions differentiate based on whether the modeled information flow is uni-directional or bi-directional. A delegated refinement is the common form of a uni-directional hierarchical problem. In this pattern, there is a higher level that decides some generally desired outcome. Implementation is made by lower levels making decisions about alternative approaches that implement the direction. There may be multiple levels of delegations in the model. Delegated refinement is widely used in hierarchical organizations. The bi-directional hierarchical patterns allow feedback from the lower to the higher levels. This feedback can cause the higher level to reevaluate a previous decision. The delegated refinement with feedback pattern is used whenever one has hierarchically divided control responsibilities. Status information from a lower level may signal a need to adjust the overall strategy at a higher level. While delegated refinement focuses on control, the activation pattern focuses on information. It models problems where the decision-maker controls several information collection assets, where each asset has its own local control responsibilities for its operations. Finally, probe and re-

sponse is an information seeking pattern used when one has great uncertainty about a situation. Like the sequential refinement pattern, it differs from most other patterns by having a set of models that are created by updating previous models. Here, one creates a revised model of the situation one is trying to understand and uses it to conduct an experiment or information development action to test the fit of the model to the observed features of the situation. The probe and response pattern is the control structure for the revision process.

The primary impact these patterns have on modeling requirements is that many patterns exhibit one or more forms of asymmetry. Many decision graph implementations require that the modeler address the asymmetry as part of the modeling process. Incorporating automated symmetry handling frees the modeler from this task and its associated and its associated errors. For knowledge reuse, identifying the applicable patterns for a decision problem class provides useful information.

3.5 Magnitude of Uncertainty

Uncertainty is a barrier to decision-making. Under extreme uncertainty, decision-makers often refuse to make decision except in dire circumstance (Lipshitz and Strauss 1997). Decision uncertainty is much broader than the existence of chance outcomes that affect the consequences of a decision alternative. It includes uncertainty (including complete ignorance) about the range of available alternatives, their attributes and consequences, and the value of those consequences to the decision-maker (Berkeley and Humphreys 1982). One can have uncertainty about any element within the Decision Problem Elements Schema (Section 3.6, page 59), including relevant decision context. A decision-

maker desires to eliminate as many uncertainties as possible and to understand the range, scope and impact of any remaining uncertainties. How one addresses uncertainty in a decision model depends on its magnitude. Integrating work by Scherpereel (2006), Snowden and Boone (2007), Stirling and Scoones (2009), and Kwakkel, Walker, and Haasnoot (2016), one can categorize uncertainty into four magnitude levels (Table 3).

The impact of increasing magnitude of uncertainty is threefold. First, the structuring process is partially driven by the degree and nature of uncertainty at the beginning of the process. The structuring process includes uncertainty reduction / elimination activities. If significant second-order uncertainties or ignorance exist, the structuring process tends to focus on them. Second, the uncertainty remaining at the end of the process affects the resulting model. Stirling and Scoones (2009) found that as the degree of uncertainty in either the likelihood of events or knowledge of outcomes increased, the structuring activities and decision-making processes changed. Specifically, alternatives are created and evaluated based on their ability to address the uncertainty. Decision knowledge reuse tools need to include common uncertainties and approaches to addressing them.

Table 3: Levels of uncertainty

Magnitude level	Degree of uncertainty
Certainty	No significant uncertainties
1st order probabilistic uncertainty	Defined uncertainty (point) about likelihood of future events. Everything else known with certainty
2nd order	Bounded uncertainty (range) on one or more elements of decision model. Includes event likelihoods, magnitude of consequences, levels of alternative attributes (including achievability), and achievable criteria values
Ignorance	Lack of information to make an informed judgement of the range of uncertainty on one or more elements of the decision model

The larger uncertainty, business and risk management literature has some useful insights that should be incorporated (see e.g. Han, Klein, and Arora 2011; Miller and Shamsie 1999; W. D. Rowe 1994). The third impact is on decision modeling. Decision graphs are designed to address problems with certainty or first-order probabilistic uncertainty. One can use sensitivity analysis to understand the range of effects of second-order uncertainty. Incorporating such capabilities into a modeling toolset would be useful.

3.6 Decision Problem Element Schema

As part of this dissertation research, it was useful to develop a comprehensive top-level schema of the decision-making elements. This schema defines the elements of a decision problem and the major relationships between those elements. It incorporates the differentiator elements previously discussed. This schema has two objectives:

- Identify how the elements relate to each other in the structuring process, with a focus on the effects on knowledge reuse
- Identify what decision problem elements could be included in a decision model, and what the major relationships are between them.

In addition to literature review on decision analysis and decision structuring discussed in Chapter Two, I also explored the literature on decision problem ontologies. While this schema is at a more general level than an ontology, the decision ontology literature did identify concepts appropriate to this research. There have been only a few efforts to specifically define a decision ontology. Efforts include work in the R&D project selection (Liu, Tian, and Ma 2004), software architecture (Kruchten 2004), smart home support (Latfi, Lefebvre, and Descheneaux 2007), chemical engineering domain (Theis-

sen and Marquardt 2008), and information systems engineering (Kornyshova and Deneckère 2010). Of these, Latfi, Lefebvre, and Descheneaux (2007) emphasize the integration of a rather simple decision ontology with others that collectively describe the domain of interest. Theissen and Marquardt (2008) emphasize the interaction between a decision ontology and a decision-making process ontology, while Kornyshova and Deneckère (2010) provide a comprehensive class-oriented structure of decision elements.

The Decision Problem Element Schema is organized following the major structuring activities of establishing the context, framing the problem, and developing the specific decision elements. It is shown in Figure 7. The first level is the Context Level, comprised of the Stakeholder, the Situation and the Stakeholder Factors classes (elements

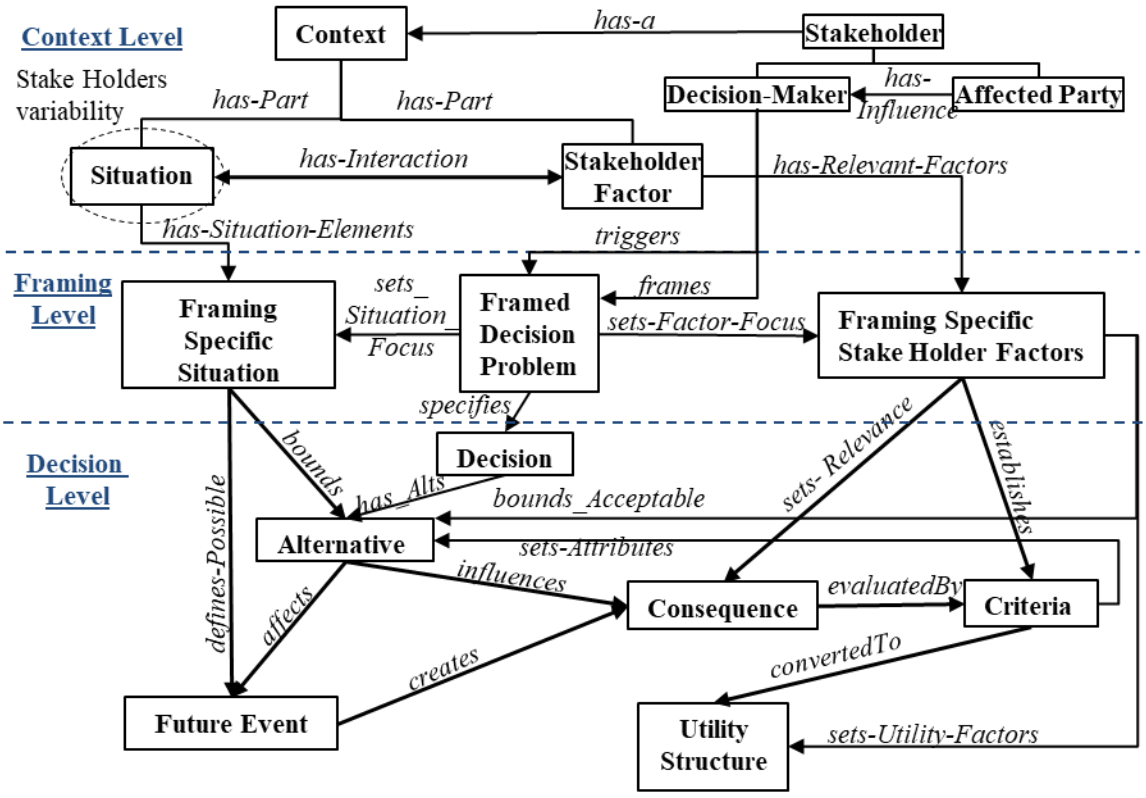


Figure 7: Decision Problem Element Schema

defined below). A context may cause the decision-make to trigger one or more Framed Decision Problems at the Framing Level. A framed decision problem has a basic problem type orientation, which sets a focus that selects the relevant elements of the Situation and Stakeholder Factors for further analysis.

It is possible for a context to trigger multiple framed decision problems. If so, they tend to be loosely coupled, with a different model for each. At the Decision Level, a framed decision problem may require multiple decisions. An example would be a doctor deciding upon various diagnostic tests prior to deciding upon a treatment regime. Each decision has its own elements, although they usually overlap between decisions. At this level, there is normally a close coupling between decisions. At each level, the schema identifies the element classes, major relationships among the elements, and key class attributes.

3.6.1 Context Level Elements

At the context level, there are two major classes: *Stakeholder* and *Context*, with each having two subclasses. *Stakeholder r*epresents all parties that have an interest in the decision, and that have some voice in making the decision. Each Stakeholder entity has a key attribute, *has-Standpoint* (see Section 3.2.1.2, page 40). The Stakeholder class has two subclasses. One is *Decision-Maker,* which is the entity(ies) that make the decision. The class has a key attribute, *has-Structure*, characterizing the decision-making structure for this decision problem. It may be a single individual, a collection of individuals that act as a unified entity for decision making purposes, or a negotiating group. The second subclass is *Affected Party*, which includes all entities that have some voice in the deci-

sion-making process, but do not actually make the decision. Each Affected Party entity has a *has-Influence* relationship with the decision-maker, reflecting the level of influence that an Affected Party has on the Decision-maker.

Stakeholder has a *has-a* relationship with *Context*, which provides the lens through which a stakeholder views and defines a decision problem. *Context* has two subclasses: *Situation* and *Stakeholder-Factor*. *Situation* delineates the elements of the world, that capture the aspects of the world relevant to a Stakeholder. The focus of a situation is on the time sequence of changes among the entities. A situation has a key attribute called *purpose*. This is the underlying reason or condition that differentiates situations. Many fusion models are situation models. It is expected that for decision problems with a significant situation, a distinct domain-specific model will be developed, using processes not described here. For decision making, the decision situation in a decision model consists of those elements of the situation model that represent the information that a decision-maker should learn during the decision-making process.

Situation is common to all stakeholders; however, certain elements of the situation may not be relevant to a specific stakeholder. *Situation* has a set of *has-Interaction* relationships with *Stakeholder-Factor*. This is a bi-directional relationship (not necessarily symmetric) that signals that certain situational elements and states trigger certain Stakeholder factors, and that specific Stakeholder factors drive the stakeholder to monitor specific situation elements.

For each Stakeholder, the *Stakeholder-Factor* subclass captures key elements of the Stakeholder as it relates to a specific context. It has five subclasses:

- *Decision-Objectives*, which are the ultimate outcomes a stakeholder expects the decision to support. Its attributes are the states of the objective and the desired outcome
- *Values*, the guiding principles and considerations that shape a stakeholder's preferences
- *Related-Plans*, which are ongoing and planned activities that can be affected by or affect a decision
- *Situation-Relevant-Knowledge*, a set of facts regarding the knowledge each stakeholder has regarding the situation
- *Resources*, a description of the relevant resources available to a stakeholder.

As stated above, *Stakeholder-Factor* has a major bi-directional relationship with *Situation* via the *has-interaction* relationship.

There are four key relationships between the Context Level and the Framed Decision Problem Level. The first is *triggers*, between the *Decision-Maker* and the *Framed-Decision-Problem*. This signals the decision-maker's judgment that a need exists for a decision. This triggers the decision structuring process with initial focus on framing the problem and identifying the relevant aspects of context. The *frames* relationship highlights the decision maker's role in specifying what exactly the problem is that will be analyzed. In addition, there are the *has-Situation-Element and has-Relevant-Factors* relationships, described in the next section.

3.6.2 Framing Level

The Framing Level has three classes: *Framed-Decision-Problem*, *Framing-Specific-Situation*, and *Framing-Specific-Stakeholder-Factors*. The Decision-maker triggers and frames a *Framed Decision Problem*, that describes the specific decision problem to be addressed. In addition, the decision-maker identifies the boundary conditions (scope, range of effort, excluded or mandatory inclusion factors), captured via a set of *Conditions* attributes. In addition to the *frames* and *triggers* relationship with the decision-maker, the *Framed-Decision-Problem* class has two important relationships. The *sets-Situation-Focus* relationship with the *Framing-Specific-Situation* class works with the *has-Situation-Elements* relationship between *Situation* and *Framing-Specific-Situation* to define what situation factors are relevant to the framed problem. For the structuring process, this also defines what additional refinement to the situation model are required. The *Framing-Specific-Situation* class has selected elements of the situation, determined relevant to the framed decision problem.

Similarly, the *sets-Factor-Focus* relationship with the *Framing-Specific-Stakeholder-Factors* class works with the *has-Relevant-Factors* relationship between *Stakeholder-Factor* class and *Framing-Specific-Stakeholder-Factors* class to define the stakeholder factors relevant to the specific framed decision problem. *Framing-Specific-Stakeholder-Factors* class is a subset of all stakeholder factors that the framed decision focus identifies as relevant to the framed decision problem. These two classes have a major influence on the Decision Level elements.

3.6.3 Decision Level

A framed decision problem may contain more than one decision, so the first class at the Decision Level is the *Decision* class. The s*pecifies* relationship from *Framed-Decision-Problem* links a framed decision problem to the specific decisions at the decision level. The way a specific decision problem is framed also provides boundaries on what alternatives are acceptable. For example, if the problem is a broken machine, and the decision frame is to repair it, the alternative space is different than if the decision frame is to replace it.

Alternative is the class of all decision options under consideration for a specific decision. Each entity in Alternative has a set of attributes that define each entity's major characteristics, as applicable to the decision problem. Three major enabling relationships affect what alternatives are viable. The *specifies* relationship limits selected alternatives to those that match the decision needs. The *bounds* relationship with the *Framing - Specific-Situation* class bounds specific alternatives to what is relevant to situation, while the *bounds-Acceptable* relationship with the *Framing-Specific-Stakeholder-Factors* class bounds acceptable alternatives given the stakeholder values, resources and related plans. In addition, the *Criteria* class identifies the required alternative attributes (*sets-Attributes*) to properly evaluate the alternatives.

Alternative has two major enabling relationships that influence other elements. First, *affects* identifies that specific alternatives have an influence on what instances of future events are possible. Second, specific alternatives map to specific possible consequences via the *influences* relationship.

Future-Event are those situation entities whose states may change or be revealed at some time in the future. In addition, an alternative may trigger possible future events for specific alternatives (e.g. major nuclear accident if a nuclear option is chosen for an electrical power plant. The *Future-Event* class influences, via *can-cause* relationship, both the types of consequences that may occur and their magnitudes. *Consequence* are the outcomes relevant to a stakeholder that occur from the interplay between alternatives and future events. The *sets-Relevance* relationship from the *Framing-Specific-Stakeholder-Factors* class establishes which potential consequences are significant to a stakeholder and must be considered in the decision.

The *Criteria* class has abstract entities that define how a consequence will be evaluated. The criteria are derived from the Stakeholder factors. This is captured in the schema by the *establishes* relationship from the *Framing-Specific-Stakeholder-Factors.*

Criteria has a *convertedTo* relationship with *Utility Structure* class. *Utility Structure* transform the specific criteria levels to a common scale that allows trade-offs between criteria. Each utility entity has the following attributes:

- Weight relative to other utilities
- Criteria inputs
- Conversion method from criteria to utility. This conversion method includes risk attitude, constraints (e.g. min or max required criterion levels), and interactions between criteria inputs (which may be nonlinear).

In UnBBayes-MEDG, multiple utility entities have a linear additivity relationship with each other. Nonlinear interactions and criterial dependencies are captured within a specific utility entity.

The above elements can be formed into a core decision graph model fragment (Figure 8 below). It represents the basic information flow from a specific decision through to the utility node(s) that assess the utility of each alternative. This fragment can be used for any decision problem that can be modeled by a decision graph. Of the seven types of nodes shown, two are mandatory in any decision graph: the decision node with alternatives (created using the *Decision* and *Alternatives* class information) and the utility node (derived from the *Utility-Structure* class). Without either of these nodes, one does not have a decision graph. A specific decision will have only one node. A modeler may use as many utility nodes as useful to the modeling and understand the problem[3]. Whether one uses the remaining five node types depends on the decision problem and the

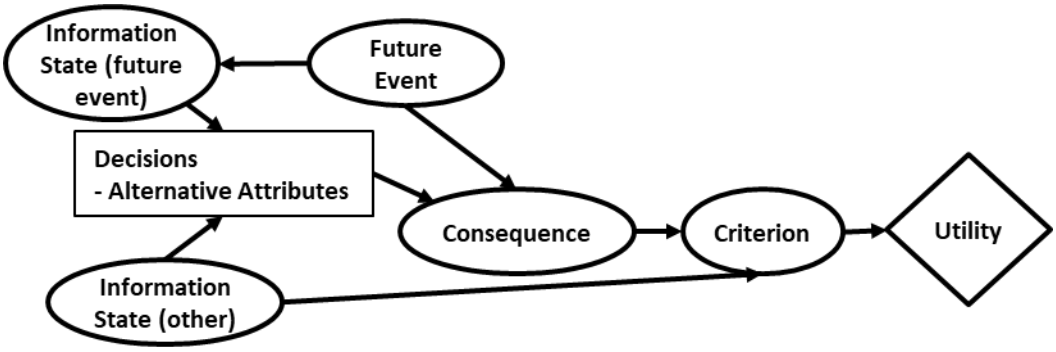


Figure 8: Decision graph core model fragment

[3] Subject to a standard decision graph convention that multiple utility nodes are combined assuming additive independence between utility elements.

modeler's need for clarity. Note that the Criterion node can be embedded into the utility node, but this should be done only for simple problems. For more complex problems, it is useful to decompose the problem elements to specifically identify the different criteria. One can also embed the consequence node type information into a set of criteria nodes. A future events node is used only when there is the probabilistic future event that can affect the outcome of a decision. While the figure shows only one node of each type, one may use as many as is useful. In addition, any of the non-utility node types shown may be a decision graph fragment that models a complex set of interactions that create a future event or consequence. One may also use criterion and utility nodes to model an objective hierarchy decomposition.

Either of the two information state nodes exist only when there is prior knowledge that will be made available to the decision-maker before making the decision. Information state nodes that provide information about a future event are most often modeled using the parent-child configuration shown in Figure 8. Other information state nodes may connect to other nodes in the model. Like the other node types, the information state node may be a complex set of nodes that develop the information provided to the decision-maker.

3.7 Conclusion

This chapter explored the key decision characteristics that establish needs for first-order expressive decision graph capabilities and decision structuring support tools. This advanced the decision modeling state-of-the art by identifying and refining a set of four decision problem differentiators - context, problem type, decision pattern, and uncer-

tainty - that establish first-order expressive decision graph capability needs and relevant content for a knowledge reuse tool. It also created an schema of decision modeling elements focused on supporting first-order expressive modeling. This schema defines the relationships between the decision elements, which is useful for supporting decision graph modeling.

CHAPTER 4
LOCAL PROBABILITY DISTRIBUTION LANGUAGE REQUIREMENTS

Early research identified that the local probability distribution (LPD) script language in the UnBBayes-MEDG implementation could not address parent configurations where one or more parents have entities as states, rather than a Boolean or a fixed categorical list of possible states. Without this capability, UnBBayes-MEDG cannot model problems where the possible decision alternatives can vary from instantiation to instantiation. As discussed in section 2.5.3 above, this was a result of a research gap on the range of requirements for a LPD creation language supporting first-order expressive probabilistic modeling. This chapter describes the approach taken to fill that gap and its results. The research developed a new approach to explore how different types of parent RVs interact with each other structurally and within the LPD to define the child RV's probabilities. Because the focus is on LPDs, the results are applicable to both Bayesian networks and decision graphs.

The results are described at two levels of implementation specificity. The higher level is applicable to any first-order expressive probabilistic modeling framework whose models can be expressed as grounded Bayesian networks or decision graphs. The results at this level are described behaviorally, defining actions that must be done in an LPD creation language. The lower level is for implementations of Multi-Entity Bayesian Net-

works and Multi-Entity Decision Graphs. They take advantage of certain features of the MEBN/MEDG approach to identify capabilities implementations can have. While focused on the UnBBayes implementation, the concepts are readily adaptable to other MEBN / MEDG implementations.

This chapter first describes the key concepts of the existing UnBBayes-MEDG LPD script language, providing a baseline of LPD behaviors. Next, it outlines the dependency modeling approach, describing why the dependency modeling approach is both comprehensive and applicable to modeling a wide range of domain. Details are found in Appendix A. Dependency modeling explored LPD behaviors in 37 cases. The results are described, identifying the needed LPD creation behavior (called *LPD behavior* for short) in non-implementation specific language, and then in the UnBBayes-MEDG-specific LPD script language (called *LPD language* for short). The results also identified modeling patterns useful for the design of any first-order expressive modeling capability. Appendix B provides pattern details and examples. The patterns' usefulness is demonstrated by showing how they identify a solution to a longstanding upgrade request to the UnBBayes software. Finally, the dependency modeling effort also identified two useful modeling features that should be in any implementation of a first-order expressive modeling framework.

4.1 Overview of the UnBBayes-MEDG Local Probability Distributions (LPD) Script Language

The fundamental approach in creating the LPD in first-order expressive probabilistic modeling is conditional matching and count. In the MEBN specification, this is

called influence counting (Laskey 2008). For any parent configuration, a set of conditions are identified, and the number of parent instances that meet that set of conditions are counted. This sections discusses a specific LPD creation language that is part of UnBBayes-MEBN and UnBBayes-MEDG implementations. These implementations share a common LPD language. Both implementations assume a finite number of entities, and the LPD language creates LPDs in tabular form. Its basic form is

```
IF ANY / ALL paramsubset HAVE (Boolean Function) [ fur-
 ther IF conditions or probability assignment ]
ELSE …
```

Paramsubset is a list of ordinary variables (OVs), placeholders for the entities that will be tested in the `IF` statement. The OVs are comma or dot-separated. The *Boolean function* is a set of parent RVs whose states will be tested. It has the form of *Node=NodeState*, where *Node* is the RV name and *NodeState* is a specific state of the RV. There may be any combination of nodes (RVs) using "&" (and), "|" (or) and "~" (not). Any Boolean function can be implemented with "and" / "not" or "or" / "not" (Enderton 2001). RVs used in the *Boolean Function* must have an OV that is called out in the paramsubset list. There is no requirement that all the OVs in an RV must be called out. For each PC, `ANY` checks if there is at least one set of entities whose RVs meet the Boolean function condition. `ALL` requires that all possible entities meet the *Boolean Function* condition. If the `ANY / ALL` conditions are met, one may either immediately assign a probability to the states or test additional conditions before assigning probabilities. Any `IF` statement whose conditions are met results in a probability assignment before the `IF` statement is closed.

The ELSE statement closes an IF statement. It is active whenever the previous IF statement fails to find any matches. The ELSE statement may have either further IF statement (i.e. ELSE IF…) or have a final probability assignment. Every LPD ends with a final ELSE statement, which is the default probability distribution when nothing matches the previous IF conditions.

The basic behavior is a count of the number of entity combinations that meet the specified conditions. When determining how many entity combinations meet the Boolean condition, it uses only the entities instantiated in the parent configuration (PC). Any other entities in a class are ignored. The count results are available for use through the CARDINALITY(**paramsubset**) command. For any paramsubset, only one cardinality result is available at a time. No fine grained results are possible (e.g. how many entity combinations are in each of N possible states). A CARDINALITY value is available until the IF statement that generated it is closed.

It is important to understand exactly how IF ANY **paramsubset** counts. When **paramsubset** has a single variable, CARDINALITY counts the number of entities represented by that variable that meet the Boolean function condition. When there are multiple variables in **paramsubset,** then CARDINALITY counts the combination of entities represented by those variables that meet the Boolean function condition. For example, assume **paramsubset** has three variables**, f.m.c**, where the parent configuration has two entities from **f**, three from **m** and four from **c**. Then there are 24 possible

entity combinations among **f.m.c** and each combination is tested to see if it meets the Boolean function condition[4].

4.2 Dependency Modeling - Identifying Required LPD Behavior[5]

To uncover the full range of needed LPD language enhancements, this research explored a comprehensive range of possible parent configuration interactions, using a newly developed approach called dependency modeling. Dependency modeling looks at how combinations of RVs that model an entity's attributes, its relationships with other entities and its functions interact with each other in the LPD. It designates three RV types:

- Attribute random variable (A-type), with a single OV, model attributes. For each entity in a class, an A-type RV assigns a probability to each possible attribute state, drawn from a fixed (categorical) list
- Relationship random variable (R-type RV) with two OVs. It implements relationships such as *isFriendsWith*(**x,y**). It has Boolean states
- Functional random variable (F-type), model functions, and has two entity variables. One is the OV, the entity that is the subject of the RV. Since an F-type RV has entities as states, there is a second variable, called a state variable, that represents the states of the RV. This state variable is used in dependency modeling.

[4] The UnBBayes LPD language is intended as a general purpose language, able to model a wide variety of possible LPDs. This can make it computationally inefficient. It is recognized that some problem domains benefits from more specialized computationally efficient approaches. Capabilities to incorporate such approaches are also being pursued. That work is outside the scope of this dissertation.

[5] The approach is highlighted here and described in detail in Appendix A.

Assume there is an RV called *MachineLocation(**m**)*. It models a functional relationship between a specific machine entity and the room it is located in (F-type RV). When dependency modeling requires knowing what the state variable is, the RV is annotated as *MachineLocation(**m**)/**r***. It is possible for the same variable to be an OV for one RV and the state variable for another RV.

Since dependency modeling explores needed LPD behaviors for first-order expressive probabilistic modeling, it uses a template approach for its models. The template captures the basic model structure. In the model instantiation for a specific problem domain, the grounded model may have multiple instances of each RV type in the template. Dependency modeling focuses on combinations of RV types and how the parent interactions with each other influence the dependency with the child RV. The concepts modeled by the different RV type - attribute, relationship, and functional relationship - are knowledge representation concepts, used in knowledge modeling of a wide range of domains. Different interactions between RV types define different conditional counting behaviors. By exploring the range of interactions at the knowledge representation level of abstraction, one has confidence that the results are both widely applicable and reasonably complete. For the research done here, it explores different models where, at the template level, a child RV has one parent and where it has two parents. Figure 9 provides an example of each. Since there are three designated RV types, there are nine possible models with a single parent, and eighteen possible models with two RV parents, as shown in Figure 10.

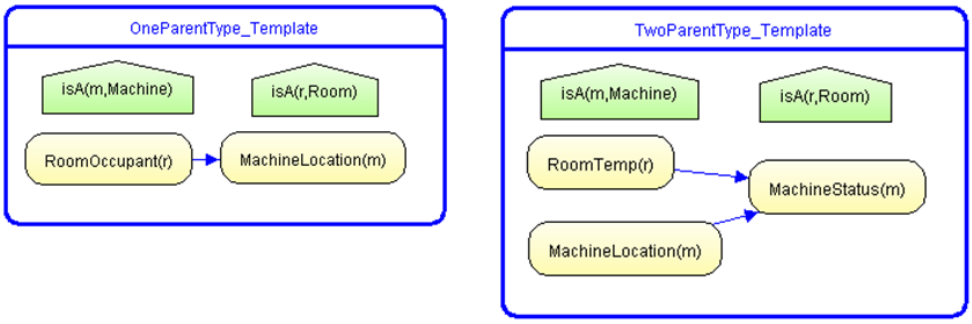


Figure 9: Templates where the child RV has one or two parents at the template level

In the usage here, the child RV is always RV3, and A / F / R identifies the RV type. In the two-parent-type models, the numeric identifiers for the parents (e.g. F1, R2) are reversible. There is no requirement for RVs to be different. In the two-parent-type models, if both parents have the same RV-type (e.g. an F-type RV), the two RV's may be the same or they may be different[6]. The dependency models in Figure 10 do not include

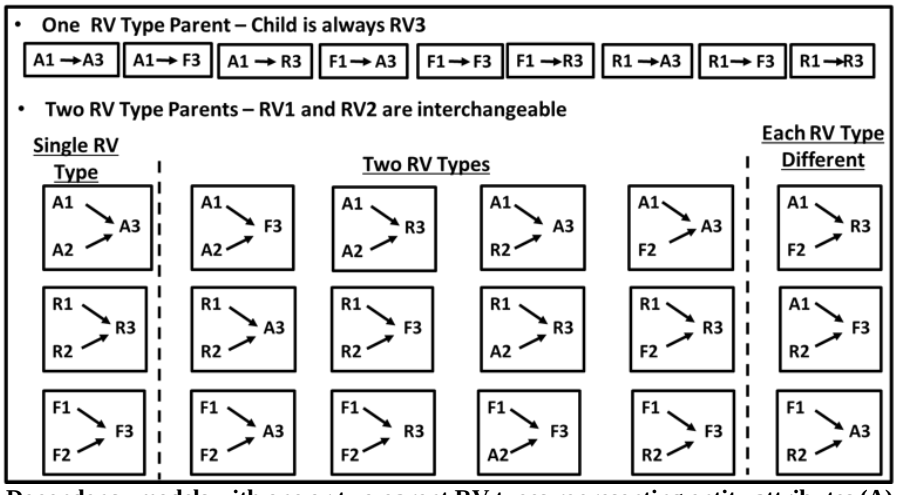


Figure 10: Dependency models with one or two parent RV types representing entity attributes (A), functional model and the informal interpretation of the meaning of the dependency.

[6] If they are the same RV, they must have different OVs, with constraints added that they cannot be the same entity.

OVs or state variables, and multiple variable combinations may be used. For the single parent models, there can be from one to four distinct OVs or state variables, and multiple variable combinations may be used. The two parent models can have up to six distinct variables (from every RV has the same variable to each RV has a different variable for its represented entity and its states). For many of the dependency models, different variable combinations can create a different grounded model. In the modeling, A(**x**) means **x** is the OV of the A-type RV A, F(**x)/y** is an F-type RV with **x** as the OV for the represented entity and **y** is the state variable, and R(**x,y**) means **x** and **y** are the R-type RV's OVs.

The construction algorithm uses the OVs in determining the structure of the grounded model. Each instantiated RV is for a specific entity or set of entities. If a parent RV has the same OV as the child, the parent OV must be the same entity as the child's OV. For the OV(s) not the same as the child's OV(s), one RV of that parent is instantiated for every entity in the class (subject to context variable limitations). Different combinations of functional relationships (F) or relationships (R) in the template can instantiate different parent sets for the same basic dependency model. Figure 11 below gives an example. It has two different instantiations of the same dependency model (A1, F2, A3), but the order of the variables for the F2 RV type is inverted between the two. The idea being modeled in both cases is that the status of a manufacturing machine (*working / broken*) depends on the room's temperature (*normal / hot*). Which room the machine is in is uncertain. This uncertainty is captured by the F2 RV-type. In Figure 11A,

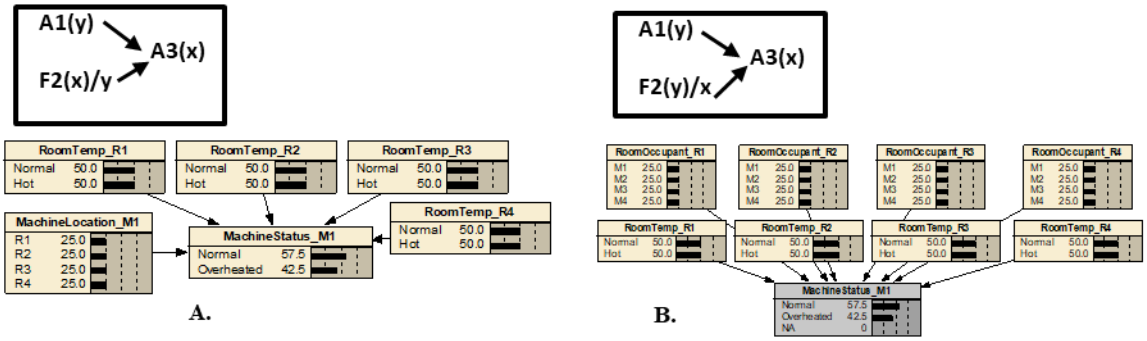


Figure 11: Inverting the variables in RV F2 results in a different pattern

this is *MachineLocation*. In Figure 11B, the F-type RV is *RoomOccupant* – note that it is the inverse of *MachineLocation* and its OVs are inversed. This results in two different parent sets. Assume the database has four machine entities and four room entities. In Figure 11A, *MachineLocation* has the same OV as *MachineStatus*, so only one RV is instantiated[7]. In Figure 11B, *RoomOccupant* has a different OV, so four instances are instantiated (state variables are not used in the construction algorithm).

Different variable combinations (OVs and state variables) can also affect the understanding of what is being modeled. There are collectively 39 distinct variable combinations cases for the one-parent-type dependencies and 414 cases for the two-parent-type dependencies. As discussed in Appendix A, it is difficult to find meaningful examples for many of these cases, especially when each RV has variables different from every other.

[7] In Figure 11A, there is a unmodeled requirement that a room can have only one machine in it. It was not modeled because it adds additional nodes that clutter the point of the model and does not affect the LPD under discussion. It could be modeled by including the remaining three MachineLocation nodes (for machines m2, m3 and m4) and adding a constraint node enforcing the requirement that each machine must be in a separate room. In Figure 11B, the constraint is modeled, using an embedded constraint approach. Note the state NA with 0 probability in RV MachineStatus_M1. See Appendix A.4, page 179 for constraint discussion.

Meaningful examples generally require that entities are shared among the RVs. So, this effort was restricted to having two distinct variables in the one-parent-type models, and two or three variables for the two-parent-type models. There is an additional requirement that a parent and the child RV must share a variable. Cases where every RV had the same variable were excluded, as these had a simple structure. This resulted in 41 cases. For each model, a simple example was created that had a reasonable dependency between parent(s) and child. In four models, no meaningful example could be found. This resulted in 37 cases for which the LPD interactions could be explored. For some of the cases, the existing LPD language capabilities were sufficient. But most required additional capabilities.

4.3 Patterns

It became obvious that there are recurring patterns in the model structures and LPD behavior. Eight distinct patterns were identified. These patterns have both a structural component and an internal LPD behavior component. The structural component is driven by the OV combinations, as discussed above. The internal LPD behavior results from the interaction between the knowledge representation concepts, as captured by the RV types. The patterns have two uses. First, they represent general behavior that is applicable to a variety of domains. As such, they are available for modeling support. Second, the patterns describe needed LPD behaviors. This section highlights the patterns, focusing on what they reveal about LPD language capabilities. The needed behaviors are identified in this section. Implementation details are deferred to the section 4.4. The patterns

are discussed in-depth in Appendix B. The patterns fall into three major areas that are called selector, existential selector, and embedded dependency.

4.3.1 Selector Patterns

In the Selector patterns, the Template has either an F-type or A-type RV (listed as either F2 or A2) with the same OV as the child RV. In the grounded model, this RV instantiates only one instance and acts as the selector. It identifies a condition that the instantiated RVs of the other variable type (RV1) must meet for the instantiated variables to have an influence on the child RV. There are three selector patterns: Select-One, Select-Match and Child-Select.

In the ***Select-One pattern***, the selector is an F-type RV. The state variable is the same variable as an OV for the other parent RV. The LPD for this pattern always exhibits context specific independence**.** This pattern is a generalization of the well-known pattern used to resolve reference uncertainty (Getoor and Grant 2006; Getoor et al. 2007). Figure 11A above has this pattern, in which a specific instance of RV A1 (*RoomTemp*) has an influence on a child RV A3 (*MachineStatus*). The child's OV may be from the same or different class than the OV in A1. The functional relationship in RV F2 enables the dependency between A1 and A3, identifying the specific A1 entity with the influence. The LPD behavior is to identify and use the state of the specific instance of the RV A1 whose OV entity is the same entity as the state variable of the selector. Figure 12 below gives the LPD behavior for two CPT parent configuration lines. To address this type of parent configuration and reasoning model, the needed LPD behavior are to

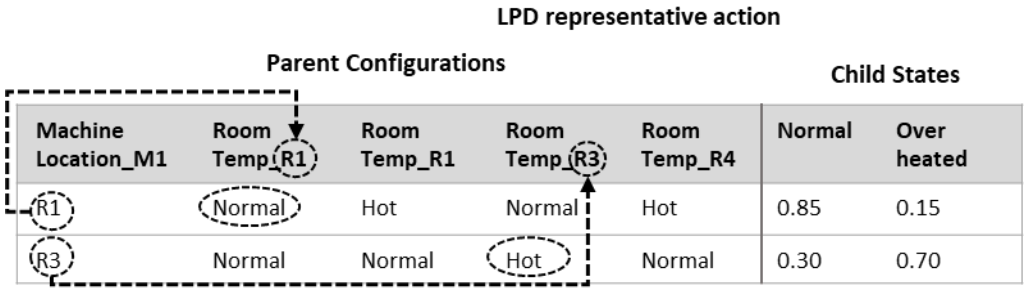


Figure 12: Select-one LPD behavior

- Recognize entity states
- Use an F-type RV's entity state as an indirect reference to the RV instance of interest. Specifically, one needs to able to say

  `(State(`*`A1(State(F2))=statex`*`)`, where A1 and F2 are the parent RVs.

The implementation described in section 4.4 below has a different form but the same effect.

The ***Select Match pattern*** has the same structural pattern as the select-one pattern, but a different LPD behavior. In this pattern, the selector identifies a state of interest, and the LPD language then counts the number of the other parent-type instantiations that match the selector state. In this case, the selector may be either an F-type or an A-type RV. The LPD behavior depends on the parent types. If the selector (RV2) is an F-type RV, then so is RV1, and the comparison is how many RV1 instances have the same entity state as RV2, e.g.

`COUNT((State(`*`RV2)= State(RV1)`*`).`

If the selector is an A-type RV, so is RV1, and the two RVs have an embedded dependency that specifies how the two RVs interact to establish the child RV's state

probabilities. The count behavior is more complex and will be shown below. In either case, the child RV in this pattern may be either an F-type or R-type RV. If it is an F-type, the OV of RV1 must be the same as the state variable (both are from the same class) of the child RV. If it is an R-type, its other OV must be the same OV as in RV1. A simple case with F-type parents is shown in Figure 13. Here, the basic dependency is that a machine can be in a room only if the same organization both owns the machine and is the assigned tenant of the room. In this example, there are four rooms and two possible organizations.

For each parent configuration, the number of matches is counted. The Select-Match pattern requires the LPD language have an entity attribution capability. It must know which entities in the Tenant RV instances matched the selector. In general, entity attribution is needed whenever the child RV is an F-type RV – the LPD language needs to know which entities in the child RV's states have a positive probability. The assigned probability in this example is 1/(# of matches) for the entities whose Tenant state matches the selector.

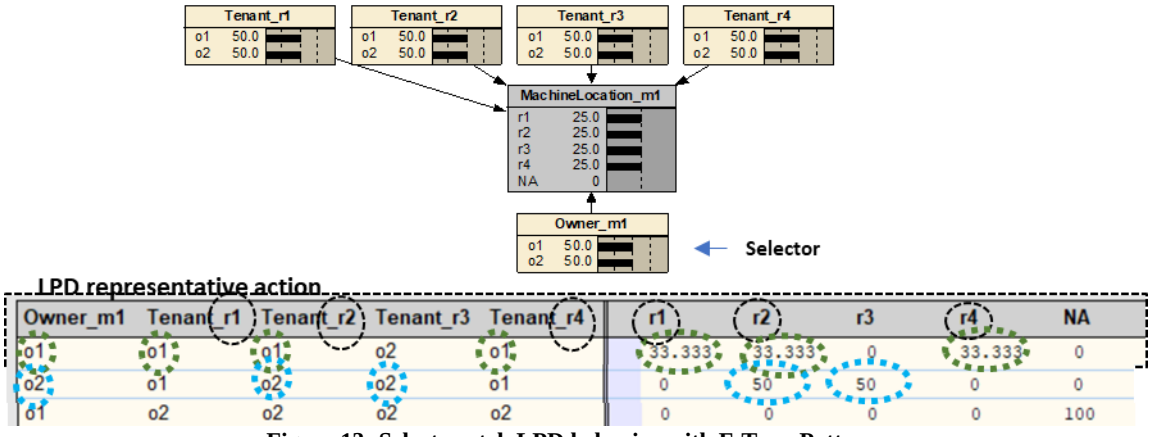
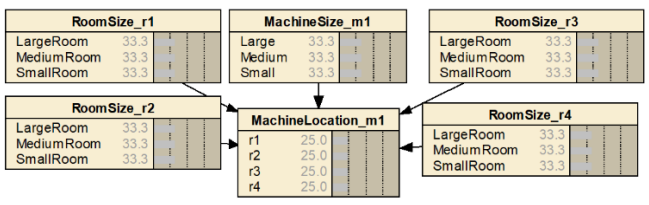


Figure 13: Select-match LPD behavior with F-Type Patterns

The counting used for the example in Figure 13 is simple. But the Select-Match pattern (and others) can require more robust counting capabilities. Consider the example in Figure 14. It is also a machine location problem. But here, the selector is an attribute, not a functional relationship. The probability that a machine is in a particular room depends on its size and the sizes of the different rooms. Any size machine can be in any size room, but a machine is more likely to be in a room the closer the room size is to the machine's size. The likelihood is also shown in the figure, where 3x means three times more likely than in the least similar room. In this example, a possible conditional probability for the ith room is

$w_i \,/\, \sum_j w_j n_j$, where w_i, w_j are the likelihood weights of each room, based on the degree of match to the selector attribute, n_j is the number of rooms that have the weight assigned to state j and j is the number of states.

Equation 2: Conditional probability of the ith room in the select-match example



	Machine Size		
Room	Large	Medium	Small
Large	3x	2x	1x
Medium	2x	3x	1x
Small	1x	2x	3x

Figure 14: Attribute-based select-match example with multiple states[8]

[8] If there is a constraint that only one machine may be in a room, the same comments as made in footnote 7 apply

The probability depends both on the number of rooms and the number that have each size. The LPD language here needs to count all possible conditions for *MachineSize / RoomSize* and organize by degree of match before assigning probabilities. This capability is called state counting. In addition to the entity state recognition capability discussed above, this example requires the following LPD behaviors:

- Entity attribution - know which entities of RV1 match the selector criterion
- State counting

The ***Child Select pattern*** applies to certain dependencies for an R-type child RV. In this pattern, both OVs in the R-type child RV are involved in setting state probabilities. As in all selector patterns, one child RV OV has the same class as the selector RV2. There are two variants. In the first, both parent RVs are F-type RVs. Here, the other OV from the child RV is used to determine whether the selected RV1's state supports a state of True. Specifically, the child RV has a state of *True* if the selected RV1 instance has the same entity for its state as the second OV in the child. In the second variant, the selector is an F-type RV and the other parent is an A-type RV. The OVs of both parents match the same OV in the child. In this case, the F-type RV's state variable and the other child OV are from the same class. If they match (unlike OVs for an RV's entity, state variables are not bound by a child's OVs), then the child RV has a state of *True*. Otherwise, it is *False*. The Child-Select examples explored requires the LPD behavior to have access to the entity instances in the child RV's OV to be able to make comparisons such as

> `State(RV1)= OVi(RV3), Where OVi` is the OV in ith position from the left.

4.3.2 Existential Selector Patterns

The second pattern category, Existential Selector, occurs across most of the cases examined. These patterns involve either a R-type or F-type RV as selector. If it is a R-type RV, one or both of its OVs will be the same as for the child OV. If the child RV has only one OV (A-type or F-type), one of the R-type's OV will be free. If the selector is a F-type RV, it is that RV's state variable, not OV, that will be the same as the child RV's OV. These patterns differ from the selector patterns in that they instantiate multiple instances of the selector RV, instead of only one. Each existential selector determines whether a specific relationship exists for each entity in the class of entities that may influence the child RV. If it does exist, the instances of a second RV type with the same entity have an effect. Uncertainty of a relationship's existence is a form of property uncertainty, hence the existential name (Poole 2011). These patterns use a variety of counting behaviors.

In the ***Existential Paired pattern***, the instantiations of both parent RVs are paired based on a common entity in their OV. The selector determines whether the entity is involved in the LPD. The state of RV1 determines the dependency. Figure 15 gives an example, where a student's grades partially depend on their professors' teaching abilities. There are four possible professors. The existential selector is `Teaches`. Only if a professor teaches John, does their teaching ability matter.

This example lends itself to conditional K out of N counting, where N is the number of John's professors, while K is the number of those professors that have Excellent teaching ability. The current LPD language can do this counting, but it has a limited

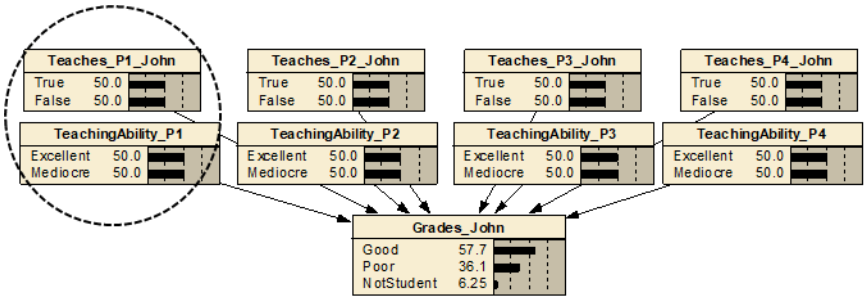


Figure 15: Existential paired pattern

ability to use the results in making probability assignments. Specifically, it can use the counts to assign probabilities based on arithmetic operations (e.g. probability *Good* =.6+k/n*0.4, including using min/max functions). More nuanced probability assignments occur if the LPD language would support:

- Using the cardinality / count results in an `IF` condition.
- Using table matching
- Using parametric probability distributions

An `IF` cardinality LPD language capability could be used as follows. Let `CARDINALITY(s)` be the number of John's teachers and `CARDINALITY(p)` be the number of John's teachers with Excellent teaching abilities. The usage would be

```
IF ANY s HAVE (CARDINALITY(s) = 3) [ IF ANY p HAVE (CARDINALITY(p) = 0) [ Good = 0.3, Poor =0.7]] ELSE...
```

`ELSE` repeats the above for `CARDINALITY(p)` of 1, 2, 3, with appropriate probability assignments. This `IF` capability would also be useful for implementing constraint requirements, such as "a room may hold only one machine", or "a person has two biological parents".

Using counting results as a conditional test parameter is a needed capability, but it results in a significant LPD language writing workload. Putting the probabilities in a table with a look up function would simplify the effort. This table and function are comparable to the Excel® lookup function. Finally, enabling the LPD to have a broader range of functions, including allowing predefined probability functions, would allow more probability models. For example, the logistic function is a viable probability distribution function for a variety of problems that use count-based aggregations (Kazemi et al. 2014).

In the ***Multi-Existential pattern***, each selector instance is applied against multiple instances of the second RV type. Rather than pairing between specific instances, it works across subsets (or all) of the instances. Figure 16 models the case where there are three possible individuals as the first author of a particular paper P1. Which person it is depends on two things: the topics the paper covers (`hasTopic`), and the expertise of each of the possible authors on those topics (`expertOn`). A paper can touch upon several topics. `HasTopic` is the existential selector in this model. For each topic covered by the paper, the LPD behavior is to determines which possible author has expertise in that topic. The assigned conditional probability then depends on the range of expertise of each

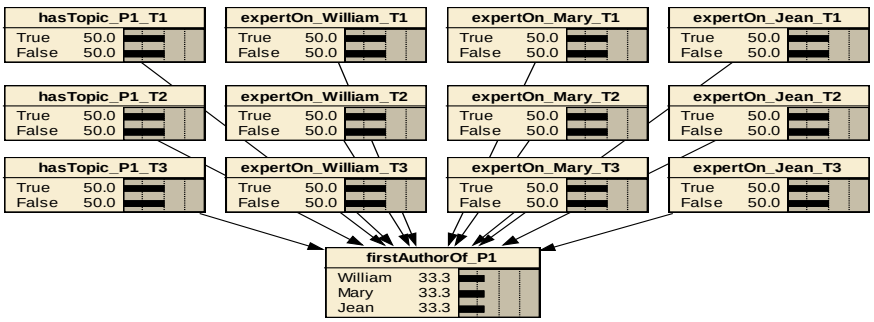


Figure 16: Multi-existential example – need for by-entity counting capability

possible author (e.g. William is an expert in two of the paper's topics). The person with the most expertise is more likely to be first author. This requires an ability to count states by specific entities. That is, for each possible author, the LPD language must count the number of topics they have expertise in. In addition, this example requires entity attribution, tying the count to an entity.

The ***Existential Child pattern*** is like the Child Select pattern discussed above, in that there is a direct dependency between the child RV's OV instance(s) and the selectors' states. Instead of a single selector, there are multiple existential selectors. In some examples, the child RV's OV(s) identifies a state that an existential selector must match for that selector to have an influence. In other cases, the child RV's OV(s) are the selectors. Figure 17 gives an example of the latter. The premise here is that the probability that two people are married depends partially upon knowing how many biological children they have in common – the more children, the more likely they are married.

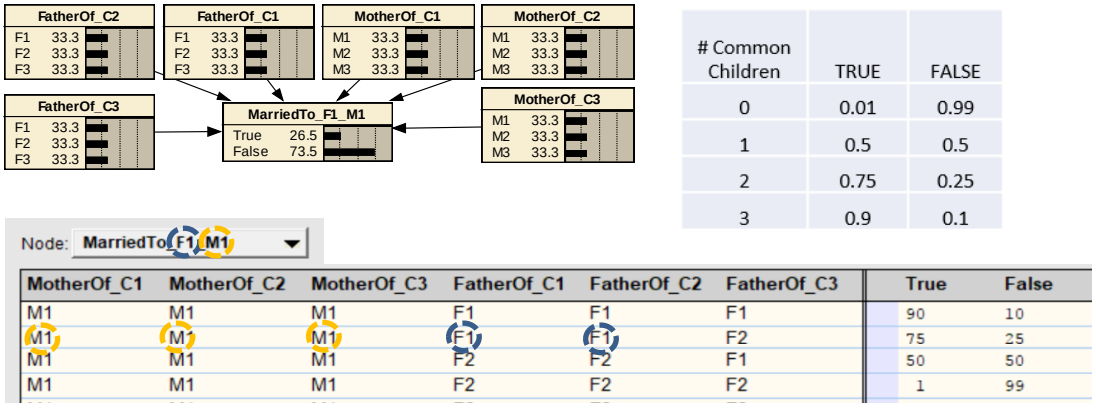


Figure 17: Existential child pattern, where the child OVs are the selectors

The required LPD behavior is

> Count number of child entities where `State(`*`FatherOf)`* `=` *`OV1(MarriedTo)`* `AND State(`*`MotherOf = OV2(MarriedTo)`*,where OV1 and OV2 are the first and second OV in *`MarriedTo`*

This counts the number of child entities whose parents match *`F1`* and *`M1`*, the specific father and mother entities identified in *`MarriedTo`*. Since one cares only about the total number and not which possible children match, there is no need for attribution or count by entity.

4.3.3 Embedded Dependency Patterns

The embedded dependency pattern are patterns applicable to one-parent-type cases. Because of the rules discussed in section 4.2 above, there are always two variables in one-parent-type cases, and both the parent and the child share at least only variable. These patterns model the influences an entity's attribute has on one of its relationships (including functional relationships), influence of an entity's relationship on one of its attributes, and the influence of a relationship on another relationship. There are three patterns within this category: conversion patterns, inversion patterns, and existence patterns.

The **Conversion pattern** is used to convert an F-type RV to an equivalent set of R-type RVs, and vice-versa. It is useful when the knowledge base has information in one form, but when the other form is more appropriate for the model. The resulting RV(s) has the same semantic sense as the parent RV(s). For example, if the parent is an F-type RV called *`FatherOf(`***`child`***`)`*, a possible conversion could be the child RV

*hasFather(**child,father**)*. The naming of the RVs and determining whether the two RVs have the same semantic meaning is the modeler's responsibility. The pattern only ensures that the uncertainties among the states is preserved between the RVs. Any F-type can be converted into a set of R-type RVs, one for each entity in the F-type's state variable. The resulting set of R-type RVs will be inherently constrained - only one instance can be true at any one time and the sum of the probabilities of each R-type RV being true must equal 1 across the set of R-type RVs. An R-to-F conversion can occur when the set of R-types is constrained to meet these same two conditions. If so, then one can convert them to a single functional F-type RV.

Inversion patterns provide the inverse of a relationship, which is not necessarily the same relationship with the variable order reversed. For example, *hasFriend(**x,y**)* is normally considered a symmetric relationship and is its own inverse. But *hasChild*(**x,y**) and *hasParent*(**y,x**) are each an antisymmetric relationship, but can be considered inverses of each other. If the inverse of a relationship is a different relationship, the modeler needs to understand the nature of the inversion. If the relationship is identified using a verb phrase, one can always create an inverse (in the English language) by switching the voice (active / passive) of the verb. For example, *hasFather(**x,y**)* and *isFatherOf(**y,x**)* are considered inverses of each other with the normal meaning assigned to those verb phrases in the English language. But often one is interested in an inverse that describes the relationship differently, such as *hasChild(**x,y**)* and *hasParent(**y,x**)*. In those cases, the dependency between the two may not be as tightly bound.

Because relationships can be expressed either as R-type (always) and F-type (for a functional relationship), there are four possible implementations. In the F-F case (e.g. *BeltLocation*(**belt**)/ **machine** to *BeltOn*(***machine***)/ ***belt***) [9], there must be a one-to-one relationship between the entities in the OV and state variable of the two RVs. The R-R case (*hasGrandparent*(**y,x**) to *hasGrandchild*(**x,y**)) is the most flexible, as it puts the least restrictions on what can be modeled. In the R – F case, there is a requirement that the R-type parent must be functional for the OV that is also the OV of the child variable. The last case, F – R, is unlike the conversion case. The resulting set of R-type RVs may have different constraints. For example, while the *FatherOf* relationship is constrained, inversing it to a *hasChild* relationship is not constrained in the number of children a father can have. But for a specific child, only one *hasChild* relationship can be true, if the parent class is filled only by fathers or by mothers (but not both – then the constraint is two RVs must be true).

Existence patterns are single parent patterns that describe dependencies between attributes and relationships. In these patterns, the same entity has both the attribute and the relationship under consideration, and the state of one influences the state of the other.

All the patterns discussed above model basic knowledge representation interactions. While specific examples are given to aid understandability, these patterns are not domain specific, and are widely applicable in first-order expressive probabilistic modeling.

[9] State variables are shown for understandability

4.4 LPD Language Changes

The dependency modeling exploration and the resulting patterns identified the required LPD behaviors. Table 4 highlights the both the overall LPD behaviors, as described in section 4.3, and the implementation for UnBBayes. The first four behaviors were previously implemented, as described in Section 4.1. The remaining ten behaviors are enhancements to the LPD language: eight enhance conditional counting, two expand the probability assignment options. The first enhancement is to allow the LPD language to recognize and use states that are entities of a class, rather than categorical or Boolean states. It applies specifically to F-type RVs. The enhancement recognizes that the Boolean function (*RV2 =* **OV**…) is a valid state reference, equivalent to (*RV2 = state*). The added requirement is that the OV must be called out in the parameter subset. In a PC, each RV instance has a defined state (Boolean, categorical or entity). The form *RV*=**OV** signals the replacement of the OV with the entity state of the RV. Its use assumes that only one instance of *RV* is created, as fixed by the structure of the model. This enhancement enables the uses specified below.

The second enhancement is to perform indirect referencing. The Select-One pattern identified a need to do so – specifically, the need to identify the entity whose attribute or relationship influences the child. The implementation for the example in Figure 12 (page 81) is

IF ANY **m.r** HAVE (*MachineLocation =* **r** *&*
RoomTemp = normal)

Table 4: Local Probability Distribution behaviors and language enhancements

Behavior	Command
Multiple condition testing	IF ANY **paramsubset** HAVE (*Boolean Function)*
Nested Ifs	IF ANY **paramsubset** HAVE…. [IF ANY **paramsubset** HAVE….
Summary count by entity combination	CARDINALITY(**paramsubset)**
Probability assignment	With modeler defined values or arithmetic function, with MIN / MAX
Recognize entity states	IF ANY **paramsubset** HAVE (*RV = OV …..*)
Perform indirect referencing (RV1(RV2) = state)	IF ANY **paramsubset** HAVE (*RV2 = OV & RV1 = state…*)
Perform state comparison (RV1 = RV2)	IF ANY **paramsubset** HAVE (*RV1(OV) = RV2(OV)…*) May have multiple OVs
Identify the specific entities of a class that met the IF conditions	COLLECT#(**OV**)
Use child RV's OVs as test conditions	Implemented as part of entity states recognition
Count by RV states	IF ANY **paramsubset** HAVE (…*RVx = state1*, COUNT1) ORSC (…*RVx = state2*, COUNT2) ORSC…..
Count by individual entity	FOR EACH **paramsubset**{IF ANY…….., COUNT#(a)) [WAIT]}
Allow conditional behavior based on the count results	IF COUNT# < / > / = *x* may use CARDINALITY as well
Use cardinality / count in lookup tables for probability assignments,	TABLE / TABLEMATCH
Enhance use of parametric probability distributions	Add exponentiation and combinatoric functions. Add interface to common probability functions

Here *MachineLocation* always is a single instance with a specific entity state in each parent configuration. Within the LPD language, this dynamically fixes the **r** OV to a specific entity, the entity state of *MachineLocation*. Only one *RoomTemp* instance has the same entity.

The third enhancement is to perform entity state comparison. The select match pattern in Figure 13 on page 82 identified the need for this capability. Its implementation is

IF ANY **m.r.o** HAVE (*Owner*(**m**) =*Tenant(***r***)*)

The count in this case is the number of *Tenant* states that match the owner state. This works correctly only when there is one possible entity for the OV of one of the RVs called out in the Boolean function. The variable **o** is in parameter subset because it is the state variable being compared between the two RVs.

The fourth enhancement is to allow the LPD language access to the entities represented by OVs in the child RV. This enables comparing specific parent RV entities (OV or state variable) to the child's entities, as required by the Child-Select (page 84) and Existential-Child patterns (page 88).

The fifth enhancement is to support entity attribution. This is done via the COLLECT#(**OV**) command. It supports probability assignments to F-type RVs. This command provides entity attribution. It collects the specific entities whose properties meet the IF conditions. In the example in Figure 13 (page 82), one needs to know which room entities

have the same owner as the machine entity. COLLECT#(**OV**) is a list of entities. There may be multiple collects made. "#" is replaced by a unique numeric value to distinguish the different instances of COLLECT. **OV** identifies the class whose entities are collected. COLLECT#(**OV**) is then used in the probability assignment section of the LPD. In the example, it would be used in the probability assignment section of the LPD as follows:

COLLECT1(**r**) = 1/CARDINALITY(**m.r.o**)

This states that each entity in COLLECT1(**r**) is assigned a probability of 1 over the number of entities that met the match condition. For the second PC line in Figure 13 (page 82), it results in

r2 0.5,
r3 0.5

The next two changes enhance the LPD language's counting capabilities, by enabling state counting and by-entity counting. A limitation of the current LPD language is that it cannot do state counting. It allows the LPD to step through the state conditions and count the occurrences of each prior to assigning the child RV's state probabilities. Figure 14 (page 83) gives an example of the need. One needs to know the count of each possible state to correctly compute the conditional probability. The sixth enhancement, state counting, has two parts. First, it adds an ORSC ("or State Count") condition to the IF statement. It tells the script interpreter that the IF statement is not complete until all the conditions tested by the ORSC clause(s) is completed. The full implementation is

IF ANY **paramsubset** HAVE (*RV2=stateX*)
[IF ANY **paramsubset** HAVE (*RV1= state1*, COUNT1, COLLECT1(**OV**)) ORSC (…*RV1 = state2*, COUNT2, COLLECT2(**OV**))
ORSC….

which continues until all the states of RV1 have been counted.

The second part adds a new option for recording counts. Presently, such information is accessed via the CARDINALITY command. But CARDINALITY has two limitations. First, CARDINALITY provides a summary count. Here, it would provide a count of all the entity sets that met one of the state conditions. But we need a count by state. Second, CARDINALITY only retains count information to the end of the IF-statement that generated it. But there are cases where one needs to keep that information, possibly to the end of evaluating a PC. For example, if one is using a state count to obtain the value of the child's state probabilities, the state count is not complete until all IF statements have been assessed. To retain compatibility with legacy applications, a new command is used: COUNT#, where # is a number assigned by the developer. Each subclause within the ORSC extended if clause has its own distinct COUNT# value. The LPD language user can associate each instance of COUNT# with a particular state and can use it in the LPD probability assignment. The implementation for the example in Figure 14 (page 83) is

IF ANY **m** HAVE (*MACHINESIZE = LARGE*)
[If any R have (*ROOMSIZE = LARGEROOM*, count1, collect1(**r**)) ORSC (ROOMSIZE = MEDIUMROOM, count2, collect2(**r**)) ORSC (ROOMSIZE = SMALLROOM, COUNT3, COLLECT3(**r**))
[COLLECT1(**r**) = 3/(3*COUNT1+2*COUNT2+COUNT3),
COLLECT2(**r**) = 2/(3*COUNT1+2*COUNT2+COUNT3),
COLLECT3(**r**) = 1/(3*COUNT1+2*COUNT2+COUNT3)]
ELSE….

The LPD would be repeated for the other two *MachineSize* RV states, with appropriate adjustments for the numeric values in the probability assignment section.

The seventh enhancement enables by-entity counting. The example in Figure 16 (page 87) identifies the need for by-entity counting. The driver here is that in the example, there is nothing to group the counting by the possible authors. The enhancement uses a for-loop construct to provide this grouping

> FOR EACH **OV** {IF ANY **paramsubset** HAVE (*Boolean expression*, COUNT(**OV**)), an optional IF COUNT(**OV**) test, [probability assignment or WAIT]} [probability assignment]

There are six distinct features of this construct (OV is replaced by the actual OV symbol):

- The FOR EACH **OV** segment tells the LPD interpreter to loop through the remaining section with the OV value fixed to a particular entity on each pass
- The {} contains the conditional testing executed during the looping. It may use any legal Boolean expression
- COUNT(**OV**) is a two-column list. The first column identifies the entity from the class of OV that is being counted. The second column is the count for that OV
- Within the {}, one may make an immediate probability assignment. For example, if the segment being counted has a constraint condition that fails, one may immediately make an *NA* assignment (see enhancement seven below). This ends the expression
- If there is no immediate probability assignment, WAIT is used in the probability assignment section to signal that probability assignments will be made after the looping is completed

- The final item is the probability assignment. Since COUNT(**OV**) already includes the specific entity to which a count is assigned, it serves as both a COUNT and a COLLECT.

The use for COUNT in the Figure 16 example (page 87) is

COUNT(**a**) = COUNT(**a**) / SUM(COUNT(**a**))

This means that the probability for an entity in **a** is the count for that entity divided by the sum of the counts for all entities in **a**.

The eighth enhancement allows COUNT# and CARDINALITY to be used in IF conditions. For example, suppose there is a constraint condition where the set of parent RVs may have only one instantiated node with a certain state. An IF command is executed to test for that condition. One could then use CARDINALITY as follows:

IF CARDINALITY(**paramsubset**) > 1 OR CARDINALITY(**paramsubset**) <1 [*NA* = 1]

>, <, and = are all allowable test conditions. Either side of the IF statement can be any Boolean combination of COUNTS / CARDINALITY, and any arithmetic combination of COUNTS / CARDINALITY and / or numbers.

The ninth enhancement is to incorporate lookup tables. A lookup table eliminates the need for lengthy probability assignment sections using the IF CARDINALITY / IF COUNT# commands. The TABLE command is used to build the table. Its form is

TABLE((ordered list of allowable COUNT or CARDINALITY names) (list of names of *child states* in order used in Template))
(COUNT1a,…,COUNTna, probability state 1a, …., probability state ma)
(COUNT1x,…,COUNTnx, probability state 1x, …., probability state mx)

Example:
TABLE(COUNT1, CARDINALITY(**m.r**), *Short, Average, Tall*)
(2, 2, 0.8, 0.2, 0)
(2, 1, 0.4, 0.4, 0.2
(2, 0, 0.2, 0.7. 0.1)
(1, 1, 0, 0.2, 0.8)
(1, 0, 0, 0.1, 0.9)
(0, 0, 0, 0, 1)

The largest numeric value should be interpreted as that value or anything greater. This allows for capped tables. In the case above, a COUNT1 of 4 and a CARDINALITY(**m.r**) of 1 will use the 2, 1 probability values.

TABLEMATCH is invoked within a probability assignment segment (the […….]). It is a complete assignment. No other entries are needed for that specific [….].

EX: [TABLEMATCH(COUNT1, CARDINALITY(**m.r**)]

The tenth enhancement adds a broader range of mathematical functions, including exponentiation and combinatorics, for use in making probability assignments. It also allows access to a probability function library for commonly used probability distributions.

4.5 Using the Patterns

This section provides an example of how the design patterns can be used. They identified a solution to a long-standing capability request for UnBBayes-MEBN. Both MEBN and MEDG use context variables (CV) to identify specific conditions that must be met before instantiating RVs in an SSBN / SSDG. Often the CV is of the form F(x) = y or R(x, y), where F is an F-type RV and R is a Boolean R-type RV. Any entity that does not meet the CV requirement is not instantiated in the SSBN using that MFrag's RVs. If the database has findings that an entity meets or does not meet the condition in

the CV, the construction algorithm knows whether to instantiate RV's with that entity. In those cases, the RV identified in the CV is not a part of the SSBN. But if there is uncertainty whether a CV is true for an entity, then a different approach is required. In those cases, the MEBN specification calls for instantiating the CV in the SSBN. The UnBBayes MEBN and MEDG implementations can only instantiate CVs of the form F(x) = y, where the x OV is the identical OV to one of the child RV's OVs. It does not instantiate CVs of the form F(y) = x, where x, not y, is a child RV OV, or of the form R(x,y). In addition, the current implementation instantiates the CV as a distinct RV type separate from the RV that is used to form the CV. This means that any information in the MTheory about parents or children of that RV are not incorporated in the SSBN.

The patterns described in section 4.3 above provide the way to correctly implement CV uncertainty for these three forms. In these forms, the relationship modeled by the CV licenses a dependency between the two entities in the relationship. This allows the attributes / relationships of the parent entity to influence the child RV. When one is uncertain as to which specific entity has the relationship for the child RV's entity, one has reference or property uncertainty. Reference uncertainty occurs in the form F(x) = y, where x is the same OV as in the child RV. In implementing it, one uses the Select One pattern.

When the form is either F(**y**) = **x** (**x** is also a child RV OV) or R(**x,y**), one has property uncertainty (one does not know whether a property exists between a specific set of entities). The form R(**x,y**) is modeled using the Existential Paired pattern, while the form F(**y**) = **x** is modeled using the Existential Child pattern. The implementation is

straightforward. When uncertainty exists in a CV with one of these three forms, the RV represented in the CV is instantiated in the SSBN, using the construction algorithm rules. The resident MFrag for the RV in the CV is checked and any parents to that RV are instantiated as well. In addition, any children to that RV may be instantiated in accordance with the construction algorithm. An arc is drawn from each instantiated CV RV to the child RV. The LPD in the child RV is modified to account for the new parent(s). First, the OVs in the CV RV are added to the **`paramsubset`**. The **`y`** OV should already be in the existing paramsubset, as this is the OV for the RV that has the influence on the child RV. Second, in the LPD script where the **`y`** OV is invoked, add either F = **`y`** &…, F = **`x`** &…, or R = *`True`* & …. to the condition list, where F or R is the RV name and it is followed by the existing LPD condition statement.

When modeling problems with constraint variables of the form (where **`x`**, but not **`y`**, is also the child RV's OV) or, the modeler must address whether there are constraints on how many instances may be true (if R(**`x,y`**)) or have the child's entity for its state (if F(**`y`**) = **`x`**). If there is a constraint, it needs to be included in the CV's MFrag **.**

4.6 Other Modeling Tool Enhancements

The dependency modeling effort identified two useful modeling enhancements, applicable to any first-order expressive probabilistic modeling framework. The first is to allow mixed entity / attribute states for functional RVs. This makes it possible to use functional RVs when there is the possibility that no relationship of the type represented by the RV exists for the entity. For example, in much of the Western world, a person may be legally married to one other person but does not have to be married. If one has a

MarriedTo(**x**) RV where the state variable represents the possible marriage partners, it would be helpful to have a state of *NotMarried* as well. Also, if there is an embedded constraint on the RV[10], there needs to be an ability to add an *NA* or similar state to map parent configurations that violate the constraint to that state. These added states should not be part of the entity class, as they will be invoked as entities in every case where the class is invoked.

The second is to define a fixed finding for all instances of RVs used as constraint nodes. Constraint nodes only work when they have a finding that states that the probability of the state to which the disallowed PCs map to has a probability of zero. In the current implementation, the user must add a finding for every constraint node instantiated in the model. It would be useful to enable the model developer to declare that every instance of an RV in an template will always have a finding of "x". The template structure is modified to add an optional field in the section where states are defined to declare that a specific state always has a probability of 1 (true) or 0 (false). Any instance of that RV will then have a finding created with that information.

4.7 LPD Language Requirements Conclusions

The LPD undergirds a first-order expressive probabilistic modeling tool's capabilities, providing the probability and utility information used in the decision graph's solution algorithm. The robustness of the LPD language capabilities affect the range of decision problems the DG can address. Dependency modeling was developed for this re-

[10] See appendix A.4 for details if interested

search to explore how RVs that model knowledge representation concepts interact in an LPD. These concepts are widely used to model semantic content in many domains. This makes dependency modeling results applicable to the decision theoretic modeling of those domains. The analysis of the results identified ten generally applicable LPD language enhancements needed to model all the LPDs represented in the dependency modeling cases. The modeling results also provide test cases for verifying the correctness of the implementation of these enhancements. In addition to identifying LPD language changes, the case analysis uncovered eight structural patterns that are useful in any first-order expressive probabilistic or decision modeling. The patterns connect the external structural and the internal LPD behavior. Two patterns provided a solution for a long-standing UnBBayes-MEBN deficiency report. This sets the stage for enhancing decision graph modeling capabilities.

CHAPTER 5
DECISION MODELING ENHANCEMENTS

This chapter details enhancements to increase the range of decision problems that the first-order expressive decision graph tool implementations can address. The analysis of decision problem differentiators discussed in Chapter 3 identified two desirable enhancements: asymmetry management and context variation. Decision graphs cannot inherently handle asymmetric problems. The standard solution algorithms require a symmetric decision graph. This chapter presents two algorithms that symmetrize decision graphs in preparation for solution. For context variation, a modeler could include all possible context variations in a single decision graph. But this can result in a large decision graph, most of which is not used in a specific case. One can tailor the instantiated decision graph to a specific context if one's modeling framework allows the use of attribute RV-based context variables. While the Multi-Entity Decision Graph modeling framework does, its current implementation, UnBBayes-MEDG, does not. How to do so in any MEDG implementation is addressed below. Finally, Chapter 4 addressed the LPD behaviors needed to use functional relationship RVs, which have entities as states, as parents to other nodes. These LPD behaviors also allow the modeling of decision problems where a decision node has alternatives defined as entities from a class. But modeling entity alternatives adds additional modeling requirements that a modeler needs to be aware of. These

are described in this chapter. Asymmetry management, improved context variation modeling, and modeling demands for entity alternatives are discussed below.

5.1 Enhancing Asymmetry Management Capabilities

This section describes changes that enhance modeling of asymmetric decision problems in a first-order expressive decision modeling tool. Chapter 2 identified that for problems with multiple decisions, the effects of learned information and prior decisions can introduce decision graph asymmetry. There are three kinds of decision problem asymmetry:

- Functional asymmetry occurs when a previous decision or learned information disallows one or more alternatives in a follow-on decision. The integrative, sequential, and hierarchical patterns from Chapter 3 can have functional asymmetry
- Structural asymmetry occurs when a previous decision or learned information nullifies the effects of one or more decision graph nodes in the decision problem under consideration. Here, there are multiple decision paths through the decision graph, driven by the effects of previous decisions and learned information
- Order asymmetry, when one has a sequence of decisions with learning occurring between decisions. This learning affects the value of follow-on decisions, and the order in which decisions should be made is itself a decision problem.

The Granddad decision problem described in Figure 18 below exhibits all three forms of asymmetry. Figure 19 provides an initial Granddad problem model. It does not show the problem's asymmetry characteristics. In addition, standard decision graph solution algorithms will not execute correctly on this graph.

Granddad promised his three granddaughters that he would take them on an adventure. He still hadn't made up his mind on what to do. The choices are between going to an amusement park or an outdoor adventure park. He decided he would ask them what they wanted to do. For each choice, Granddad would have to decide which one of two nearby amusement parks or one of two outdoor adventure parks they would go to. Which one will be selected depends on the costs and difference in attractions of each option, and how happy the girls will be with their adventure.

For the selected choice, Granddad would need to pick the route to get there. For each choice, there are three routes: use a very convenient toll road that would get them there quickly, take a scenic route that Granddad enjoys taking, or take a route that would go by a special bakery shop the girls love to visit. If Granddad forgets to bring the toll road transponder, he cannot take the toll road.

First, Granddad must get to the girls' house. But he cannot start the car. He suspects three likely causes: clogged fuel filter, broken ignition cable, or a faulty engine controller. He has a spare for each. Each takes a different amount of time to replace and each has a different probability of resolving the problem. If replacing them didn't work, he can rent a car at a rental agency up the street. He can select the rental car option at any time. Granddad wants to minimize the amount of time and cost for repairing the car. There is no obvious sequence to resolving this problem.

Granddad's overall goal is to maximize his "granddad points", while keeping the costs reasonable.

Figure 18: The Granddad problem

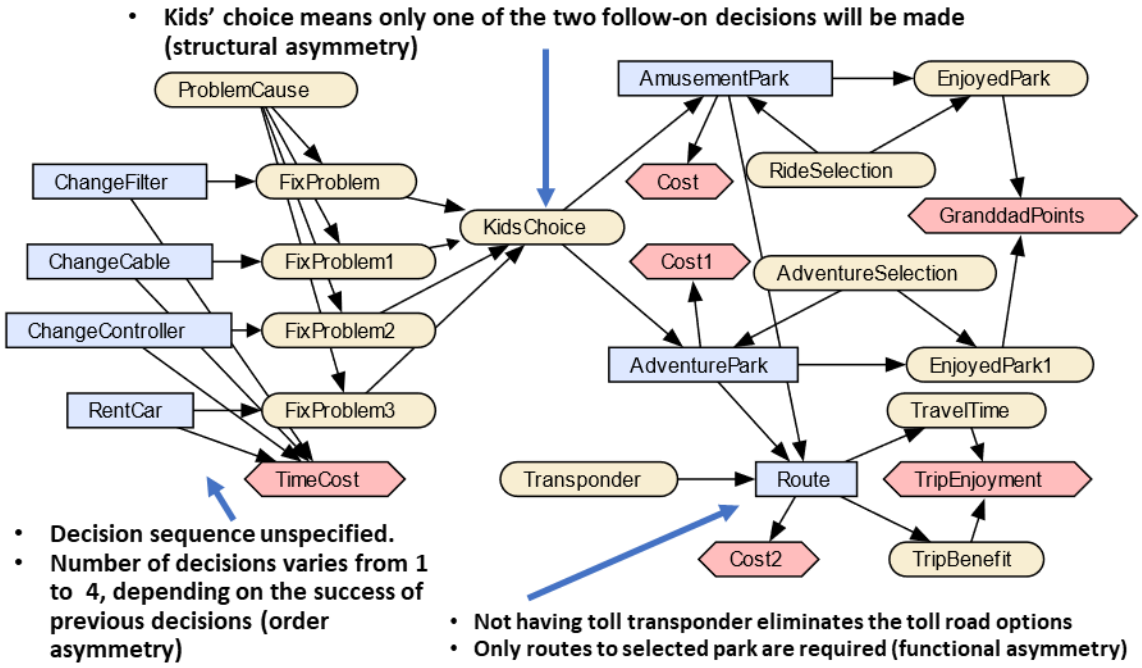


Figure 19: Initial Granddad problem model, demonstrating need for asymmetry visualization

5.1.1 Asymmetry Visualization

The key reason for developing a graphical model is to provide the decision-maker with a visual display of all knowledge relevant to the decision problem. As shown in Figure 19, asymmetry is not visually apparent in a standard decision graph. Two distinct visualization methods are proposed in the literature: the sequential influence diagram and the decision analysis network. The methods are shown in Figure 20. The sequential influence diagram method adds a dashed line to signal the existence of either functional or structural asymmetry. The dashed lines are labeled with either the parent state that triggers functional asymmetry and the inhibited child RV state, or the parent state

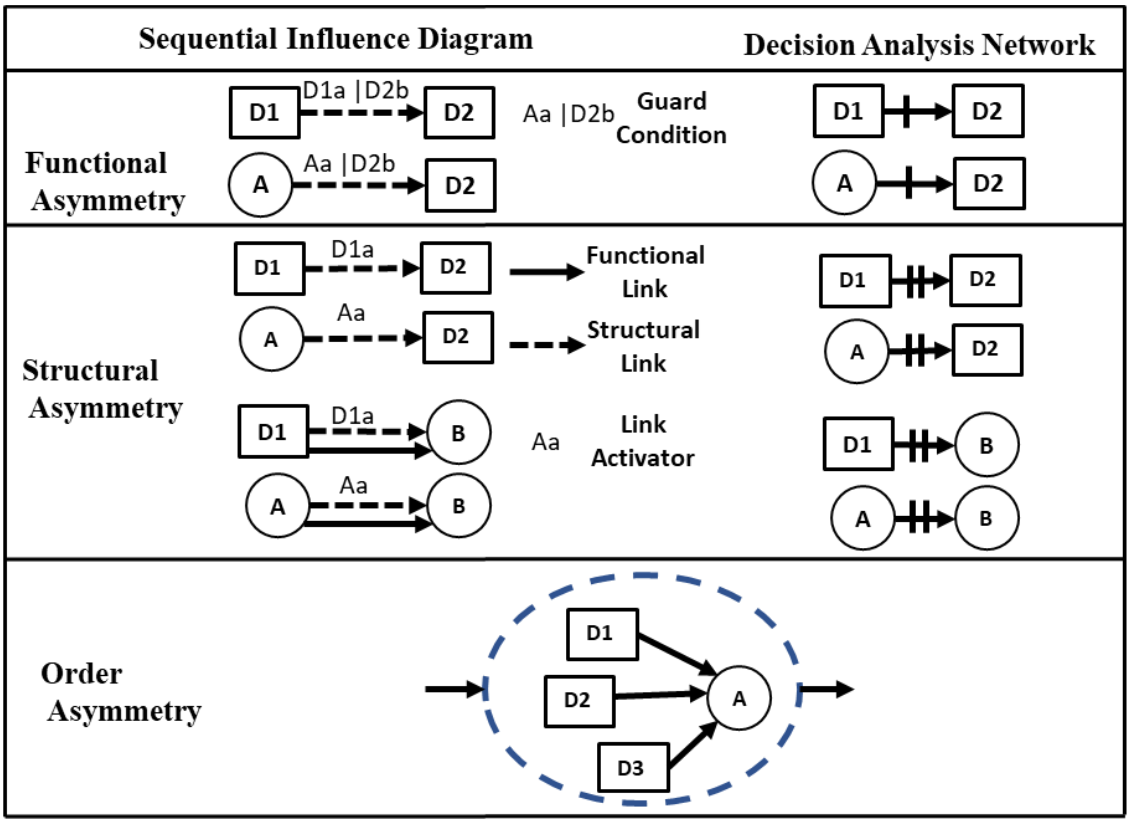


Figure 20: Approaches to asymmetry visualization - Sequential Influence Diagram vs Decision Analysis Network

that indicates the link exists (for structural asymmetry). Solid lines show functional links that one node's state depends on another node's state. A pair of nodes may have both a structural and a functional relationship. (F. V. Jensen, Nielsen, and Shenoy 2006). The decision analysis network method uses conventional decision graph lines, but adds distinct markings to signal asymmetry. Functional asymmetry is signaled by the single slash on an arc, while structural asymmetry is signaled by the double slash on an arc. This method does not display the specific condition(s) that trigger the asymmetry. Rather, detailed asymmetry information is included in an associated compatibility restriction table (CRT) (Díez et al. 2014). In both methods, order asymmetry is indicated by enclosing all involved nodes within a dashed ordering oval.

For MEDG implementations, the proposed asymmetry visualization method uses the Decision Analysis Network method to visualize asymmetry. It is a less cluttered method, but still effectively signals the presence of an asymmetry. Figure 21 shows the asymmetry visualization of the Granddad problem. All the nodes affected by order asymmetry are included within an ordering oval. If multiple instances of order asymmetry exist, each has its own ordering oval. Each possible decision is identified separately within the oval. The chance nodes represent the possible consequences of decisions. These can include a stopping condition, which means that the underlying problem has been resolved or is unresolvable within the current decision set. In either case, no more decisions from the current set are required. The compatibility restriction table (CRT) is part of the node data section within the specific implementation. It includes information not only to support the visualization, but information needed to support automatic

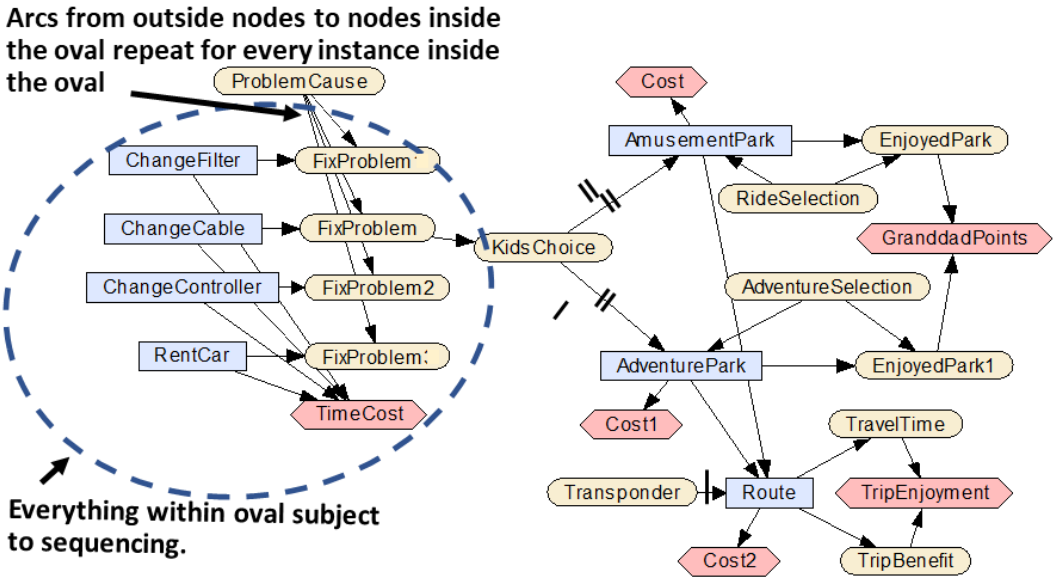


Figure 21: Granddad problem with asymmetry markings

asymmetry handling. The node with the CRT is the affected node. The parent node(s) that create the asymmetry are the trigger node(s). The parent configuration (the states) of the trigger node(s) that create the functional or structural asymmetry are collectively called the triggering condition(s). The dependency structure among the nodes is visualized in the template's graphical section.

Each affected node has a CRT. In addition, decision and chance nodes within an ordering oval each have a CRT. The CRT format in Figure 22 has three parts. The first is a checkbox to identify the type of asymmetry the node is involved with. A node may have multiple asymmetries. The second part provides five items of information needed to manage order asymmetry. Order asymmetry nodes are identified by having the same order problem number. This allows the construction algorithm to identify and associate all involved nodes. The name of the decision supernode name must be provided in at least

one CRT. If the affected node is a consequence node, it has a consequence supernode name. Consequence nodes with the same supernode name will be merged into a single supernode with that name. The name may be the same as one of the consequence names being merged, or a unique name not found in the MTheory. If a consequence node has a stopping condition, that condition is identified in this section. A stopping condition signals that a previous action has resolved the problem or entered a condition where the current decision set is no longer applicable. Finally, there is an entry to identify whether resolving the order asymmetry requires a positive first decision, or whether it is possible that no decisions are made at all (e.g. the prior information may be optimal for making the decision). The third part provides information on restricted states and provides key information for all three forms of asymmetry. It identifies the specific parent configurations (PC) that trigger the asymmetry. For functional asymmetry, the restricted state box identifies the node state inhibited (one PC and one inhibited state per line). For structural

Asymmetry Type		Order Asymmetry Factors	
Functional ○		Order Problem Number	#
Structural ○		Decision supernode name	Name
Order ○		Consequence Supernode Name	Name
		Stopping condition state	State(s) names
		Absurd allowed on 1st decision	Y/N
State Restrictions			
Trigger Conditions			Restricted State
Parent Node			
State			

Figure 22: Compatibility restriction table template

asymmetry the restricted state is `?All`, which signals that all states are restricted. For order asymmetry, the compatibility restriction table identifies which set of alternatives are restricted for use in follow-on decisions. The grounded model construction algorithm is modified so that when a node with a restriction table is found, the appropriate visualization marker is included on the graph. This visually signals to a reviewer that a restriction table exists for that node. The next section describes the table's use.

5.1.2 Symmetrization Process

Asymmetry adds a dynamic element to decision graphs. During the graph execution, learned information or prior decisions can eliminate decision options, make segments of the decision graph irrelevant, or make the appropriate decision order uncertain. Standard solution algorithms do not work on asymmetric graphs. The literature documents several solution approaches that work on asymmetric graphs, but only one claims to work on all forms of asymmetry (F. V. Jensen, Nielsen, and Shenoy 2006). However, it has not been widely adopted. The other option is to symmetrize the graph, modifying the decision graph by adding or modifying elements to account for these dynamic effects. Each asymmetry type requires a different set of modifications. The literature tends to describe these at a general level, with limited implementation detail. The best one found is König (2012). These are described below. It is recognized that symmetrization has significant scalability issues when dealing with order asymmetry. This should spur future work to develop a more scalable approach. Section 0 then provides two algorithms that apply these actions automatically.

5.1.2.1 Symmetrizing Functional Asymmetry

Functional asymmetry is easiest to symmetrize, as only the affected node requires action. Essentially, one inhibits one or more states, but not all of the states, when the triggering condition(s) exist (if all states are inhibited, one has structural asymmetry, not functional asymmetry). Figure 23 identifies the basic graph additions needed to symmetrize graphs with functional asymmetry. If the affected node is a chance node, no special action is required. Inhibiting a chance node state means giving it 0 probability, and this is done within the node's LPD. As only decision nodes require additional action under functional asymmetry, all references to functional asymmetry will be for affected decision nodes only.

In a standard DG, one cannot dynamically eliminate decision alternatives from consideration when a triggering condition exists. Instead, one makes these alternatives unattractive by ensuring that when the triggering conditions exist, they have the lowest expected utility of all the decision alternatives in the affected decision nodes. There are two ways to do this. The easiest way to do this is to add a lockout utility node (or lockout node) to the graph. This is a utility node which assigns a decision alternative a large negative value when a triggering condition exists to inhibit it[11]. The lockout node has the triggering condition node(s) and the affected decision node as its only parents. In the lockout node, the local utility distribution assigns a value of 0 to every combination of non-triggering parent configuration and decision alternatives. For a parent configuration

[11] This assumes the standard decision graph convention that higher utility values are preferred over lower utility values.

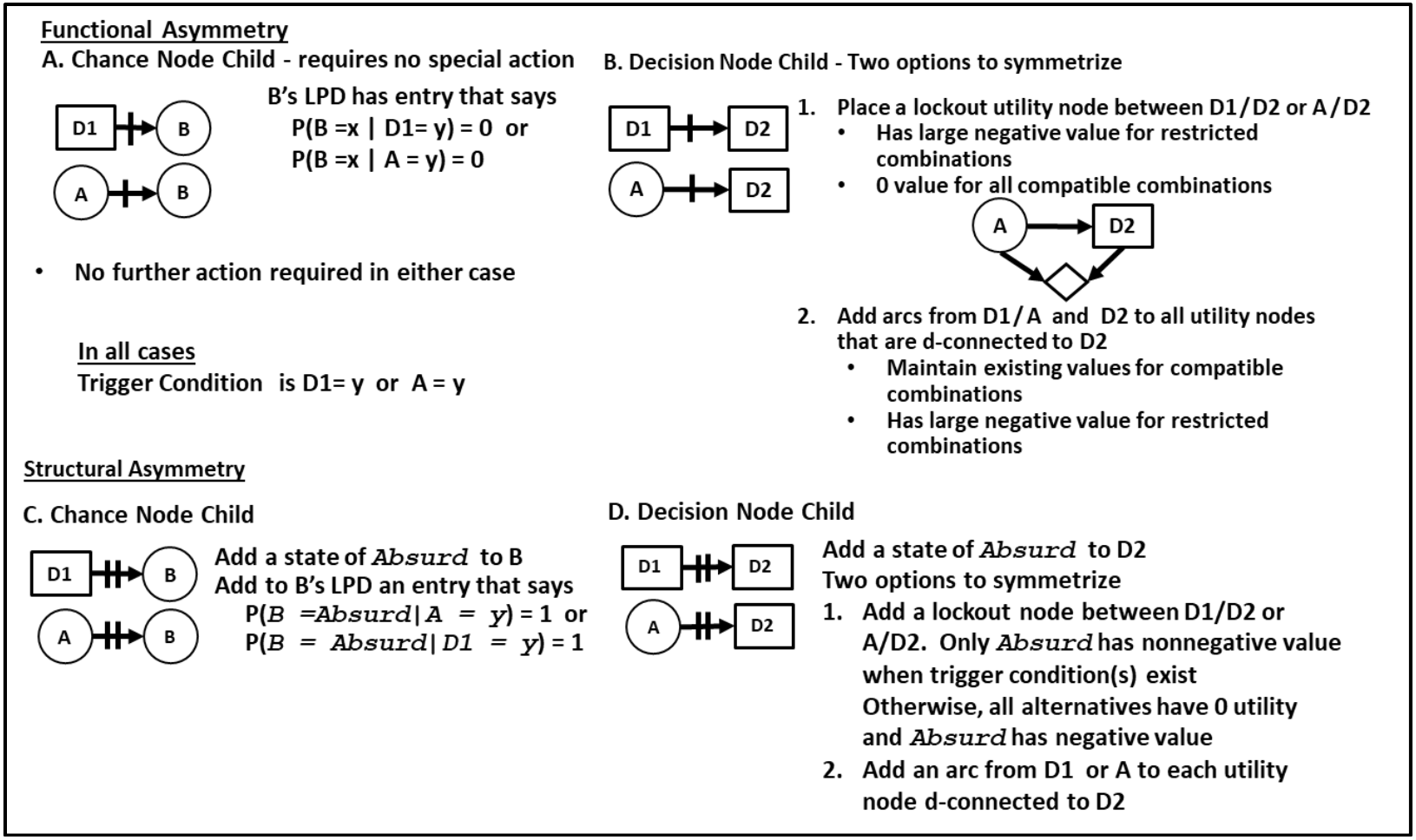


Figure 23: Basic additions to symmetrize decision graphs with functional or structural asymmetry

that is a triggering condition, the local utility distribution assigns a large negative value to that PC. This value (which is the least negative value required) is obtained as follows:

- Identify every utility node that is d-connected to the affected decision node (using a standard BN d-connection algorithm (e.g., F. V. Jensen and Nielsen 2007))
- For each such node, identify the largest utility value (call it x_i, i is an index number for each utility node that is d-connected to the affected decision node) and the smallest utility value (call it y_i)
- Calculate X = $\sum_i x_i$ and Y = $\sum_i x_i$. X is the highest possible utility value that any alternative in the affected decision node can get, and Y the is the lowest. There is no requirement that any decision alternative can achieve X or Y
- The lockout utility value is -(X-Y+1). The lockout utility value can be any number more negative than this.

Observe that since X is the highest possible utility value any alternative from the affected decision node can achieve, adding -(X-Y+1) to X means that the utility value of a restricted alternative when the triggering condition(s) exist is Y-1, a lower utility than any other decision alternative. This is sufficient for addressing functional asymmetry. An alternate approach is to add arcs from both the triggering nodes and decision node to each utility node that is d-connected to that decision node. For acceptable PCs, the existing values remain the same. Inhibited combinations have a large negative value, obtained the same way as for a lockout node. This approach adds significant size to the utility node table.

5.1.2.2 Symmetrizing Structural Asymmetry

There are two variants of structural asymmetry and the symmetrization approach differs between them. The first is when only a single node's influence is eliminated. This applies to information nodes preceding a decision node. A previous decision may have decided not to collect the information represented in the information node, or previous learned information identified that learning that information became impossible. In those cases, while the information node has no influence, the associated decision node and its descendant nodes still have influence. The second is when a sequence of nodes has its influence eliminated when the triggering condition(s) exist. In the granddad problem, if the girls choose to go to the amusement park, then the nodes associated with selecting an adventure park have no influence.

For any implementation, the first variant should be handled in the decision modeling process, and will not be part of the structural asymmetry handling algorithm described in Section 0 below. This variant has the form shown in Figure 24. For all

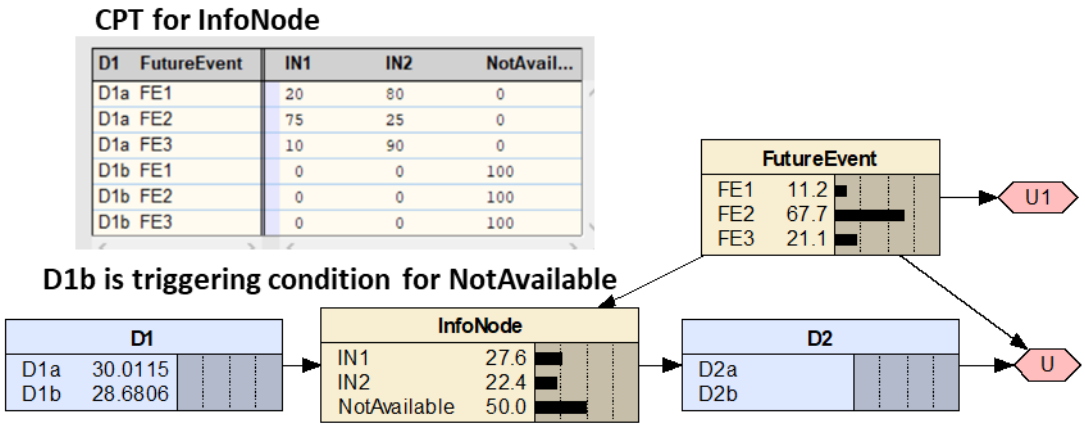


Figure 24: Form for a single node structural asymmetry problem

cases where information may not exist, the modeler adds a *NotAvailable* state (or equivalent) to the information node. When the triggering condition(s) exist, the information node LPD assigns a probability of 1 to the state *NotAvailable*. This approach can also be used for information nodes in decision graphs where there is a possibility that the requested information may not be received in time to support the decision. Time-sensitive decision problems often have this characteristic. Having *NotAvailable* as a state signals that the desired information may not be available at the time a decision must be made. The effect of the *NotAvailable* state (or its equivalent) is to leave probabilities unchanged in the decision graph. This is because of the way the CPT for the InfoNode is constructed. Observe that when the triggering condition D1b exists, every PC that includes D1b has the identical probability assignment. When this is the case, the probabilities of the other parent nodes will not change from their previous states. This has the same effect as if the InfoNode did not exist.

For the second variant of structurally asymmetric graphs, the decision graph is modified to add states / decision alternatives to the nodes affected by the asymmetry that signal that these nodes have no influence in the decision-making process when the triggering condition exists. This is defined to mean that the affected nodes do not change the overall utility values of optimum decisions in the unaffected portions of the decision graph, and that any decisions in the affected segments have an optimum alternative that signals that no decision was actually made, whenever the trigger conditions exist.

This dissertation uses an *Absurd* state to signal that the RV modeled in the affected node[12] has no effect on the overall outcome. This use of an *Absurd* state is derived from MEBN to signal that the concept of the attribute or relationship in the RV is meaningless, irrelevant, or contradictory for the particular application of that RV (Laskey 2008). Model implementations are free to choose any state name appropriate to their decision domain. To add the *Absurd* state to RVs modeling relationship (Boolean) or functional relationship, the ability to add categorical states to RVs with noncategorical states as discussed in section 4.6 above must be implemented.

To symmetrize structural asymmetry, one begins by adding *Absurd* to the affected node (see also Figure 23, page 113). In addition, when the affected node is a:

- Chance node - add an LPD script language line that states that if the triggering PC exists, the probability of *Absurd* is 1
- Decision node - add a lockout utility node with only the trigger node(s) and the decision node as parents. If a triggering condition exists, the value of any decision state other than *Absurd* is negative and *Absurd* is 0. If not, then the allowable states have a 0 value while *Absurd* has the large negative value
- Utility node – assign a value of 0 when all its parents are *Absurd*. See next paragraphs for utilities for PCs with at least one nonabsurd state.

The lockout utility negative value uses the procedure described in section 5.1.2.1. It is done only after all affected utility nodes have a value of 0 assigned as the value of *Ab-*

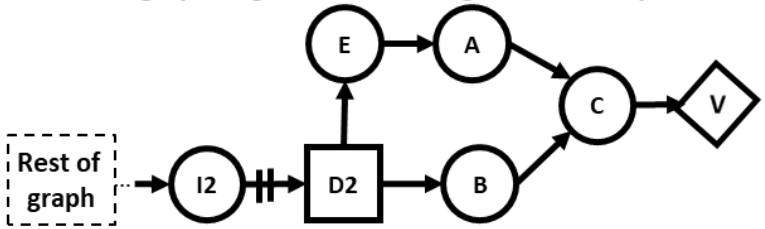
[12] The discussion here will use the terms RV and chance nodes interchangeably. Nodes may also include decision nodes and utility nodes

surd parents. This ensures that the lockout value will always force the non-*Absurd* decision alternatives to having a total utility less than zero when the triggering condition(s) exist. Since *Absurd* has a utility of zero, this will give *Absurd* the highest utility value whenever the triggering condition(s) exist.

Since this variant of structural asymmetry affects more than one node, additional action is required to propagate the *Absurd* state for as long as it is applicable. First, the *Absurd* state is added to every descendant node of the affected node. For each descendant chance or utility node, its LPD is modified so that the node's state is *Absurd* if every parent is in the *Absurd* state. If the descendant node is a decision node, any information node that has an arc to that decision node is a possible triggering node, with *Absurd* being the triggering condition. One adds a lockout node with that decision node and its information node(s) as parents. If all information nodes have state *Absurd*, the lockout node's utility values are set using the same procedure described in the second paragraph above.

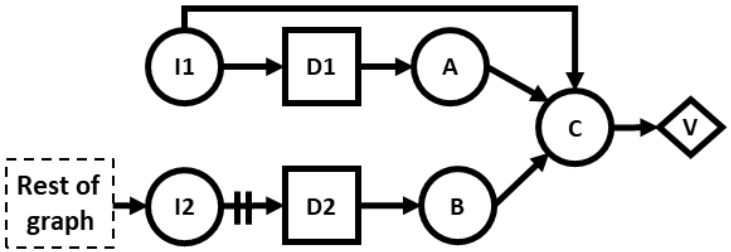
A merging node is any node which is in the path between the affected node and the utility node, and which has more than one parent. If none exist, the procedure described above is sufficient to symmetrize the network, as will be proven below. If merging nodes do exist, then additional action may be required. For merging nodes, three possible configurations are possible, as shown in Figure 25. In all configurations, it is

A. Isolated graph segment with merge node with parents that have a common affected node ancestor



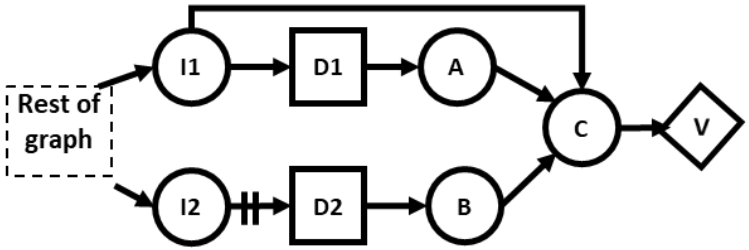
- All nodes get *Absurd* state and associated LPD or LUD language / lockout nodes.

B. Isolated graph segment with merge node C that has a parent that end in a root node



- All nodes on path from affected node to the utility node get *Absurd* state and associated LPD or LUD language / lockout nodes
- Merge node LPD has language that it is in the *Absurd* state if any parent is *Absurd*

C. Open graph segment with merge node C that has a parent with an independent chain to the rest of the graph



- Only affected node and nodes on path from affected node to the last node prior to merge node get *Absurd* state and associated LPD or LUD language / lockout nodes
- Merge node LPD will act as if node B does not exist if it has *Absurd*.

Figure 25: Structural symmetrization actions for descendent nodes

assumed that the decision sequence[13] is established and that a single information or decision node has been identified as the starting node. In each case, node C is a merging node. Observe that in cases A and B, all nodes below node I2 form an isolated graph segment. That is, all chains from nodes D1, D2, I1, A, B, C, E, and V to the rest of the graph go through I2 (chains through utility nodes are ignored). I2 triggers whether D2 and the path from D2 to the utility node has any influence. If the affected node is part of an isolated graph segment, then all the nodes in the isolated graph segment have no influence on the decision process when the triggering condition exists. This is because all information flows through the trigger node(s), which isolates the segment from the rest of the graph.

In case A, all paths to the merging node route through the trigger node, I2. Every node in the isolated graph segment is a descendant of the affected node, D2. In this case, setting D2 to *Absurd* will set every other node to *Absurd* as well. This is because each node has LPD language that sets its state to *Absurd* whenever all of its parents are in the *Absurd* state. In case B, the isolated graph segment includes a path of nodes that begin from a root node. In the Granddad problem, both the amusement park decision and the adventure park decision have root nodes. Because that root node path is isolated from the rest of the graph, it loses any influence when the affected node does. All nodes on the paths from the affected node to the utility node have an *Absurd* state. The LPD in the

[13] The decision sequence is the order in which decisions are made and states of information nodes are revealed. If order asymmetry exists, it must be resolved before the sequence can be established. Algorithms exist to establish the sequence (F. V. Jensen and Nielsen 2007). It is possible for the sequence to contain subsets of nodes where the sequence does not affect the overall utility computation. A sequence within those subsets may be arbitrarily made.

merge node is instead set so that the node is in the *Absurd* state if any parent is in the *Absurd* state. Because of this, the nodes in the path from the root node have no influence as well. No changes are needed in those nodes. In this case, the utility node has an LUD that for PCs with any parent in the *Absurd* state, the utility value is 0.

If there is at least one chain between a node's parents and the starting node that does not go through the affected node, then the path embedded in that chain can have an independent existence. Case C in Figure 25 shows an example. The nodes I1-D1-A-C-V have a chain to the rest of the graph that does not go through node I2. The graph segment is not isolated. If the trigger conditions exist, then only the affected node (D2 in this case) and any node whose parents all must be in the *Absurd* state when the trigger conditions exist have no influence (B in this case). The rest of the nodes have influence. In the Granddad problem, both the utility node *ParkExperience* and the decision node *Route* are non-isolated merging nodes. For determining whether a graph segment is isolated, chains through utility nodes are ignored. But a utility node may be a leaf node on multiple chains, like *ParkExperience*, making it a non-isolated node, even if all nodes on a specific chain to that utility node are isolated. It is isolated only if all chains to it have the same triggering conditions.

If Node C is a decision node, then a lockout node is instantiated between A, B, I1 and C. The LPD states that only if A, I1 and B all have the absurd state, then C has a large negative value for any state but *Absurd*. Otherwise, it is *Absurd* that has the negative value. If node C is a chance or utility node, the behavior for node C is determined by nodes I1 and A only. Node B has no influence. The modeler must specify in the LPD

what happens in node C when node B is in the *Absurd* state. In the MEBN / MEDG modeling framework, there already is a requirement to specify an LPD for when one or more parent nodes identified in the Template does not exist. Since the *Absurd* state means that the node has no influence, it is appropriate to use this existing LPD capability to specify the states of a merging node when one or more parents may be in the *Absurd* state. If the merge node is a utility node (as in the GrandDad problem), the LUD specifies that only the state of the non-absurd parents have influence. Those utility values may be different from the values when there are no parents in the *Absurd* state.

The effect of structural asymmetry is to eliminate the influence of a segment of the decision graph when a triggering condition exists. This means that none of the decision alternatives in that segment are selected, and the segment contributes nothing to the overall utility score of the decision graph. The above symmetrization actions enable that to occur while still allowing the use of standard decision graph solution algorithms. The use of *Absurd* as a state for the chance nodes allows the assignment of a probability to a state that signals the node is irrelevant. Adding *Absurd* as a decision alternative and using lockout utility nodes force *Absurd* to have the highest utility value (of 0) whenever the triggering condition(s) exist. The *Absurd* propagation process described above ensure that the non-lockout utility nodes that are descendants of the affected decision nodes get *Absurd* states in their possible parent configurations, and therefore assign 0 utility to those configurations. The *Absurd* propagation process addresses how to handle propagation in isolated graph segments and when merging nodes exist.

There is one remaining configuration that has not been explicitly addressed. It is possible to have a decision node d-connected to a utility node through a network chain such as is shown in Figure 24 on page 115. Observe that utility node *U1* is influenced by decision node *D1*. Assume *D1* is an affected node for triggering node not shown on the figure. The *Absurd* propagation process applies only to descendant nodes of *D1*. *FutureEvents* and *U1* are not descendants of *D1*. How is the influence of *D1* on *FutureEvents* and *U1* eliminated when the triggering condition exists? Observe that the propagation process gives *InfoNode* an *Absurd* state and modifies *InfoNode*'s LPD to assign a probability of 1 whenever *D1's Absurd* state is in the parent configuration. This has the same effect on *FutureEvents* as the *NotAvailable* state does. The probability distribution for different parent configurations of *FutureEvents* doesn't change whenever *D1*'s configuration is the *Absurd* state. In those case, D1 has no influence on *FutureEvents*. No additional action is needed to symmetrize the graph when the d-connection configuration in Figure 24 exists.

5.1.2.3 Symmetrizing Order Asymmetry

Order asymmetry exist when one has a set of decisions to make, and there is learning between subsets of decisions. This learning has the potential to give different expected utilities depending on the order in which the decisions are made. The decision process has two questions to address:

- In what order should the decisions be made?
- For each decision what is the optimal decision alternative?

Order asymmetry is visualized with an oval that captures the decision nodes, consequence nodes that signal the consequences of the various decisions, and utility nodes providing the cost of each decision. Each decision and consequence node within the oval has a CRT that provides needed information for symmetrizing the graph. The nodes within the ordering oval must meet the requirements in Figure 26.

The distinction between test and action decisions is important. Test decisions only result in information that changes the decision-maker's understanding of the probabilities of some world state. An action decision may resolve the problem. If it does, then no future decisions are needed. As a result, there can be two kinds of consequence nodes within the ordering oval – one that has stopping conditions, where an action decision resolved the problem, and one without stopping conditions.

The fundamental problem is that the standard decision graph solution algorithms requires a defined decision sequence, while order asymmetry says that determining the sequence is one of the decision problems. A solution is to change the question from

Design requirements

- All decisions are either test decisions, whose consequence is a test result that can change the probability of some world state, or action decisions, whose consequence may resolve the problem of interest
- All decisions made by the same entity
- Each decision has two or more alternative.
  - One alternative for all decisions is not to do anything regarding this decision. That alternative has the same name in all decisions
- Each decision is made once
- A consequence that can signal that the problem is resolved is said to have a stopping condition.
  - The state that signals the problem is resolved has the same name in all consequence nodes that have stopping conditions

Figure 26: Order asymmetry node requirements

"which decision should be decided first?" to "of all of the decision alternatives in all of the decisions, which one should be the first one selected?" One then repeats the question for all of the remaining alternatives, until one has N answers, where N is the number of original decisions. To do this, one replaces the nodes in the ordering oval with their supernode equivalent. A supernode is a decision graph node that results from merging the states / alternatives of a set of original nodes into a single node. There are four types of supernodes described below.

The decision supernode has all of the alternatives of the original set of decision nodes, except for the alternative "no decision" or equivalent (specified on CRT). Instead, it is replaced by the A*bsurd* state. Once created, N copies are made, where N is the number of original decisions. The copies have the same OV / entity as the original decisions (see requirement in Figure 26 that they are the same for all decision nodes). The supernode is given an additional OV, called an index OV, representing a class created for this purpose. It has N entities that are ordered. This OV distinguishes the decision supernodes from each other and sequences the supernodes.

If there are test decisions, then there are test consequence nodes without any stopping conditions. These are merged into a test consequences supernode. This supernode has all of the states of the various merged nodes. It has the decision supernode as a parent, and the collective set of nondecision-node parents found in the merged nodes. It merges the LPDs (in LPD language form) of each merged node. N copies are created, with the same OVs (including the index OV) as the decision supernode. Each copy is sequenced, and the ith test consequence supernode has the ith decision supernode as a par-

ent. The ith test consequence supernode has an information arc to the ith+1 decision supernode.

Likewise, if the CRT indicates that at least one original consequences node has a stopping condition, then a stopping consequence supernode is created. It merges all consequences nodes that have a stopping condition. It has all the parents that the original set of stopping consequences nodes had. Like the test consequence supernode, N copies are made and given the same OVs as the decision supernode. It merges the LPDs (in LPD language form) of each merged node. The ith stopping consequence node also has the ith decision supernode as a parent, and has an information arc to the ith+1 decision supernode.

Since every consequence supernode has a decision supernode as a parent, there can be test decisions that were not in the original set of stopping condition consequence nodes. These are assigned a non-stopping condition state in the LPD. Likewise, there can be action decisions that were not in the original set of test consequence nodes. These are assigned to a `NoInfoAvailable` state or equivalent in the LPD.

The original set of decision nodes each had a utility node that identified the cost of that decision. These utility nodes are also merged into a decision cost utility supernode. Like the other supernodes, it has the same expanded set of OVs as the decision supernode. It is also replicated N times. The ith copy becomes a child to the ith decision supernode.

If there are any action decisions in the set of original decision nodes, this means that something can be done within the ordering oval set of decision nodes to resolve the

problem, and all possible actions will be tried until the problem is resolved, or gathered evidence indicates it is impossible for the remaining actions to resolve the problem

There are two important constraints that must be enforced. The first is that for each original decision, only one alternative from that decision's alternative set can be chosen. Second, when a stopping condition is reached, any remaining decision supernodes need to be forced to the A*bsurd* state. For ordering problems with only test decisions, only the first constraint exists. The constraint is enforced by a lockout node. (N-1)* N/2 lockout nodes are created. Each has two decision supernodes as parents, and are assigned so that every possible pair combination of decision supernodes has a lockout node. The assigned lockout node values are shown in Table 5B (next page). Lockout node values are assigned depending on whether the parents chose alternatives from the same or different original decisions.

For action decisions, two types of lockout notes are created. The first is the same as used for testing-only decisions, but only (N-2)*(N-1)/2 are instantiated. They are assigned pairwise only to non-sequential pairs of decision supernodes. For the sequential pairs (DSN_i and DSN_{i+1}), a different lockout node is created. It has three parents: DSN_i, DSN_{i+1}, and CSN_i. It uses the lockout node value rules shown in Table 5A (next page). In addition to enforcing the condition that the alternatives must be from different original decision nodes, it also enforces the condition that if a stopping condition is reached, the following decision must be the `Absurd` alternative. It also enforces that `Absurd` cannot be the selected alternative if the stopping condition has not yet been met.

There is one additional change for decision supernodes that have action decisions. Because the cost of decision alternatives is usually specified as a negative number while the `Absurd` has a zero value, it is possible for the solution algorithm to select `Absurd` as the optimum decision alternative for the first decision supernode. To avoid this, the CRT can specify that a positive first decision is required. This means that `Absurd` is not a state for the first decision supernode[14].

Figure 27 shows the order symmetrization result for the Granddad problem. In looking at the decision graph, one sees that it has a similar structure as the original graph. However, the decision nodes are now sequenced, and each has all of the alternatives of

Table 5: Order asymmetry lockout node values

A. Lockout node values for utility node with a consequence node parent that has stopping conditions. Node also has two sequential decision supernodes as parents

Parent States			Value
D_i	Conse-quence	D_{i+1}	
Any	Stop Condition	Any Except Absurd	Neg #
Any	Stop Condition	Absurd	0
Absurd	No Stop Condition	Any	Neg #
Any except Absurd	No Stop Condition	Any alternative from same original decision as in D_i, or Absurd	Neg #
Any except Absurd	No Stop Condition	Not Absurd / Not from same original decision as in D_i	0

B. Lockout node values for utility node with two nonsequential decision supernodes. Also used for two sequential decision supernodes when there, is no consequence node with stopping conditions

Parent States		Value
D_i	D_{i+j}	
Any except Absurd	Any alternative from same original decision as in D_i	Neg #
Any except Absurd	Any alternative not from same original decision as in D_i	0
Absurd	Any Except Absurd	Neg #
Absurd	Absurd	0

[14] It is possible for a test decision only ordering problem to select none of the tests. This occurs when the cost of any of the tests exceed the expected utility of the results. In this case, a positive first decision is not required and the first decision supernode has an `Absurd` state.

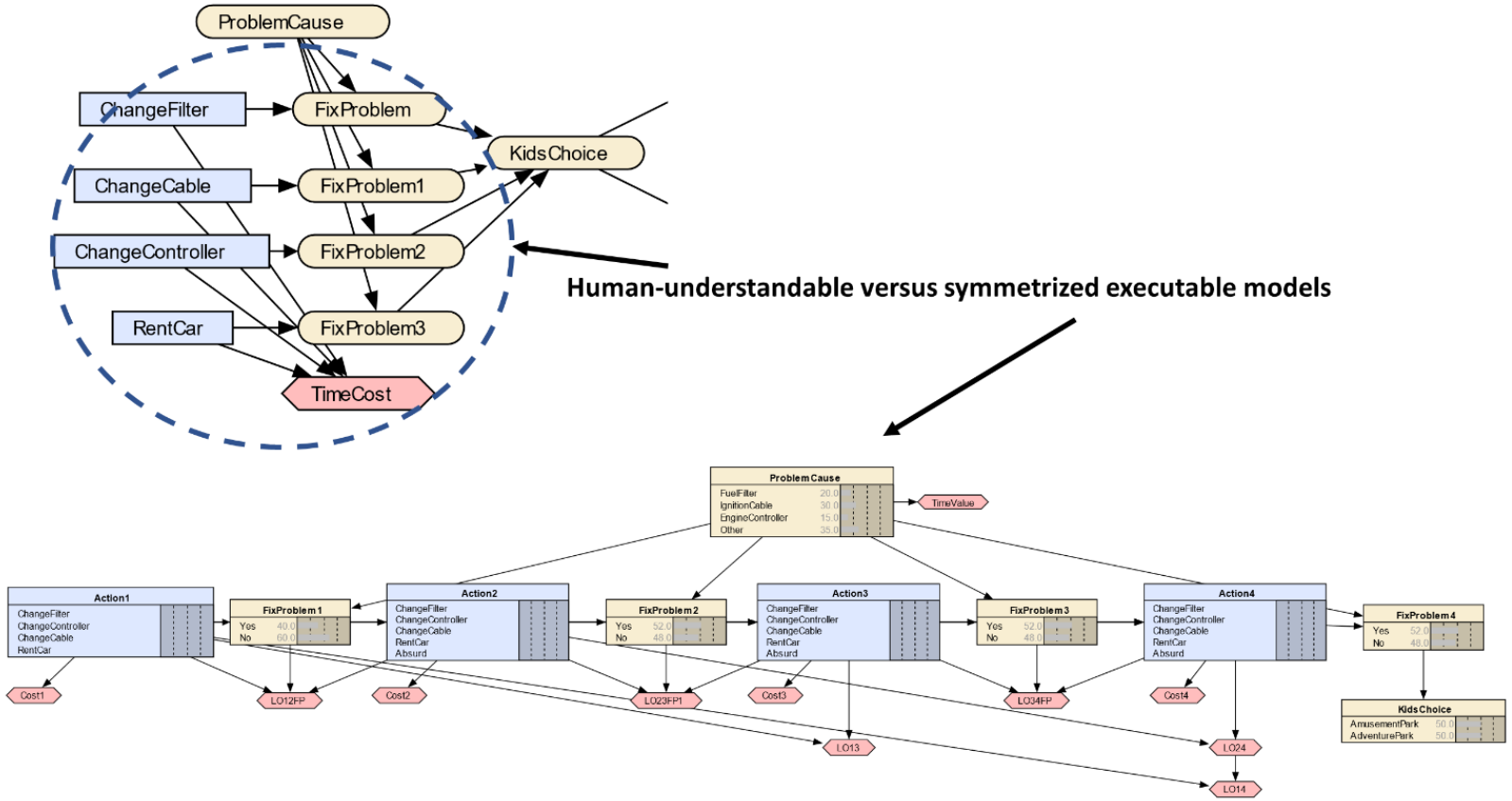


Figure 27: Order symmetrization of the Granddad problem

all of the original decision nodes. This allows a standard solution algorithm to test all possible combinations of allowable decision alternatives to determine both what is the optimum sequence of original decisions and what is the optimum decision alternatives for each original decision. The specific algorithms for automatically symmetrizing an asymmetric graph are in the next section.

5.1.3 Symmetrization Algorithms

This research created two algorithms to automatically symmetrize a decision graph – one to address order asymmetry, given in Figure 28 and Figure 29 below, and

Initialization

1. Conduct initialization per functional / structural asymmetry symmetrization algorithm
2. Identify all nodes and associated CRTs with the same problem number / name
3. For all nodes being merged into a supernode – if there are identical state names, each identical name is appended with the name of the node, including entity name, it came from.

Creating Decision Super Nodes (DSN)

1. Create a DSN that incorporates all of the decision alternatives in the original set of decision nodes
   a. Merge all decisions alternatives from the decision node into a single decision super node (DSN) except for any *Absurd* states
   b. Add an *Absurd* state to the DSN
   c. If an original decision node had more than one non- *Absurd* state, identify the alternatives that came from the same decision node
   d. Assign DSN name from CRT
2. Replicate the DSN N times, N is the number of original decision nodes
   a. Each DSN keeps the original ordinary variable (OV)
   b. Each DSN receives an additional OV representing a class with N ordered entities (OV is an index)
   c. DSN replicas are ordered based on the index order
3. If the CRT requires a positive first decision, `Absurd` is removed from the DSN replica with the first order index entity.
4. All information arcs that went into any of the original decision nodes now go into the first DSN. Per no-forgetting, they are known to all succeeding DSNs.

Creating a Utility Super Node (USN) for decision costs

5. Create a USN by merging the cost utility nodes from the original decisions Pj;lhop
   a. Replicate N times. Use the same OVs as for the DSN.
   b. Create a directed edge from DSNi to USNi
   c. Incorporate the utility node LUD information for each decision parent in the original cost utility nodes into the USN

Figure 28: Order asymmetry symmetrization algorithm (Part 1 of 2)

Creating a Consequence Super Node (CSN)

6. One or two CSNs are required. Identify using the consequence number / name from the CRT
   a. If two CSNs are created, note which one has the stopping condition
7. For each CSN, merge all states from the set of consequence nodes with the same consequence node number. Name the node using the distinct name identified in the CRT
   a. Add an Absurd state to the CSN.
   b. Add a directed edge from any parents external to the ordering oval for any CN merged into the CSN
8. Replicate the CSN N times, using the same OVs as for the DSN
9. Create a directed edge:
   a. From DSNi to CSNi
   b. From CSNi to DSNi+1
   c. If DSN_{i+1} does not exist, create an edge(s) to the nodes that are shown as child nodes outside of the ordering oval, as shown on the graph (it is possible that no such node exist)
10. Transfer the LPD language from each of the original CNs merged
    a. Change the name of the parent node to the name of the DSN
    b. Add LPD that if DSN_i = *Absurd*, CSN_i = *Absurd*
    c. If CSN is a test CSN, and DSN includes alternatives not in the original set of CNs, add state *NoInfoAvailable*, with probability 1 for those alternatives.
    d. If CSN is a stopping condition CSN, and DSN includes alternatives not in the original set of CNs, assign those alternatives to the non-stopping condition state with probability 1.

Creating Lockout nodes

12. If there is a CSN with stopping conditions, two sets of lockout utility nodes are created
    a. Create N-1 Sequential Lockout Nodes (SLN). N is the number of DSN. The SLNs have the same OVs as the DSN (The last index is not used)
       i. SLN_i has DSN_i, DSN_{i+1},and the CSN_i with stopping conditions as parents
       ii. The LPD is given in Table 5A
    b. If N > 2, create (N-2)(N-1)/2 NonSequential Lockout Nodes (NSLN)
       i. They have an OV from class created for this purpose. It has (N-2)(N-1)/2 entities. No ordering is required.
       ii. Each NSLN has a pair of nonsequential DSNs as parents. Collectively, they cover all possible nonsequential pairs of parents
       iii. The LPD is given in Table 5B
13. If there is no CSN with stopping conditions, only one set of lockout utility nodes are created
    a. Create (N-1)*N/2 Lockout Nodes. They have OVs from the same class as the NSLNs in 11.b above, but with (N-1)*N/2 entities
    b. Each lockout node has a pair of DSNs as parents
    c. The LPD is given in Table 5B

Figure 29: Order asymmetry symmetrization algorithm (Part 2 of 2)

one for the initialization and symmetrization of functional / structural asymmetry, shown in Figure 30. Both algorithms use the compatibility restriction table data to identify where symmetrization is needed.

Initialization

1. Identify the decision sequence (decision past) (Algorithm in Jensen / Nielsen 2007)
2. For each node in the decision sequence, check if the node has a compatibility restriction table
   a. Maintain list in decision sequence order of affected nodes and asymmetry types
   b. For nodes with order asymmetry, also include both problem number and supernode name
3. Create initial SSDG (Human-understandable model) with markings generated from list

Create Solution Version

4. Determine if order asymmetry exists. If so, run the order symmetrization algorithm for each problem number
5. For each node on the asymmetry list, determine if functional or structural asymmetry
6. If functional,
   a. add a lockout utility node with the affected decision node and the trigger node(s) as parents, as identified in affected node's CRT.
   b. Determine all utility nodes that are d-connected to the affected decision node
   c. Sum up best possible utility values and the worst possible utility values from all d-connected utility nodes. The negative lockout value is -(sum(best values) – sum(worst values) +1)
   d. In the lockout node, create LUD language that if a parent configuration (PC) has a CRT-identified restricted state, that PC is assigned the negative lockout value as the utility value.
   e. Otherwise, assign a "0" utility value
7. If structural, check if the descendant nodes from an isolated graph segment.
   a. Select each utility node on a path from the affected node.
   b. Search for a chain from the utility node to the starting node that does not run through either the affected node or any of the trigger nodes
      i. Any chain that runs through another utility node is ignored
      ii. If no chain is found, all of the descendant nodes and any nodes that end in a root node form an isolated graph segment.
      iii. For any utility node, if it does have a separate chain that does not run through any of the descendant nodes from the affected node, all descendant nodes but that utility node are isolated.
   c. If the nodes are isolated, add the *Absurd* state to affected node and every node descendant from the affected node in the isolated graph segment
      i. To the affected node and each descendant node, add chance / utility LPD language or a decision node lockout node to activate Absurd.
      ii. Decision nodes – algorithm for setting the lockout value is the same as in 6c, but is not run until all affected utility nodes have had the "0" utility value added (7.c.iv below)
      iii. Chance node LPD language includes (IF ANY paramsubset HAVE (parent1 = *Absurd* or…parentn = *Absurd*) [*Absurd* = 1]
      iv. Utility node LUD language includes (IF ANY paramsubset HAVE (parent1 = *Absurd* or…parentn = *Absurd*) [Utility = 0]
   d. If not isolated, add *Absurd* state to affected node and LPD language / lockout node to activate. Then add *Absurd* and LPD or LUD language / lockout node to every descendent node whose ancestor chains all go only through the affected node or trigger node.
      i. For non-isolated merge nodes with at least one parent that is not an affected path parent, modify the existing node LPD for missing parent to also apply when that parent has the *Absurd* state.

Figure 30: Functional / structural asymmetry symmetrization algorithm

5.2 Addressing Context Variation

Chapter 3 identified context variation as a significant factor within some decision problems. Context variation means that based on a specific context setting, any element within the decision model may vary between settings. An example of context variation is selecting a restaurant for a dinner. The purpose of the dinner affects the selection criteria and the utilities of each criterion. A business dinner has different emphasis on a restaurant's attributes than a romantic dinner. In any decision graph, one has two choices for modeling context variation. First, one could create a decision graph that incorporates each possible context variation and uses context selectors to identify which RVs and which parts of the LPD within those RVs are applicable to a particular setting. This can lead to large graphs that are difficult to follow, with significant information embedded in the LPD (including the utility node LPD). The second approach to modeling context variation is as a structural asymmetry problem. The context switching node would act as a trigger node identifying that a specific section of the graph is active and that the remainder have no effect on this particular decision. This approach reduces the size of the tables but can add a significant number of nodes to the graph, especially if there are a significant number of variations possible within the problem.

For MEDG, a third alternative exists. This is to enhance UnBBayes-MEDG's context variable (CV) capabilities. Chapter 2 identified that a context variable sets a condition that must be met for instantiating an RV in an MFrag. Context variables can also restrict which entities may be instantiated in an RV. Chapter 4 presented two UnBBayes-MEDG enhancements for handling context variables, both related to functional and rela-

tionship RVs. This section details a third enhancement, using an attribute RVs as context variables. Here, one has a context variable of the form *ContextSetting* = *StateX*.

For each possible value of StateX, a set of MFrags are developed that define the appropriate decision graph for that decision problem context. Figure 31 provides an MTheory for a simplified implementation of the restaurant selection problem. This MTheory assumes a variable list of restaurant list. The context switching node is *DinnerPurpose*, found in the Context_MFrag. It has two states: Business and Romantic. There is an MFrag for each context, with the context variable containing DinnerPurpose and its applicable state in each MFrag. In each MFrag, there are four resident nodes that have similar functions, but have distinct names. This avoids having RVs being defined in two different MFrags. Each MFrag assesses ambience differently, evaluating the choice based on having an ambience conducive to the context. In addition, each has a

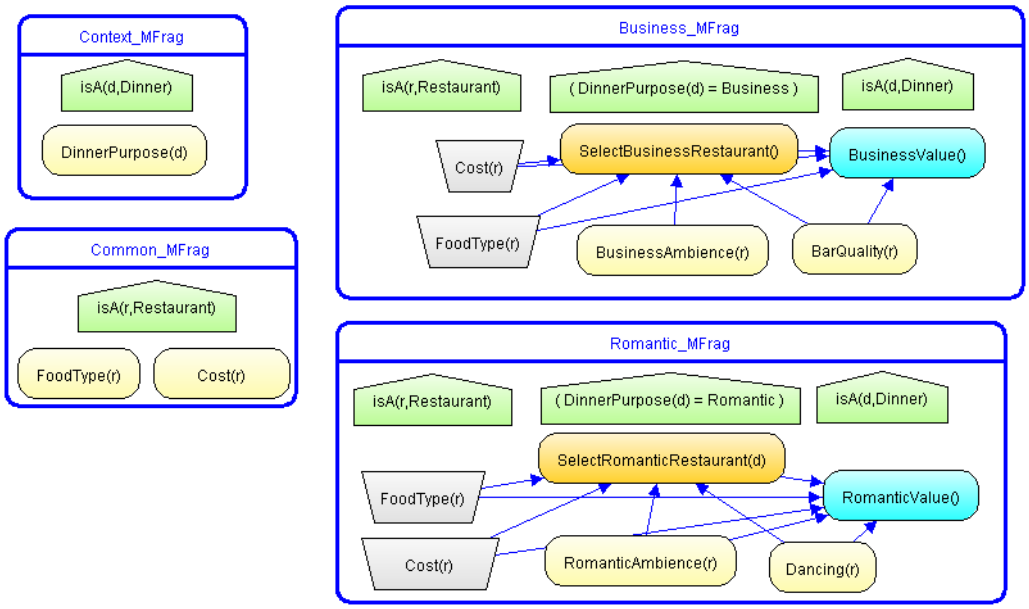


Figure 31: Restaurant MTheory showing use of attribute context variable to set context variation

distinct attribute. A restaurant that has a dance floor can be attractive for a romantic evening, while a good bar for an after-dinner conversation is a business meeting consideration. In addition, the utility values given to each attribute can vary between the two MFrags. There are two restaurant attributes that do not change between contexts: cost and food type. By placing them in a different MFrag and then using them as input nodes, they can be defined once.

The initial presumption is that the context setting is known, so that only one of the two MFrags is used. If the context is uncertain, one has context variable uncertainty. There is a difference between the uncertainty effect of an attribute CV and the functional and relationship CVs addressed in Chapter 4. Those CVs determine which entities get instantiated for a set of RVs within an MFrag. An attribute CV determines which MFrag from a set of MFrags gets instantiated. This means that if there is attribute CV uncertainty, every MFrag associated with that CV is instantiated. This makes the problem structurally asymmetric, with the attribute CV becoming the trigger node. The question becomes which RV(s) does the trigger RV trigger? The answer is developed when the MFrag is developed. The modeler identifies the affected node(s), and builds a CRT signaling structural asymmetry and that the CV node is the triggering node.

There are three modifications required to implement this enhancement. First, modify UnBBayes-MEDG to recognize and store an attribute CV. The current implementation can create an attribute-based context variable, but deletes the variable state when storing. Second, modify the construction algorithm to recognize the attribute CV and to instantiate the MFrag. Third, modify the construction algorithm to address attribute CV

uncertainty. If the attribute RV CV is uncertain, then the construction algorithm needs to search through the node(s) within the MFrags to locate the CRT that has the CV as a triggering node. If the CV is certain (100% in a state), then the portion of the CRT identifying the CV as a trigger node is ignored. If there is CV uncertainty, the construction algorithm instantiates the attribute RV, instantiates the nodes on each MFrag for which the attribute RV has nonzero probability, draws an arc from the attribute RV node to each of the affected node, and includes a visualization symbol on each arc that structural asymmetry exists. The structural asymmetry symmetrization algorithm is run prior to the solution process.

5.3 Modeling Variable Decisions

This section discusses the effects of modeling variable decisions. Decisions may vary in one of two ways:

- The alternatives for a specific decision may vary between instances on the problem. For example, alternatives for movie watching will vary from week to week
- The objects about which decisions are made may vary. For example, the number of threats facing a military aircraft self-defense system can vary, affecting the set of decisions to address those threats.

First-order expressive decision graph modeling is intended to model problems where these can vary. There are two primary modeling effects of allowing variable decisions:

- One must explicitly model those attributes of a decision alternative or decision object that affect the decision recommendations

- For decisions about varying decision objects, one must identify and model interactions between decisions.

Decisions with varying alternatives will model the alternatives as entities in a class. This allows the decision-maker to modify the alternatives before executing the model, by modifying the class entities. The key modeling addition is that for each possible alternative, the decision-maker must enter a finding with the relevant attribute values for each alternative. The criteria assess the relevant attributes of each decision alternative. To do this, the modeler must know what the value of each attribute is. When dealing with a fixed set of alternatives, the attribute values are known when the templates are developed. Sometimes, one does not need to explicitly include a decision graph node for each decision attribute. Rather, one can use that information implicitly in the LPD, stating if alternative X is chosen, the probability distribution of criteria Y is A. But when there are variable alternatives, this means one does not know what the alternatives are when the templates are developed. Rather, one knows what the relevant attributes are, and what their possible values are. These are modeled as attribute chance nodes. When the model is run, the decision-maker enters the possible alternatives into the database and enters a finding for each alternative's attribute node(s) specifying what the state of each attribute each. Figure 32 gives a simple example. The template for the example is on the left side. The node *Attribute1* (**Alt**) is used to specify the state of each alternative's *attribute1*. The right side shows the grounded model when the data base has three entities in the Alternative class, and a finding for *Attribute1* for each **Alt (Alt1**, **etc**.). The `Criterion_D1` LPD uses the select-one pattern with `Decision_D1` is the selector.

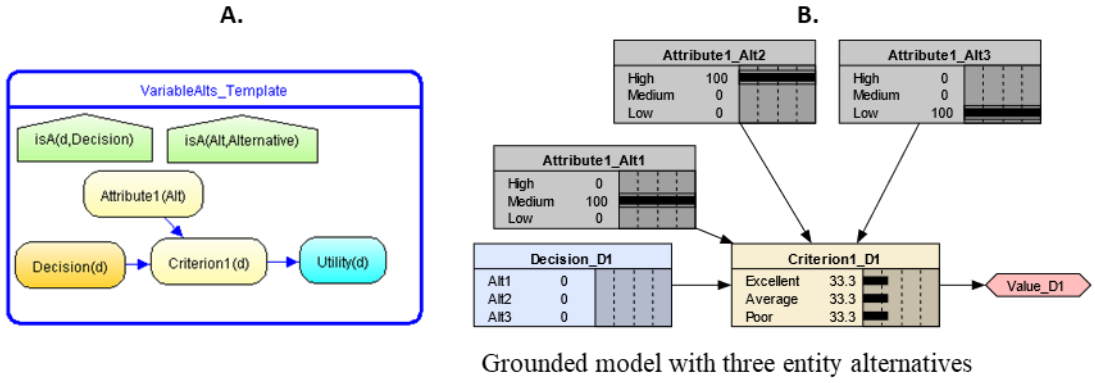


Figure 32: Template and grounded model for a simple variable alternative decision problem, showing how the attribute nodes are used

When dealing with a varying set of decision objects, one has two considerations. First, there often is some information about each decision object that is relevant to the decision. For decisions with a fixed set of alternatives or decision objects, these attributes may be implicitly modeled. For variable decision objects, they need to be explicitly modeled. Figure 33 gives a simple example for a three-object problem. The template in Figure 33 does not indicate how the decisions among the decision objects can interact with each other. The decision patterns in Chapter 3 identified three possibilities: independent,

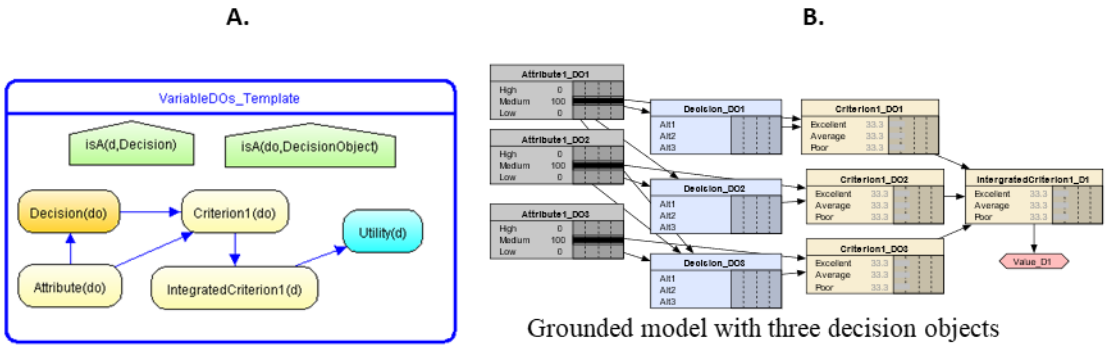


Figure 33: Template and grounded model for a simple variable decision objects problem, showing how the information nodes are used

integrative, and unordered sequential. If they are independent, the template in Figure 33 is complete. If they are integrative, this means there is some constraint among the decisions and their alternatives. This means there is functional asymmetry in the model, and a functional asymmetry CRT is required for each affected decision node. If there is an ordering with learning between decisions required, then the problem has order asymmetry and the items in sections 5.1.1 and 5.1.2 above are required.

5.4 Summary

This chapter identified the changes in template design, model visualization, structuring process and algorithms, and solution process needed to correctly model and solve problems with asymmetry. The research developed two algorithms applicable to any first-order expressive decision graph tool that collectively relieve the modeler from manually symmetrizing a decision graph, significantly reducing the chance for error. The research also developed a new attribute RV based context variable, along with the construction algorithm procedures for handling uncertainty in that context variable. This allows MEDG implementations to model context variation in an easy to understand manner, reducing model size and complexity. Finally, it identified specific modeling considerations for variable decision problems and described what needed to be done if these kinds of problems also have asymmetry.

CHAPTER 6
DECISION STRUCTURING KNOWLEDGE REUSE SUPPORT

Decision problem structuring is the process that transforms an incompletely defined decision problem into a form enabling choice (Dillon 2002). Structuring focuses on developing the decision information the decision-maker needs, presented in a decision model and supporting documentation. This arguable makes decision problem structuring the most important step in solving a decision problem. Yet there is a limited set of tools designed to assist structuring (von Winterfeldt and Edwards 2007). Knowledge reuse tools support a modeler / decision-maker with information from previous and related decision efforts. This would reduce structuring time and can improve the quality of the decision model. UnBBayes-MEDG has an infrastructure to support MFrag-level knowledge reuse. But to facilitate reuse, knowledge should also be at a higher level of generality, enabling the decision-maker to adapt it to the specific problem. This chapter develops a comprehensive decision knowledge reuse tool, called a decision template, that is usable with any decision modeling capability, and can significantly aid in structuring and modeling decision problems.

6.1 Structuring Products

Decision knowledge is captured through the decision structuring process's products. The literature identifies multiple approaches and specific steps for decision struc-

tures (e.g. Clemen and Reilly 2014; Grünig and Kühn 2013; Belton and Stewart 2010; von Winterfeldt and Fasolo 2009). The process can be divided into three main areas:

- Context identification: understanding the relevant situation, the structure of the decision maker, the internal factors of each stakeholder (often grouped by common interests), and the context differences identified in Section 3.2.2
- Decision problem framing: refining the felt need / issue that triggers a decision requirement into a focused problem statement and desired outcomes (the objectives) sufficiently clear to guide the remaining decision structuring actions
- Decision element development: defining and populating the decision elements supporting each specific decision that must be made.

The decision elements are the primary content for the decision model. The process potentially develops a number of products, many focused on the decomposition of problem elements to a level where they can be modeled. These products can be grouped into four categories:

- Situation models that identify cause and effect or correlations significant to understanding a decision problem - includes fault trees, event trees, consequence analysis, and Bayesian networks
- Objective hierarchies, which capture fundamental decision-maker objectives and then decompose them to lower-level objectives that are measurable – called a value tree if it has criteria and relative weighting of those criteria
- Decision outcome models, modeling the flow from decision alternatives through to the utility of the various consequences - includes ends-means diagrams, deci-

sion trees, and influence diagrams. If there is no uncertainty about the consequences, a consequences table maps alternatives to criteria

- Supporting tools, such as mind maps, Strength / Weakness / Opportunity / Threat (SWOT) analysis chart, and strategy charts.

The decision problem elements schema (per section 3.6) identifies a core decision model applicable to any decision problem that can be modeled as a decision graph. This captures the information flow from the decision and future events information through to potential consequences, criteria evaluating the consequences, and the utility of criteria levels. This flow is central in every decision pattern identified in section 3.4. Figure 34 maps the graphical structures listed above to the areas of the core decision model that they support. For example, a decision-maker's objectives / values decomposes to measurable criteria via an objective hierarchy / value tree, and these become the criteria - utility segment of the model.

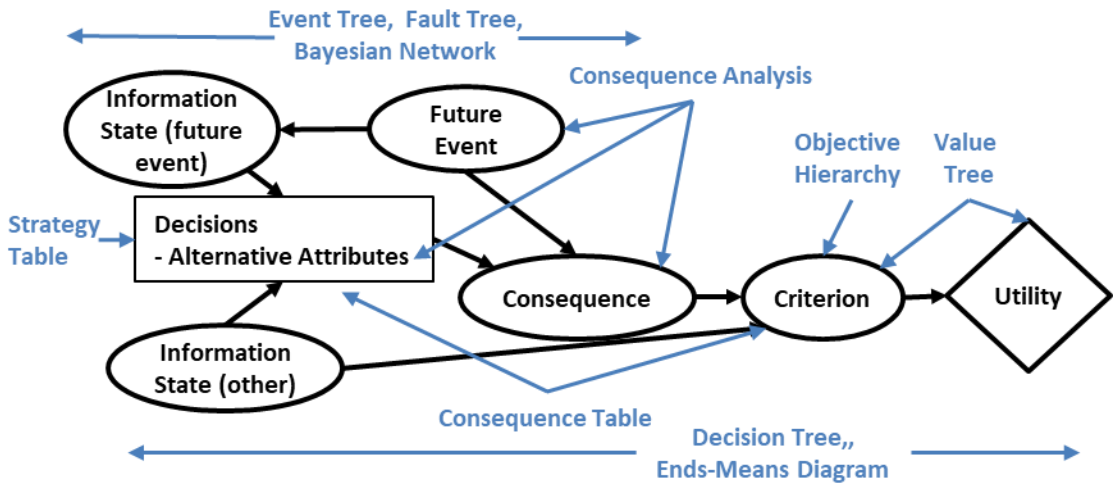


Figure 34: Supporting analytic decision models mapped to core decision model

A decision problem class has a common problem type and uses a common set of decision patterns. Problem type identifies the general objective, structuring starting point, problem focus areas, and core processes required to solve the problem. These provide a basis for identifying the general information needs of a problem from that class. The decision patterns from Chapter 3 and the core decision model shown in Figure 34 provide a starting point for the decomposition process. It is possible that a problem class may have two or more problem types within it. For example, a decision model may include both a determination as to the state of the world, and an appropriate response to that determination.

Finally, decision structuring can also be viewed as an uncertainty reduction process. At the beginning of the process, much about the problem is unknown. Through information development and discovery activities, much of the uncertainty is reduced or eliminated. Sometimes, significant uncertainties remain, and the structuring process should provide options for addressing the remaining uncertainty.

6.2 Decision Templates

The knowledge reuse support tool developed here is called a decision template. It is an organized collection of decision problem information usable by a decision modeler in structuring a decision problem. It focuses on two things:

- Identifying information items to be developed during the structuring process
- Providing examples of the kinds of information applicable to a problem class or a specific problem.

Decision templates can vary significantly in their information content. There are three levels of specificity: generic, problem class, and problem specific. The generic decision template is a shell that identifies the types of information that the structuring process for any decision problem needs to develop. The format is shown in Table 6. It identifies the information content of a decision template and its general form, but it does not specify a detailed format. It is divided into three parts. The first part covers the decision context. The most important piece of information in decision context is what is the top-level objective(s) that this decision problem supports. This is not necessarily the objective of the decision problem itself but may be a higher-level objective that is driving the need for this decision problem. The next information item is the context focus – what the primary driver for this problem is. It can be externally driven by the situation / decision alternatives, or by the decision-maker's objectives and values. It is important for the modeler to know the context variation. The contents of a decision model can vary due to a range of possibilities for a situation's purpose or for a decision-maker's standpoint. The model needs to adapt as these vary. An important item is the context's dynamics – the speed at which the context changes. There are two important time frames: how stable is the context until the decision is made, and how stable is it until the decision can be implemented. These two considerations determine the amount of time available for making the decision.

Next are four stakeholder internal factors: the stakeholder's core values, the related plans that are affected or influence this decision, the resources available to a decision-maker and the decision- maker's situation-relevant knowledge. The stakeholder's core

Table 6: Decision template general organization and contents

<table>
<tr><th colspan="2">Decision Template for XXXX</th></tr>
<tr><th colspan="2">Decision Context</th></tr>
<tr><td colspan="2">Top Level Goal(s): General statement of the decision problem and overall objectives it will support</td></tr>
<tr><td rowspan="2">Decision-maker Structure: Identifies decision makers, decision making process, and stakeholders to be consulted during the structuring process.</td><td>Key Stakeholder Factors:</td></tr>
<tr><td>- Values: Fundamental principles / priorities that guide stakeholder in decision-making</td></tr>
<tr><td>Context Focus: Situation / Alternatives or Objectives / Values</td><td>- Related Plans: Identifies other activities that affect or be affected by this decision</td></tr>
<tr><td>Context Variation: Within a problem or class, range over which context elements may vary</td><td>- Resources: For bounding feasible alternatives and identifying resource considerations tracked in the decision model</td></tr>
<tr><td>Context Dynamics: Identifies how stable the context is and over what timeframe it is likely to change.</td><td>Situation: Overall external environment and its effects on areas of interest to the decision-maker and stakeholders</td></tr>
<tr><th colspan="2">Decision Framing</th></tr>
<tr><td colspan="2">Problem Statement: Succinct specific problem description</td></tr>
<tr><td rowspan="2">Applicable Decision Patterns: Per figure 6. May have added problem class or specific problem details</td><td>Problem Type: Per table 2</td></tr>
<tr><td rowspan="2">- Starting Point: Per table 2, with added information to describe the problem-specific Starting Point</td></tr>
<tr><td>Relevant Situation Aspects: Situation elements that will be included in model</td></tr>
<tr><td>Relevant Stakeholder Factors: Objectives / values / plans driving evaluation criteria</td><td>- Focal Point: Per table 2, with added problem details</td></tr>
<tr><td>Key Uncertainties: Identified uncertainties to be addressed in analysis process</td><td>- Core Processes: Per table 2, with added problem details</td></tr>
<tr><th colspan="2">Decision Problem Elements</th></tr>
<tr><td colspan="2">Architectural Decomposition: Problem decomposition to a model supporting level</td></tr>
<tr><td colspan="2">Decomposition supporting products</td></tr>
<tr><td>Objective Hierarchy / Value Tree: Includes the decomposition from top level objectives to specific criteria. Includes criteria weighting (i.e. trade-offs between criteria) and decision-maker's risk attitude</td><td>Strategy Chart: Identifies the specific decisions that are part of the problem, their alternatives, the relevant attributes of each decision and prohibited or mandatory combinations of decisions / alternatives</td></tr>
<tr><td>Information State / Future Events Development: Identify and assess future events and situation factors that could affect the consequences of various alternatives</td><td>Problem Class Specific Products: Products identified as useful for specific problem classes of for a specific decision problem</td></tr>
<tr><td colspan="2">Problem Class Prototypical Elements</td></tr>
<tr><td>Future Events: Prototypical future events common / significant to this problem class</td><td>Consequences: Prototypical consequences common / significant to this problem class</td></tr>
<tr><td>Criteria: Prototypical criteria widely used in this problem class. May also include low-level objective functions</td><td>Utility consideration: Prototypical utility factors (Risk, weighting) prototypical to this problem class. May also be decision-maker / stakeholder focused</td></tr>
</table>

values are the basic principles that guide both people and organizations in making decisions. These shape the decision objectives, the acceptable alternatives for achieving those objectives, and the process used in decision-making. Second, related plans need to be identified and examined for their effects on the decision problem. In many cases, the decision alternatives are modified to account for related plans. The third stakeholder factor to consider are available resources. They are important for three reasons. First, they determine how much problem structuring can be done. Second, available resources set the upper limits on allowable alternatives. An alternative that cannot be implemented due to lack of resources is worthless. Finally, some resource decisions may need to be part of the decision model. In sequential or hierarchical models, the decision-maker may need to keep track of certain resource status.

The final decision-maker factor is the degree to which the decision problem's situation is known. In some problems, determining the situation is the decision problem. The final context factor element is the nature and structure of the decision-maker: who is it; if not a single person, how is the decision made; and if there are affected parties that do not have a formal decision making role, how do they provide input.

The next part of the decision template covers the decision framing. Here, the most important information element is the problem statement developed by the decision-maker. The next information element is the problem type. As discussed in Chapter Three, the problem type establishes three very important structuring considerations: the starting point from which structuring starts, the focal point which guides the structuring process,

and the core processed used in structuring. There are four other major decision framing information items:

- The basic decision pattern applicable to this problem.
- Relevant aspects of the decision situation - these are either information states that will be known before the decision must be made, or future events that will influence the consequences of decision
- Relevant aspects of the stakeholder factors that need to be incorporated into the decision model
- Key uncertainties to be addressed.

The third part of the decision template identifies the information necessary to decompose the problem and build the decision model. One develops a top-level problem architecture based on the starting point and the applicable decision pattern. These then guide the decomposition process. For different problem classes, there can be a general decomposition approach, describing how typical problems are decomposed to their elements. These are captured in the architectural decomposition.

For each decision in the model, the core model guides the development of the model elements needed to show the effect of a decision on the decision-maker's objectives. These elements may or may not be directly migrated into the decision model. In developing the decision model, one looks at the core decision model and divides it into two parts. The first part is captured in the information state / future events nodes. These nodes can be a very elaborate network that develops the information the decision-maker needs and / or identifies the precursors for future events that affect the decision process.

This network may be built using tools such as event trees, fault trees, or Bayesian networks. For example, for a restoration problem type, one begins with the problem symptoms and the objective of restoring something to a previous state. The decomposition process then identifies tests and actions necessary to do so. These are captured in a model that identifies the decision actions and information states of the problem. The template section of information states / future events provides knowledge on this area.

The second part decomposes the path from the decision and its alternatives to the utility nodes. For any decision problem, the decision and utility nodes are the minimum required nodes. The additional nodes are used in more complex cases to show the chain of reasoning of how a decision alternative, interacting with future events, would receive a specific utility value. The consequences node is most appropriate when a future event may interact with an alternative to produce some outcome of interest in decision-maker. Like the information state and future events nodes, the consequence node is often a product of a detailed analysis of the causes and effects between the future event and the decision alternative. A consequence analysis can provide insight on how this interaction can unfold.

The consequence node then feeds one or more criteria nodes that establishes the decision-makers evaluation of that outcome. The objective hierarchy / value tree provides supporting knowledge for these model elements.

The last two decision template items are the strategy charts and the problem class specific products. For some problems, developing decision alternative are a significant effort. This is especially to for creative problem types. A strategy chart can identify pos-

sibilities for the decision-maker. The last section is used for knowledge not covered under the other sections, but which should be considered.

For a problem class template, the last section provides prototypical examples of the major decision problem elements. The decision template provides a comprehensive yet flexible format for capturing and conveying decision knowledge for use in decision problem structuring.

6.3 Uncertainty Support

An important structuring consideration is addressing the remaining uncertainty. Chapter 3 identifies four levels of uncertainty. Some decision problems have no significant uncertainty, which allows straightforward determination of the consequences of any decision alternative. Many decision problems have first order uncertainty about the occurrence of future events, information states, or consequence levels. Here, the probabilities are known with certainty. A decision theoretic approach is generally considered adequate for first-order uncertainty. Second-order uncertainty occurs when probabilities are not known with certainty, value levels are uncertain, or some relationship is only vaguely known. Ignorance exists whenever one cannot identify the information necessary to develop one or more aspects of the decision model. Second order uncertainty or ignorance means that one cannot confidently understand the risks associated with a decision alternative. This often makes decision-makers unwilling to decide. When significant decision-relevant information is only vaguely known or missing, the decision model should be modified to address those uncertainties.

Table 7 identifies options to address such uncertainty. It incorporates options identified by Etzioni (1989), Lipshitz and Strauss (1997), Regan et al. (2005), Scherpereel (2006) and Stirling and Scoones (2009). These options are generally not as

Table 7: Decision structuring considerations for addressing uncertainty

Criteria Changes	
Incorporate robustness criteria	Assess options on their robustness to uncertainty variations
Incorporate resilience criteria	Assess options on their ability to recover from adverse consequences
Decision Alterntive Changes	
Robustness	Explicitly develop alternatives that are robust to identified uncertainty
Resilence	Explicitly develop alternatives that can rapidly recover from possible adverse consequences.
Tentative / reversible options	Develop options that may or may not be implemented, or that can be easily undone if circumstances warrant it.
Preparatory options	Develop options that allow provide capability to respond to unanticipated negative developments (e.g., put forces on the alert, leave some resources unused).
Variability control options	Develop options that focus on controlling the range of specific uncertainties
Decision Evaluation	
Alternative evaluations	For significant probabilistic uncertainty, use evaluation techniques such as minimax or maximin
Model Structure Changes	
Decision staggering	Incrementalize decision implementation so that effects can be assessed and future increments adjusted
Fractionalize decisions	Divide decision problem into multiple semi-independent decision problems that can beassessed and implemented seperately
Contingency decisions	Include contingency paths in the decision model for possible future events or consequences
Problem Bounding	
Avoidance	Avoid options with significant uncertainty. Bound decision problem to avoid aspects of significant uncertainty
Tranfer	Transfer the risk from the uncertainty to another entity (e.g. insurance)
Adjust time horizon	Improve predictability by shortening the problem time horizon to avoid time-dependent uncertainties
Rescope to exploratory	Reframe the problem to focus on resolving key uncertainties. Includes efforts to establish cause-and-effect relations, understand complex interactions, and understand new phenomena

desirable as being able to eliminate the uncertainty entirely. But for some decision problems, one must either accept that uncertainty and address it, or avoid the decision problem entirely. The first set of options assess alternatives for their consequences across the range of uncertainty. It uses criteria for robustness (ability of an alternative to deliver satisfactory results across a wide range of uncertainty) and resilience (ability to recover at from an adverse outcome) as a significant part of the evaluation. Sensitivity analysis is often used to assess robustness. Resilience is often evaluated by the time or effort required to recover from an adverse consequence.

The second set of options modify the set of alternatives under consideration to include options that specifically address the uncertainty. These include specifically developing alternatives that have a high degree of robustness or resilience. Alternatives that are easily reversible or tentative provide decision-makers options to readily change course should future events so dictate. One can also include alternatives that are preparatory or contingent. This includes alternatives that enable future decisions and establish contingency reserves. The last alternatives-based uncertainty approach is to develop alternatives specifically focused on eliminating the uncertainty. For example, if a source of supply is an uncertainty, developing an internal ability to provide that item is an uncertainty elimination action.

A third option to mitigating uncertainty is to change the decision evaluation approach. Techniques such as minimax regret or maximin are specifically designed for making decisions under significant uncertainty (Clemen and Reilly 2014). Fourth, one can change the structure of the decision model to address the uncertainty. The decision

itself may be staggered (alternatives broken into parts, and implemented incrementally, with information development and reassessment between increments) and / or fractionated (divided into multiple subdecisions that address different parts of the decision problem). One can also develop contingency paths, with a branching set of decisions and results that would occur if some future condition were to occur. The fifth set of options is to rescope the decision problem boundaries. The decision problem may be more narrowly focused, to avoid significant uncertainty areas. One may seek to transfer the risks of the decision to another party (e.g. via insurance or market options). Or one may shorten the time horizon which the decision is expected to affect, reducing time-induced uncertainties. Finally, if extreme uncertainty / ignorance exists, the decision process refocuses to decisions that aim at understanding the factors driving the uncertainty and learning about new options once those factors are understood.

In a world where problem complexity is increasing and significant uncertainties exist that cannot be resolved or significantly reduced, the decision-maker requires options to address these uncertainties as part of the decision-making process. The above list consolidates a range of possible options that have been identified in the literature.

6.4 Evaluation of Decision Template Value

To assess the value of a decision template to a model developer, an experiment was conducted at George Mason University (GMU). A graduate level class on heterogeneous information fusion and decision support had a course project requirement to develop a fusion / decision support system. The system supports a fighter aircraft pilot by fusing the information received from various aircraft sensor systems about threats to the air-

craft, and then makes recommendations to the pilot about the appropriate reactions to those threats. The system is implemented as a decision graph. The system architecture naturally decomposes to a fusion element and a decision support element. The benchmark model has 43 nodes, with 22 nodes in the fusion element and 21 nodes in the decision support element.

6.4.1 Study Setup

Each student received an extensive project background information document (not the decision template) with sufficient information to understand what was being modeled (scenario and background material), develop the model, and derive the conditional probability tables and the utility node values. In addition, the students each received a modified decision template package. The design document had a significant amount of information that would be included in a problem-specific decision template. The student's template excluded that information. It was developed as a specific decision problem level template. It showed a complete decomposition from the high-level system diagram to the individual decision graph nodes used in the system. In addition, supporting documentation graphic structures were developed including an objective hierarchy, decision list, consequence table and a threat response table. Two decision template versions were created, one for the fusion element and one for the decision support element.

The class was divided into six teams, each with two or three people. Half of the teams received the fusion element decision template; the other half received the decision support element decision template. Essentially, each team received a completed model

without probability or utility tables for half of the required system. Students were asked to do two things:

- Track the amount of time they spent on the project, separated into four categories: general preparation, fusion element development, decision element development, and testing and report writing
- Complete a seven-question evaluation after they turned in their projects.

The evaluation had two sections. The first were four questions focused on the decision template to determine whether the students thought that it was:

1. Clear and understandable
2. Useful in completing the project
3. Whether it contributed to understanding the overall project requirements.
4. Whether it made a significant difference in the amount of effort the team had to expend on each element.

Each question had a 5-point scale, with a verbal descriptor for each score. A score of 1 means that the decision template rated poorly for that question, 3 was an "average" or "some" value, and 5 indicated the template rated very well.

The second section had three questions, again on a 5-point scale, that assessed the overall project.

5. The project's perceived level of difficulty (1-very easy to 5-very hard)
6. The overall quality of the information received in the project - both the background information document and decision templates (1-very incomplete to 5 – complete and comprehensive)

7. Whether the project is useful in developing data fusion skills (1-not useful to 5-very useful).

Students could also provide written comments if they so desired.

The experiment was executed following a study plan that was reviewed and approved by the GMU Institutional Review Board. Study participation was voluntary and was not a grading factor. The course instructor was not present or involved in study-specific interactions with the students. Nothing was shared with the instructor until after final grades were submitted. Copies of the evaluation and decision templates, additional project information, and additional data analysis are in Appendix C.

6.4.2 Study Results

Of the 13 students in the class, 12 participated in the study. The basic evaluation scheme was twofold:

- Compare the amount of time students spent on the element for which they had a decision template versus the amount of time on the element for which they did not. Since the model itself was roughly balanced between the two elements, it was estimated that they should take about the same amount of time on each if they did not have the decision template
- Assess how the students rated the decision templates on the four criteria, and then to determine if there were any correlation between the decision template criteria scores and the project assessments.

6.4.2.1 Data Analysis

Nine of 12 students provided time data. Data from two students was very general (e.g. more than 1 month / 5-6 hours a week) and was unusable. Data from a third student was suspect due to team dynamics issues and was not used. Of the remaining six, data was received from four students who received the decision element template and two who received the fusion element template. Three decision element students were on the same team, while the two fusion element students were on the same team. The time data is shown in Table 8 below. The student identifier indicated which template they received (D – Decision, F- Fusion).

There is a question about the data from D3 – D5, who each reported identical time spent on the project. The team was very coherent and appeared to spend significant amounts of time together on classwork (not just this project). It is possible they recorded the time mutually.

The questionnaire data is in Table 9 below. The group average and standard deviation for each question are shown at the bottom. In the analysis that follows, no data points were eliminated for being outliers. However, there are two individuals for whom there are data questions. For D6, there is one item in the data that may be an error. The

Table 8: Time data received

	Person (times in hours)					
Work Area	**D3**	**D4**	**D5**	**D7**	**F1**	**F2**
Prep	3	3	3	3	10	2
Fusion Element	20	20	20	5	5	5
Decision Element	9	9	9	2	15	20
Integration/ Reporting	4	4	4	2	0	20
	Same Team				Same Team	

Table 9: Questionnaire results

	Template Questions				Project Questions				
	1	2	3	4	5	6	7	Average Template	Average Project
Student									
D1	4	5	4	4	4	3	4	4.3	3.7
D2	4	5	5	4	4	4	4	4.5	4.0
D3	4	5	5	4	5	4	4	4.5	4.3
D4	4	5	5	5	4	3	5	4.8	4.0
D5	4	3	4	5	4	4	4	4.0	4.0
D6	5	5	5	1	5	5	5	4.0	5.0
D7	5	5	4	4	3	3	4	4.5	3.3
F1	3	3	4	2	3	3	3	3.0	3.0
F2	5	4	5	4	3	4	4	4.5	3.7
F3	2	3	4	4	5	2	3	3.3	3.3
F4	1	3	2	1	5	4	2	1.8	3.7
F5	-	-	-	4	3	4	4	4.0	3.7
Average	3.7	4.2	4.3	3.5	4.0	3.6	3.8	3.9	3.8
Std Dev	1.3	1.0	0.9	1.4	0.9	0.8	0.8	0.9	0.5

Average by Template Group		
Template	Average Template	Average Project
Decision	4.4	4.0
Fusion	3.2	3.5

student entered “5” for every question except question #4, where a “1” was entered. So the student said that the template was clear and understandable, was very helpful in completing the project, contributed significantly to understanding the problem, and then said the template made it harder to perform for the decision element (to which their template applied) than for the fusion element. Looking at the way the question was worded, it is possible that the student misinterpreted the question and answered it as if they were evaluating the model element for which they did not have a template. Student F4 identified that there was a specific circumstance that made completing the project difficult. Key results from the data:

- 78% of the individual template ratings were positive (ratings of 4 or 5)
- Nine of 12 students (75%) agreed that the templates made it either easier or much easier to develop the model

- Nine of 12 (75%) of the averaged template ratings had averages of 4 or higher, indicating that the template was understandable and helpful in completing the project.
- Eight of 12 (66%) of students rated the project as hard or very hard. None found it easy
- Eleven of 12 (91%) students found the project material (background information and template) at least adequate. Seven (58%) said it was at least mostly comprehensive and clear.
- Nine of 12 (75%) students found the project useful for developing fusion skills.
- Difference in average ratings between students that received the decision template than those that received the fusion template. The average template rating was 4.4 for students who received the decision template vs 3.2 for the ones who received the fusion template, and the average project ratings were 4 versus 3.5.

Six correlation analyses were done, comparing project scores to the average template scores and the different project score elements to each other. All showed a nonzero trend line, but in five cases, the R^2 score was less than 0.1. (all analyses results are in Appendix 4). One did, as seen in Figure 35. The number next to a dot indicates that more than one answer fell on that chart point. The chart shows a significant correlation between the students' assessment of the value of the project in learning fusion / decision support system technologies and their average on the template rating. In looking at the data, the nine students that had average template scores of 4 or higher also rated the project value as a 4 or 5.

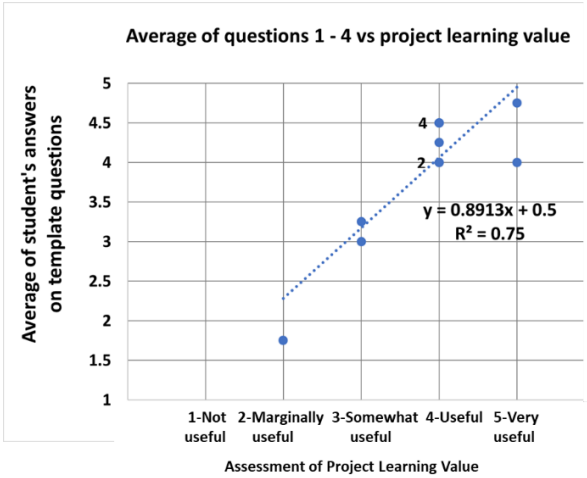


Figure 35: Correlation results for the project's learning value and the average template score

6.4.2.2 Discussion

There are three items of interest in the data. First, while the time data has significant shortcomings, there is some indication of the potential time savings. Accepting the time data as is, one can see that the duration for the element for which the team members received a template was about 55% - 71% less than the one for which they did not. The unweighted average reduction by team (and half team in the case of D7) is 62%.

Second, the differences in ratings between the students who received the decision element template versus the fusion element template was surprising. The going in assumption was that two elements were roughly equal in difficulty. The rating difference was for both the template and the overall project. It is especially surprising because the fusion element students had a lower average rating on project difficulty than the decision element students – 3.8 versus 4.1, 5 is "very difficult". The project difference is primarily driven by lower assessments of overall project information quality and project learning value. There are several possible explanations. One is that the templates were not equally

informative about their elements. While possible, it is doubtful because both gave the same level of detail about the decision graph design for that element. Second, the assumption of rough equivalence in complexity of effort may have been incorrect. This could be due to two factors. One is it is possible that the inherent complexity of the decision element was greater than the fusion element. The other is that the students' relative experience level was greater for the fusion element than the decision element. In looking at the course schedule, one sees a significant difference in class time spent on fusion (6 lessons) versus spent on decision support (3 lessons). In either case, it is possible that students who got the decision element template received more value from the template than those who received the fusion element. This could also affect their assessments on quality and overall learning value. This would support the idea that a decision template is especially useful when dealing with a decision problem class for which one has less familiarity.

The third area is the effect of the two sets of scores that Section 6.4.2.1 identified had possible issues. The analysis was redone the score changed from 1 to 5 from the one student and without the data from the other. The overall rating averages increase, but it doesn't significantly change the assessments. The fusion element average score increases, but there was still a difference in average scores between the two elements. The correlation analyses were also redone without those scores. There was a significant change in only one correlation. The r^2 score for the project quality information vs project learning value jumped from 0.09 to 0.33.

Overall, the study results indicate that a significant majority of the students found the decision templates aided their ability to complete a challenging project within a short time frame (six weeks).

6.5 Summary

This chapter defines decision templates, a tool for enabling decision knowledge reuse among decision analysis efforts. The chapter first maps the contents of widely used decision modeling tools to the core decision model defined in Chapter 3. Then, it takes the key information items determined by the Chapter 3 differentiators and the core decision model contents and integrates them into a usable form, the decision template. A simplified version was created to support developing a fusion / decision support system class project; a significant majority of the students found the template useful in completing the project.

CHAPTER 7
FUTURE RESEARCH AND SUMMARY

7.1 Areas for Future Research

There are at least three significant avenues for follow-on research to expand capabilities beyond the levels described here. One is to expand dependency modeling to three and four parent types. As decision problems become more complex and the number of interacting RV parent types increases, one can expect additional LPD behavior beyond the current capabilities. As the number of RV types increase, so does the count of meaningful ordinary variable combinations. This also opens the opportunity for a cross disciplinary exploration of the chains of interactions between entities that have meaningful dependencies (see Appendix A, section A.2 for a discussion of entity variable combinations), which in turn could require new LPD capabilities. Research in this area benefits from expertise in knowledge representation, artificial intelligence rule-based modeling, semantics, and epistemology.

Another useful line of follow-on research is to develop an asymmetry capable solution algorithm, especially for ordering problems. For ordering problems, standard solution algorithms can generate very large utility tables during the solution process, possibly exceeding the memory capacity of the computer. Adding lockout nodes can increase these table sizes even further. The proposed asymmetry solution algorithms in the literature preprocess the graph into a tree or lattice structure that divides the graph into smaller

subgraphs. They then process the subgraphs in steps. One shortcoming this research noted in those algorithms is the need for extra steps to address merging random variables (Figure 25, page 119), not accounted in the algorithms.

A third avenue of follow-on research is improved support for decision problems with second-order uncertainty or areas of ignorance. While it is unlikely to develop a comprehensive decision theoretic capability for that level of uncertainty, some research has already been done for decision-making under extreme uncertainty. The insights from this line of research could be mined for identifying additional capabilities to allow a decision graph to provide at least some support to modeling and solving these kinds of problems.

7.2 Summary of Research Completed and Contributions to the State-of-the-Art

This research extended the state-of-the-art in decision-making support by achieving its two research objectives: enhance first-order expressive decision graph tool capabilities and improve decision knowledge reuse capabilities in support of decision structuring. Chapter 1 identified that a key objective of the decision-making process is to develop a decision model that provides the decision maker an organized, readily understandable presentation of the relevant decision information. The quality of the decision model depends both on the capability of the modeling tools and the available information related to the decision problem. Decision graphs are identified as a powerful modeling tool, capable of representing most of the widely used graphical analytical structures used in decision analysis. MEDG is a recently developed decision modeling framework with first order expressivity. Theoretically, this gives it the capability to model almost any problem

executable on a computer. But the initial implementation has several significant limitations. In addition, the structuring process that develops the decision model is not well supported with tools for decision knowledge reuse. These limitations are not unique to the MEDG implementation.

To enhance the decision graph tool capabilities and to improve decision problem knowledge reuse capabilities, this research undertook four complementary research efforts. To understand the needed capabilities any decision graph modeling tool should have, the first research effort explored decision problem differentiators - the factors that define the envelope of required modeling capabilities. Four differentiators were identified. The first is context, identified in the literature as a major decision problem factor. Modeling context variation is an important capability. But there were shortcomings in the way context was defined and this research developed a refined decision context model that focused on the interaction effects between the decision situation and the decision maker's internal factors. The refined model also defines knowledge categories useful for knowledge reuse.

The second differentiator is basic problem type. The literature review uncovered a problem type differentiator. The original concept identified three basic problem types. Each type differed on its overall objective, the structuring starting point, the structuring focal point, and the core processes used in structuring. These are significant elements of information that should be included in a knowledge reuse tool. Follow-on research added two additional types to cover the range of decision problems found in the literature.

The third differentiator is the decision problem's sequencing of decisions and learned information – its pattern. The literature both explicitly and implicitly described multiple decision patterns. This research organized them into a taxonomy, using the presence / absence of decision learning within the model and flow structure as the major distinguishers between patterns. Decision learning is the defining element for decision asymmetry and the effects are widely discussed. The pattern taxonomy developed here is the first known taxonomy to use it as a distinguisher.

Uncertainty is the fourth differentiator. It is especially important for many problems today where the level of uncertainty has moved beyond first order probabilistic uncertainty. The key effect of uncertainty is on the structuring process. The more significant the level of uncertainty is that cannot be resolved during the structuring process, the more the structuring process changes focus from finding an optimal solution to mitigating the effects of the uncertainty. Identifying and providing uncertainty mitigations options for use in decision structuring is valuable information for a knowledge reuse tool. *The refined set of differentiators are a contribution to the decision modeling state-of-the-art.*

The second research effort enhanced the local probability distribution (LPD) language capabilities, which enhance any first-order expressive probabilistic modeling tool. The initial research uncovered a significant problem with the lack of LPD capability for handling of parent RVs whose states are entities rather than a fixed list. Rather than narrowly focusing on fixing that limitation, this research undertook a comprehensive exploration to determine what kinds of LPD capabilities any first-order expressive modeling capability needs. It developed a new approach for examining how parent random varia-

bles interact with each other, both structurally and in the local probability distribution, to define the probability of the states of the child RV. Called dependency modeling, it focuses on random variables that model the knowledge representation concepts of attribute, function, and relationship and how they interact with each other and with different combinations of the ordinary variables (the entity placeholders). These knowledge representation concepts are used in modeling many domains of human activity, making the dependency modeling results widely applicable. Dependency modeling is the first known approach for exploring first-order expressive modeling capabilities at the knowledge representation level. In this case, dependency modeling created and evaluated 37 specific interaction cases. The analysis of the results identified a set of LPD behaviors that provide a robust capability to any first-order expressive modeling tool and includes ten new LPD development language capabilities that significantly extend the range of problems that can be modeled. In addition to identifying LPD changes, the interaction case analysis uncovered eight structural patterns that are useful in first-order expressive modeling. Two patterns provided a solution for a long-standing UnBBayes upgrade request. *The first-order expressive LPD behaviors / enhancements and modeling patterns expand the range of tools designed for first-order expressive probabilistic modeling and are this research's second and third contribution to the decision modeling state-of-the-art.*

The third research effort focused on developing enhancements for the decision graph modeling tool capabilities, resulting in three specific enhancements that collectively are this research's fourth contribution. The first are two algorithms to automatically symmetrize asymmetric decision graphs. Differentiator analysis identified that asym-

metry is present in many important decision problems. Asymmetry makes additional demands on the modeling and solution process. This research adopted an effective asymmetry visualization approach found in the literature. The solution process requires either an algorithm that operates on an asymmetric graph, or requires graph symmetrization prior to using standard algorithms. No widely used asymmetric graph solution algorithm exists, so symmetrization is the selected course. Symmetrization adds additional nodes and states to a graph, which can clutter a human-understandable graph. The selected approach is to allow the human-understandable graph to remain asymmetric with selected annotations that explicitly identify to the decision maker the types and locations of asymmetry present in the problem. Prior to running a solution algorithm, two symmetrization algorithms developed by this research are run to symmetrize the graph. This allows both the benefit of a human-understandable graph and the use of standard solution algorithms.

The second enhancement expands UnBBayes-MEDG's capabilities to efficiently implement context variations. It allows the modeler to develop multiple MFrags for a decision problem that has different context possibilities (e.g. selecting restaurant for a romantic dinner versus a business dinner) and uses an attribute RV as a context variable for selecting the appropriate set of MFrags. The present implementation does not allow use of attribute RVs as context variables. This enhancement is available to any modeling framework that uses a context variable approach. The third enhancement describe for model builders the additional demands that variable entity decision problems make on the modeling process. *Collectively, these enhancements improve decision modeling tool ca-*

pabilities, improving decision support capabilities and are this research's fourth contribution to the decision modeling state-of-the-art.

The fourth research effort's contribution is a knowledge reuse support tool, called a decision template. It is an organized collection of decision problem information usable by a decision modeler in structuring a decision problem. It focuses on two things:

- Identifying information items to be developed during the structuring process
- Providing examples of the kinds of information applicable to a problem class or a specific problem.

Decision templates incorporate the decision information identified in the differentiator analysis and the decision elements schema development. I conducted a study to evaluate the decision template's usefulness. A graduate course on fusion and decision support had a class project to develop a fusion / decision support system, modeled as a decision graph. The project naturally decomposed to a fusion element and a decision support element. A decision template was developed for each element, and each student team received one of the two templates. The basic evaluation approach was to assess both quantitatively and qualitatively the benefit of the template on the modeling process by comparing effort on the element for which a template was received versus the one for which it was not.

Nine of 12 participating graduate students (75%) agreed that the templates made it either easier or much easier to develop the model. The time data collected during the study had significant shortcomings; it provides some support that the template for that element reduced development time by 50 – 70%. Finally, there was a strong correlation

between the students' positive assessment of the decision templates and their positive assessment of the overall value of doing the class project. In summary, the study found that the decision templates significantly aided this group's ability to complete a challenging project within a short time frame (six weeks). *The decision template is this research's fifth contribution to the decision modeling state-of-the-art.*

In conclusion, this research provides five contributions that advance the state-of-the-art in first-order expressive decision modeling:

- Developed set of decision problem differentiators characterizing decision problems on factors important to decision modeling and knowledge reuse
- Created ten LPD language capabilities that significantly expand the range of problems that may be modeled by a first-order expressive modeling capability.
- Discovered set of eight modeling patterns useful in any probabilistic first-order expressive modeling
- Created three enhancements specifically focused on decision graph modeling, improving decision model visualization, context variation support, and variable entity modeling.
- Developed knowledge reuse support tool, the decision template, and demonstrated that the concept was viable and useful.

The research effort also identified three lines of follow-on research that could enhance decision modeling capabilities and explore issues of significance to the decision-making community.

APPENDIX A
DEPENDENCY MODELING

This appendix describes the dependency modeling process, the approach used, to identify the range of needed LPD behaviors. This process was developed to provide a t comprehensive exploration of how different types of parent RVs interacted with each other to establish the child RV's LPD. This identifies the full range of needed LPD language enhancements for any first-order expressive probabilistic modeling tool. This provides confidence that these tools can address a broad range of problems. The results are described in Appendix B. Two complementary items are also discussed. First, the value of expanding the LPD language's use of various probability distribution approaches became apparent and is discussed. Second, constraints are important in first order expressive modeling. How to incorporate them is also discussed.

A.1 Dependency Model Development

Dependency modeling explores the range of interactions between different combinations of types of parent RVs to understand the demands made on the LPD language. As covered in chapter 2, RVs model an entity's attributes, relationships, and functional relationships. This leads to three RV types:

- Attribute random variable (A-Type), with a single OV, model attributes. For each entity in a class, this unary RV assigns a probability to each possible state of an attribute for that entity.

- Relationship random variable (R-Type) with two OVs. It implements relationships such as *isFriendsWith(**x,y**)*, read "x is Friends with y". These use Boolean states.
- Functional random variable (F-type), model functions, and has two entity variables. One is the OV, the entity that is the subject of the RV. Since an F-type RV has entities as states, there is a second variable, called a state variable, that represents the states of the RV. This state variable is used in dependency modeling.

The RV *MachineLocation(**m**)* is a F-type RV. It models a functional relationship between a specific machine entity and the room it is located in. When dependency modeling requires knowing what the state variable is, the RV is annotated as *MachineLocation(**m**)/**r***. It is possible for the same variable to be an OV for one RV and the state variable for another RV. OVs are placeholders for entities as specific class. There is no limit on an RV's arity.

The model builder may use multiple OVs to model complex relationships (e.g. *Purchased*(**person,item, date**)). In this effort, RV arity is limited as shown above. Modeling more complex relationships is future work. In a first-order expressive modeling template, a class may have multiple OVs. If two RVs in a template have the same OV, this signals that they are intended to model the same entity. If they have different OVs from the same class, the entities may be the same or different. Context variables may be necessary to enforce the desired distinction (($\mathbf{x} \neq \mathbf{y}$) means that the entity in **x** cannot be the same as in **y**).

This exploration focused on combinations of RV types. This means they can represent a wide range of problems and domains. Models explored the range of possible dependencies among and between attributes and relationships. The models are templates where an RV has a single parent, and where it has two parents. A parent in a template can instantiate multiple RVs in the grounded model. Since there are three RV types, there are nine possible models with a single parent, and eighteen possible models with two RV parents, as shown in Figure 36. In the models, the child RV (also called the target RV – this is the one for which the LPD is defined) is always RV3, and A / F / R identify the RV type. In the two parent cases, the numeric identifiers for the parents (e.g. F1, R2) are reversible. There is no requirement for RVs to be different. In the two parent-type models, if both parents have the same RV-type (e.g. an F-type RV), the two RV's may be the same or they may be different[15].

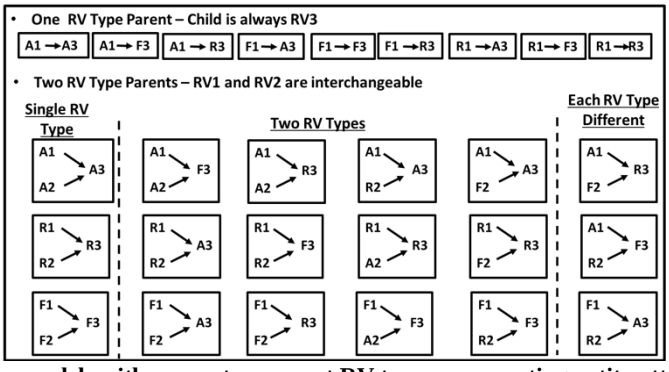


Figure 36: Dependency models with one or two parent RV types representing entity attributes (A), functional relationships (F) or binary relationships (R) in the Template

[15] If they are the same RV, they must have different OVs, with constraints added that they cannot be the same entity.

A.2 Effects of Different Variable Combinations

The dependency models in Figure 36 does not include OVs or state variables (SV). Different variable combinations may be used in the same model. For the single parent models, there can be from 1 to 4 distinct OV/SVs, depending on RV types used. The two parent models can have up to 6 distinct OV/SVs (from every RV has the same OV/SV to each RV has a different set of OV/SVs).

Different variable combinations result in different grounded models, so it is important to understand how the construction algorithm uses the template and database to create the grounded model. We assume the use of a query-specific construction algorithm, rather than creating a grounded model able to address any query. The query is of the form "What is the state of RV3 for specific entity "a" or "a and b" (if RV binary)?" The query-specific construction algorithm begins with the query RV(s). One instance of RV3 is instantiated with the queried entity(ies). Then, the algorithm looks at the OVs in each parent node in the template[16]. If the OV of an RV is the same as for the child RV, it is bound. The algorithm creates one instance of that RV, with the same entity as the child RV. If the OV is different than the child RV's OV, it is free[17]. One parent instance of that RV is instantiated for every entity in the OV's class. When the parent is a binary R-type RV and the child is unary, then if they have a common OV, the R-Type parent will instantiate as many RV instances as there are entities in the free OV's class. The common

[16] SVs representing F-type states are ignored in the construction algorithm.

[17] An template may have context variables that limit the entities that are present in a network. In general, the analysis here assumes no such context variables are in the template, unless there are two OVs from the same class and the model structure does not distinguish them.

OV is bound and will have the same entity as the child RV. If an R-type RV has two OVs different from the child RV's OV, the algorithm will instantiate m*n instances, where m and n are the cardinalities of the two classes represented by the OVs. For a template, the algorithm completes when it has instantiated every RV in the template with the number of bound and free OVs.

Different combinations instantiate different parent sets for the same basic dependency model. Figure 37 below gives an example, where there is the same dependency model (A1, F2, A3) but the order of the variables for the F2 RV type is inverted. In this modeling, A(**x**) means **x** is the OV of the RV A, while F(**x**)/**y** means **x** is the OV in the RV's name and **y** is the SV of the state entities. Figure 37 models the effect of a room's temperature on a machine's operational status. A room's temperature affects a machine's status only when the machine is in that room. Here, one is uncertain as to which room a specific machine is in. The database has four rooms and four machines. One queries on the status of machine M1. `RoomTemperature` and `MachineStatus` are both attribute type random variables and are identical between the two examples. The F2 random variable has its OV and state variable classes inverted between the two examples. The

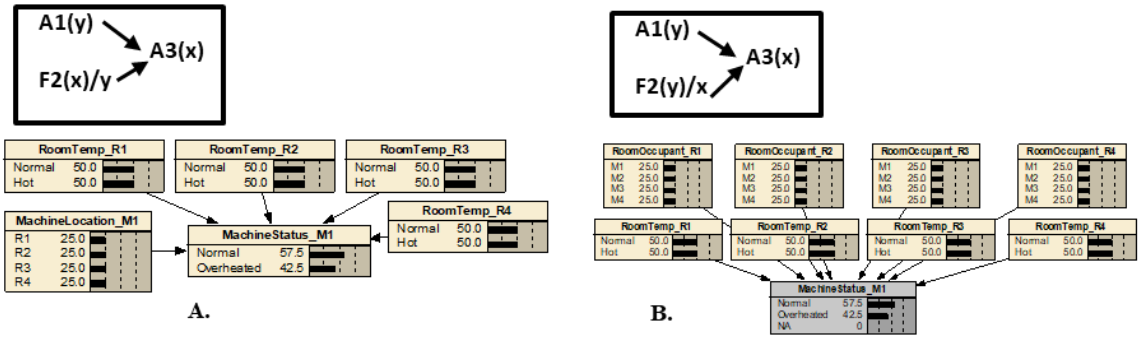


Figure 37: Inverting the variables in RV F2 results in a different pattern

actual F-type random variables (*MachineLocation* and *RoomOccupant*) are inverses of each other. This results in two different parent sets. In Figure **37**A, the F-type RV has the same OV as the child. One instance of that RV is instantiated. In Figure **37**B, it has a different OV, so four instances are instantiated. These are two different structures[18].

Different variable combinations also affect the understanding of what is being modeled. There are collectively 39 distinct variable combinations for the one-parent type dependencies and 414 for the two-parent type dependencies. However, finding meaningful interpretations for many of these variable combinations is difficult, especially when each RV has ordinary / state variables different from every other. Consider the following instantiation for the template in Figure 2 on page 31: If "Kathy has good teaching abilities" and if "Kathy teaches Mark", then "Mark has good student performance." Consider now that the template has four OVs (say **p1,p2,s1,s2**), instead of two. An instantiation then could be If "Kathy has good teaching abilities" and if "Paulo teaches Mark", then "Rommel has good student performance." Both examples have the same dependency model, but only the first one seems to make sense. In the second case, one does not see the connection between the entities being modeled. It is possible to identify circumstances where there exists enough shared background knowledge that such an entity combina-

[18] In this example, there is a unmodeled requirement that a room can have only one machine in it. It was not modeled because it adds additional nodes that clutter the point of the model and does not affect the LPD under discussion. It could be modeled by including the remaining three MachineLocation nodes (for machines m2, m3 and m4) and adding a constraint node enforcing the requirement that each machine must be in a separate room. In Figure 11B, the constraint is modeled, using an embedded constraint approach. Note the state NA with 0 probability in RV MachineStatus_M1. See Appendix A.4, page 179 for constraint discussion.

tion would make sense. But most often, one expects there is an interaction between the specific entities identified in the entity instance. This led to a search for variable combinations that carried meaningful semantic information.

In looking at the possible variable combinations, it became apparent that the concepts modeled by RV combinations make sense when they had common entities between them. The search generated four rules. First, cases where every RV has the same variable are eliminated. They create very simple models (only one instance of each parent-type is instantiated) and require all relationships and functional relationships to be reflexive. They do not require new LPD behaviors beyond what other models require. Second, the child RV must share at least one variable with at least one parent RV. This enables a meaningful connection between the concepts modeled in the parent and child RVs. Third, the one-parent-type models are limited to two variables. The second rule already limits models involving attributes to have no more than two variables. For one-parent-type models involving a mix of relationships and functional relationships, rule one allows combinations of three variables. But they require additional constraints and care in modeling that has been deferred for future work. For instance, one could have an R1 → R3 combination of R1(X, Z) → R3(X, Y), that is, some relationship R1 influences a relationship R3. In general, it is difficult to find meaningful dependencies if X, Y and Z all come from different classes. There are possibilities if, for example, Y and Z each come from a different subclass of a specific superclass. One can also have the case where two variables come from the same class, say Y and Z. In both cases, one would need to take care in modeling to avoid circularity and many cases require additional constraints to ensure cor-

rect modeling. The fourth rule limits two-parent-type cases to no more than three variables.

The effect of these rules is to force entity sharing between RVs. Three variable sharing schemes were identified. The first was an enabling scheme. One parent RV, modeling a relationship or functional relationship, shares variables, one with the other parent and one with the child. This licenses the dependency between the other two RVs. Both examples in Figure 37 are examples of this scheme. The second variable scheme is a pass-through scheme that applies to relationships or functional relationships. It occurs when there is a dependency between three relationships where the one parent has entity **x** in a relationship / functional relationship with entity **z** and the other parent has entity **z** is in a relationship / functional relationship with entity **y** (either the same kind of relationship or different). There then can be a meaningful relationship between **x** and **y**, captured in the child RV. If the three relationships are identical, and the entities are from the same class, then this is a transitive relationship. However, there is no requirement that they are identical. An example is if *hasChild(***x,z**), and if *isParentOf(***z,y***)* then *isGrandparentOf(***x,y***)*.[19] Pass-throughs also support dependencies between parent relationship RVs and a child attribute RV. The third variable scheme is a common variable. Here, the parent RVs share a variable in the same OV position, enabling a dependency for a relationship for the other two variables.

[19] There is no claim that the dependency modeling approach establishes this dependency. Rather, the modeler identifies that this specific dependency exists in the domain being modeled.

Applying these four rules to the dependency models shown in Figure 36 resulted in 41 cases. Then, there was an attempt to build a scenario for each case. In four cases, viable scenarios could not be found. Almost always, those cases required additional relationships to license the dependencies between the OVs. For instance, no meaningful scenario could be found for cases where one or two attribute-type parents influenced a child attribute RV without assuming a relationship between the OVs. This required that a third parent type be added. In addition, no scenarios could be found for two F- type RVs influencing an A- type RV, or an F- type RV and A-type RV influencing a child F-type RV. This reduced the number of test cases to 37. Exploring these test cases led to the results in Chapter 4 and Appendix B.

A.3 Canonical Probability Distributions

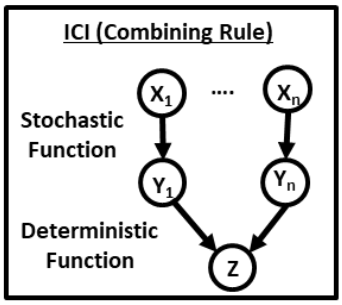
In a BN with discrete states and a fixed number of parents, the LPD is expressed as a conditional probability table (CPT). As a first-order expressive probabilistic model can vary in the number of parents, the LPD is instead a function that specifies, for each RV instantiated, how to assign a conditional probability based on the number of parents and their specific states. Where the number of possible entities is small, or where the effective influence of the number of parents can be capped (such as “for 5 or more parents with state = X”), the conditional probabilities can be explicitly defined in the LPD. Otherwise, the LPD must have a functional form that defines the conditional probabilities.

A significant problem in first-order expressive probabilistic models is the exponential growth in the number of probabilities required as the number of parents increases. This creates problems for probability development whether via expert elicitation or ma-

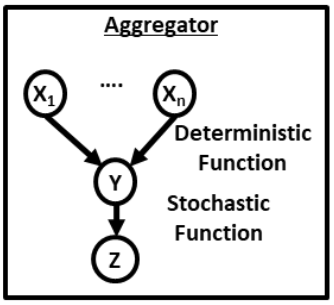
chine learning. If the LPD involves a complex degree of interactions among the parents, the problem is unavoidable. But for many problems, the LPD can be simplified by exploiting one or more independence conditions. These are context specific independence (CSI), independence of causal influences (ICI), and aggregators. Taking full advantage of these independence conditions requires specific LPD language capabilities.

CSI occurs in an LPD when only a portion of the parent configuration is needed to establish the child RV's probabilities. For example, the model in Figure 37A exhibits CSI. Although it has five parents, the state of *MachineLocation* limits the influence of the *RoomTemp* parents to only one *RoomTemp*, the one with the same room as its OV as *MachineLocation* has for its SV. ICI requires that two or more parent RVs do not have synergistic interactions with each other in the child RV's LPD. That is, the influence of each parent on the child RV's state probabilities does not depend on the state of any other parent. ICI models can be expressed using combining rules. Figure 38A shows the structure of an ICI model using a combining rule approach. The parents are the X_is. Each X_i stochastically influences the state of a Y_i. As there are no arcs from any X_i to Y_j, $i \neq j$, each Y_i is conditionally independent of all parents except X_i [20]. The states of the Y_is are then combined using some type of deterministic functional equation. The first combining rule model (noisy-or) used the logical OR as the deterministic equation (Pearl 1988). Other commonly used combining rules are the noisy-and, noisy-max, and noisy-min. ICI models can be built using any Boolean function – see Lucas (2005) or

[20] The X_is may have direct dependencies (e.g. X_i may directly influence X_j, $i \neq j$). These do not affect the child RV LPD.



A. ICI Based Combining Rule. Each parent has probabilistic effect on a intermediate variable. Results combined via function to get child RV state probabilities



B. Aggregator model. Parents are deterministically combined by some function, then result has a probabilistic effect on the child RV

Figure 38: ICI-based combining rule and aggregator models

van Gerven, Lucas, and van der Weide (2008). Some Boolean ICI models can be extended to multiple state models by exchanging the Boolean deterministic function for its algebraic extension (e.g. using Max instead of Or). There also has been some work on allowing a limited degree of LPD-level interaction between parents. This is especially the case in the medical domain, where drugs may have positive or negative synergy when used together (Woudenberg, van der Gaag, and Rademaker 2015).

In an aggregator, the states of the parent configurations are first combined via one or more deterministic functions, and the results used to create a deterministic RV. This RV then interacts stochastically with the child RV (see Figure 38B). A very simple example is to estimate a student's likelihood of success in a graduate program based on her performance in her undergraduate courses. An aggregator would average the grades in all courses (the X_is) to obtain a single value (the grade point average, in the Y RV). Then, there would be a probabilistic correlation between GPA and success in a graduate pro-

gram. Both ICI models and aggregator models depend heavily on the use of deterministic functions. Table 10 identifies commonly used functions.

Where the deterministic function produces a fixed or limited number of states, the stochastic function can be a CPT. If the aggregator produces a variable number of states, a functional probability model is often used. Here, the aggregation output (variable Y) provides the parameters of the function. This model is either a standard probability distribution (e.g. binomial, logistic, Poisson, etc.) or a custom-built probability model, usually based on some form of machine learning or expert elicitation. There are several examples in Kazemi et al. (2017). Kazemi et al. (2014) discuss the sensitivity of various probability models to different aggregator functions as the size of the underlying aggregated population changes. A good introduction to both ICI and aggregator models (called simple canonical models) is by Dıez and Druzdzel (2006).

Table 10: Commonly used combining rule and aggregator functions

Function	Form
Max / Or*	$y = Max(x_1,\ldots,x_n)$
Min / And*	$y = Min(x_1,\ldots,x_n)$
Sum	$y = (x_1 +\ldots+x_n)$
Proportion	$y_j = \frac{(x1 +\cdots.+xn)\mid condition\ \ j)}{(x1 +\cdots.+xn)}$**
Average	$y = (x_1 +\ldots+x_n)/n$
Linear combination	$y = (a_1x_1 +\ldots+ a_nx_n)$
Count	
- Histogram***	$y_j = Count(x_1,\ldots,x_n \mid Condition_j)$**
- K-of-N ****	$y_j = Count(x_1,\ldots,x_n \mid Condition)$**
* Or / And if x_i are Boolean	
**Count if condition j met. One condition for K-of-N	
*** Can determine median and mode from histogram	
****K-of-N support functions Exact (k=r), Threshold (k≥ r) and Ceiling (k ≤ r)	

Under conditions of nonuniqueness, where all $P(Y_i|X_i)$ are identical, it is possible to implement a combining rule as an aggregator. It can be done even when the X_is are uncertain. Because of this feature, it has become common to group both aggregators and combining rules as aggregators, especially in the PRM community (e.g. Raedt et al. (2016)). However, combining rules can also be applied to models where non-uniqueness holds, i.e. the $P(Y_i|X_i)$ are not identical, while aggregators cannot. ICI models with different probabilities are widely used, especially in the medical domain. See Bermejo et al. (2013), Magrini, Luciani, and Stefanini (2016), or Anand and Downs (2008) for examples and recent extensions.

A.4 Constraints

Often the problem environment has constraints among the attribute and/or relationship instances that must be captured to accurately model the problem. They fall into two categories: restrictions on the state of one or more parents, based on the state of other parent nodes; and restrictions on state of a child RV based on the state of one or more other child RVs. Constraints between parents and a child RV are enforced in the child RV's LPD. The relationship modeled between two OVs in a RV are not inherently constrained by the form of the RV. But in the intended understanding of the relationship, there may be a constraint on the number of entities instantiating a particular RV that may be true in a parent configuration. BN or DG modeling needs to include elements that enforce the appropriate constraint. These include:

- Many-to-Many: This is the unconstrained case. In the RV *isFriendsWith(***x,y***)*, an instance of **x** may have multiple friends **y** and vice

versa. The normal understanding of "Friend" does not imply any constraints on the number of friends one may have.

- N-to-Many or Many-to-N: With this constraint, only N entity possibilities in the first or second position may be true. N can be changed to "at-least N" or "at-most N". For example, for the RV *hasChild(***p,c***)*, a human child **c** has two biological parents **p**, but a parent may have any number of children (including none).
- One-to-One: any relationship with a one-to-one functional relationship between the entity instances of the two OV/state variables.

The following simplified problem is used throughout this section. It models predicting whether a woman has a child, as influenced by whether the woman is married to the father of a particular child. Figure 39 shows the template, while the model's database states that there exists a woman (W), a child (C) and three possible fathers (F1, F2, F3). It has the simplifying constraint that the woman must be married to one husband, who comes from the set of possible fathers. For the data given above, the construction algorithm creates the grounded BN shown in Figure 40A. Because this model has not yet addressed the constraint, it will not execute correctly.

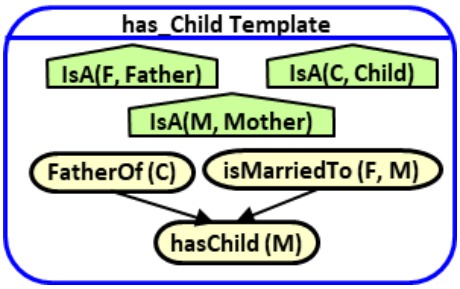
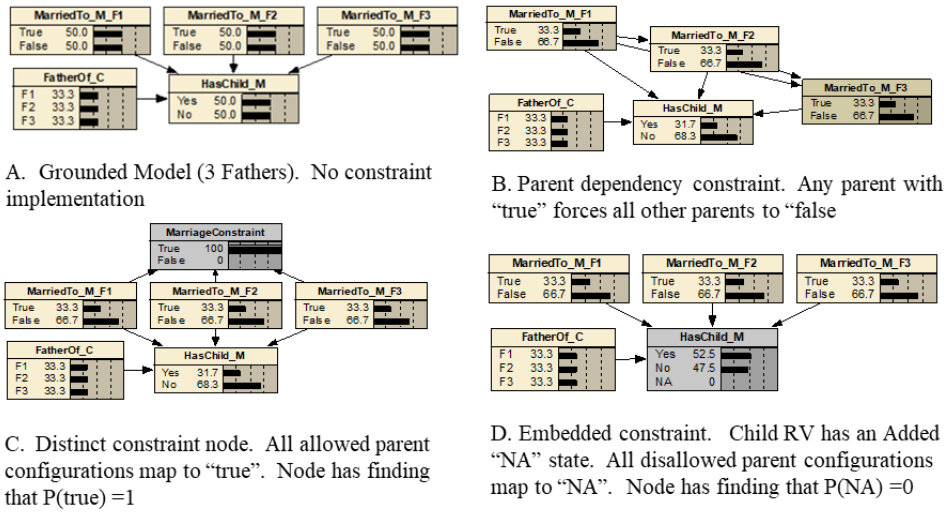


Figure 39: Constraint example Template, where probability of a woman having children depends on whether she is married to the father of possible child. Constraint is that she is married to exactly one possible father



A. Grounded Model (3 Fathers). No constraint implementation

B. Parent dependency constraint. Any parent with "true" forces all other parents to "false

C. Distinct constraint node. All allowed parent configurations map to "true". Node has finding that P(true) =1

D. Embedded constraint. Child RV has an Added "NA" state. All disallowed parent configurations map to "NA". Node has finding that P(NA) =0

Figure 40: Grounded model, with three ways to implement marriage constraint

There are three ways to model constraints: direct parent dependency, constraint node, or embedded constraint. The first approach is to add dependencies among the parent RVs that implement the constraint. Figure 40B shows a grounded model with parent dependencies. To do this requires a total order be on the entities in the classes affected by the constraints. The order may be natural (e.g. greater than, less than, etc.) or arbitrary. While a natural ordering is limited to one class, an arbitrary ordering can extend across multiple classes. In this approach, dependencies among the parents are used to enforce constraints. The core LPD is that any instance of the `isMarriedTo` RV type will have state `False` if any of its parents have state `True`. Assuming a uniform prior probability across all the instances of that RV type is appropriate, the probability that each entity in the order is married is

P(MarriedTo) =1 /(cardinality(entity set) – Order number +1)

Equation 3: Direct constraint approach's probability assignment to each ordered entity

While this example is simple, more complex constraints require a more complex LPD, which is harder to understand. Constraint nodes provide an easier approach. These are nodes added to the BN specifically to implement the constraint(s). Constraint nodes can be used to enforce constraints among any set of nodes. Figure 40C shows the basic pattern for a global constraint node, as implemented in the example model. There is a Boolean node, called the constraint node, with arcs from the nodes on which the constraint(s) is to be enforced. In its LPD, the conditions for which a parent configuration meets the constraint maps to True (e.g. in the example, exactly one `MarriedTo` RV has state = True). Otherwise, it maps to False. The constraint node has an database finding that the state of the constraint node must be True. This enforces the constraint. Constraint nodes and direct parent dependency are functionally equivalent, and if the constraint node is absorbed into its parents, then one converts Figure 40C into Figure 40B. It is generally easier to specify the LPD in terms of allowable and unallowable parent configurations, making the constraint node easier to implement. This is especially the case when a constraint applies across multiple classes or multiple types of RVs. A constraint node may be total or partial. The first is illustrated in the figure above, in which a single node applies the constraint to all nodes of one or more RV types as a group[21]. A partial constraint node applies to a subset of the nodes, and there may be multiple such nodes to enforce the overall constraint. Constraints can also be applied to a set of child RVs, using either approach. For instance, if each entity that instantiates a child RV must have a unique state, then a global or set of partial constraint nodes can be used to implement the constraint.

[21] It can enforce as many constraints as applicable to the same set of parents

When a constraint node is used, it adds additional nodes to the BN and increases the complexity of the network. For constraints among parent nodes (including subsets of parent nodes), it is possible to embed the constraints into the LPD of the child RV. The set of states of the child RV is augmented with a new state whose purpose is to implement one or more constraints among the parent RVs. The patterns in Appendix 2 use *NA* (for "Not Allowed"). For each child RV instance that has an *NA*, there is a finding that the *NA* state must be false; e.g. it is not allowed. Figure 40D above shows the use of an embedded constraint. The LPD is constructed in two parts. First, all the parent configurations that have constraint violations are mapped to *NA* (same approach as for a constraint node). Then the remaining parent configurations are mapped to their child states with the conditional probability appropriate to that PC.

The above approaches have a problem if one is implementing a 1-of-n constraint, as in the example. If prior information assigns different probabilities to the nodes being constrained, then the constraint approaches described above will change the value of those nodes. Fenton et al. (2016) discovered a relatively straightforward way to correctly initiate the prior probabilities. They also identify means to allow one to model "0 or 1 of cases", such as allowing a person not to be married in the example used above.

APPENDIX B
FIRST-ORDER EXPRESSIVE BAYESIAN NETWORK MODELING PATTERNS

This appendix details the first-order expressive Bayesian network modeling patterns this research identified. Then, it goes into depth on the patterns, which are divided into three categories. The patterns were uncovered during the dependency modeling analysis effort described in Appendix A. A key feature of these patterns is that the structural arrangements in the template shapes the structure of the instantiated grounded model. This, in turn, influences the structure of the LPD. This is also the basis for the LPD script language needs identified in Chapter 4. While this research focused on Bayesian networks, it can be readily applied to probabilistic logic programs, as any BN can be implemented using an equivalently expressive probabilistic logic language (Poole 2008). The LPD examples described below represent the actions to be performed, not the specific implementation described in Chapter 4.

B.1 Modeling Patterns in the Literature.

Modeling patterns discussions in the literature can be grouped into three categories: domain-specific patterns, common domain-independent element patterns, and qualitative probabilistic network patterns. Various patterns to guide domain-specific models have been developed. Examples include situational awareness models (Park et al. 2014), legal argumentation (Vlek et al. 2014), project risk management (Al-Rousan, Sulaiman, and Salam 2009), maintenance system management (Medina-Oliva, Weber,

and Iung 2013), return on investment analysis (Yet et al. 2016) and disease process modeling (Shpitser 2010). These patterns focus on identifying relevant entity classes, key attributes and relationships, and the most common or most important dependencies between the attributes or relationships for a specific domain.

Several authors have identified specific patterns that can be used across domains. Although not focused on BNs specifically, Schum identified a number of patterns for probabilistic reasoning about conclusions drawn from data using probabilistic graphical models (Schum 1994). Neil, Fenton, and Nielson identified five basic patterns, that they called idioms:

- Definitional / syntheses: nodes that synthesize results from several parents, such as defining a system's safety in terms of the frequency of safety events and their severity. Parent divorcing (Olesen et al. 1989) is included in this pattern
- Cause-consequence: modeling uncertainty in causal processes
- Measurement: modeling errors in a measurement process
- Induction: modeling inductive reasoning
- Reconciliation: modeling uncertainty when combining results from different measurements (Neil, Fenton, and Nielson 2000).

They later applied their idiom concept to the legal domain, providing six additional patterns widely applicable to any domain where one is reasoning about possible conclusions drawn from data (e.g. intelligence analysis, market analysis, investigative work, etc.) These patterns predominately focused on their structural aspects (Fenton, Neil, and Lagnado 2013).

Helsper and Van Der Gaag took Neil et al's measurement pattern and developed a more comprehensive pattern for reporting and evaluating testing results (Helsper and van der Gaag 2005). Almond et al. provided a set of patterns for use in incorporating common effects patterns in measuring outcomes that result from a common stimulus (Almond et al. 2009). These patterns also focused on their structural aspects.

The probabilistic relational modeling and Bayesian logic communities identified basic patterns relating to various uncertainties that can exist in a first-order model. These include attribute uncertainty (e.g., missing data) (Pfeffer 1999), reference and existence uncertainty (Getoor et al. 2007), type uncertainty (Koller, Levy, and Pfeffer 1997), number uncertainty (Milch et al. 2007), and identity uncertainty (Pasula 2003). In addition to the above, patterns identified in widely used teaching texts include constraint modeling using constraint variables, dealing with bidirectional relations, and simplifying modeling using temporal transformation or naïve Bayes (F. V. Jensen and Nielsen 2007; Kjaerulff and Madsen 2013). These patterns included both structural and LPD aspects.

Finally, there are qualitative probabilistic networks (QPNs). Wellman proposed QPNs as a means to qualitatively understand the behavior of a BN (Wellman 1990). A key use has been to assist model development by identifying deficiencies and providing an efficient network model (Renooij and van der Gaag 2002). Lucas used QPNs to initiate a study of modeling patterns using Boolean combining rules under the independence of causal interactions assumption (Lucas 2005). The study was then expanded to assess all 16 binary Boolean patterns for applicability to BN modeling (van Gerven, Lucas, and van der Weide 2008).

B.2 Patterns

As discussed in chapter 4 and appendix A, 37 specific cases were created which instantiated a variety of structural arrangements. For each, the actions the LPD needed to take to assign a probability to each of the states were identified. This appendix focuses on both the structures resulting from the RV-Type / OV combination, and the interactions within the LPD. Recurring behavior patterns were noted during the exploration. They grouped themselves into three major categories, called selector, existential selector, and impacted dependencies. For each pattern, there is a figure showing one or two examples. In general, the figure includes

- Dependency model case with a generic variable combination (x, y, z)
- Template with the scenario model. Scenario is described in the text
- Question being address in the scenario, as a query
- Knowledge base with number of entities, and grounded model
- Constraints enforced in the grounded model. All constraints are enforced using an embedded constraint node approach
- Basic dependency between parent(s) and child RVs
- Required LPD behavior

B.2.1 Selector Patterns

Selector patterns are two-parent-type models where the template has either an F-type or A-type RV with the same OV as the child RV. Because it has the same OV as the child, only one instance will be instantiated. This instantiated RV acts as the selector. It identifies a condition that the instantiated RVs of the other variable type must meet for

the instantiated variables to have an influence on the child RV. There are three selector patterns: Select-One, Select-Match and Child-Select.

In the ***Select-One pattern***, the selector is an F-Type RV and the state variable of the selector is an OV for the other parent RV-type. In this pattern the LPD always exhibits context specific independence**.**

Figure 41 shows two examples of this pattern. Example #1 is a well-known pattern used to resolve reference uncertainty (Getoor and Grant 2006; Getoor et al. 2007). The example begins with the basic pattern, in which an attribute RV A1 with entities from the class represented by OV **`y`** has an influence on a specific entity with attribute RV A3 (whose OV may be from the same or different class than the OV in A1). The functional relationship in RV F2 enables the dependency between A1 and A3. The template in example 1 captures the fact that a machine is in a particular room (as defined in *`MachineLocation(`***`M`***`)`*), and the room has a temperature of either normal or hot. The room's temperature affects the probability of the machine's status, either normal or overheated, per the LPD table. Here, one is uncertain about which room the machine is in but does know that it must be in one of four rooms. The construction algorithm takes the template and the database entry that there are four possible rooms to create the grounded model. In the LPD, *`MachineLocation_M1`* acts as a selector. For each parent configuration, it identifies the one room whose state will establish the probabilities for that PC, using the probabilities expressed in the table. The figure includes a representative set of LPD actions visualizing the CSI characteristics.

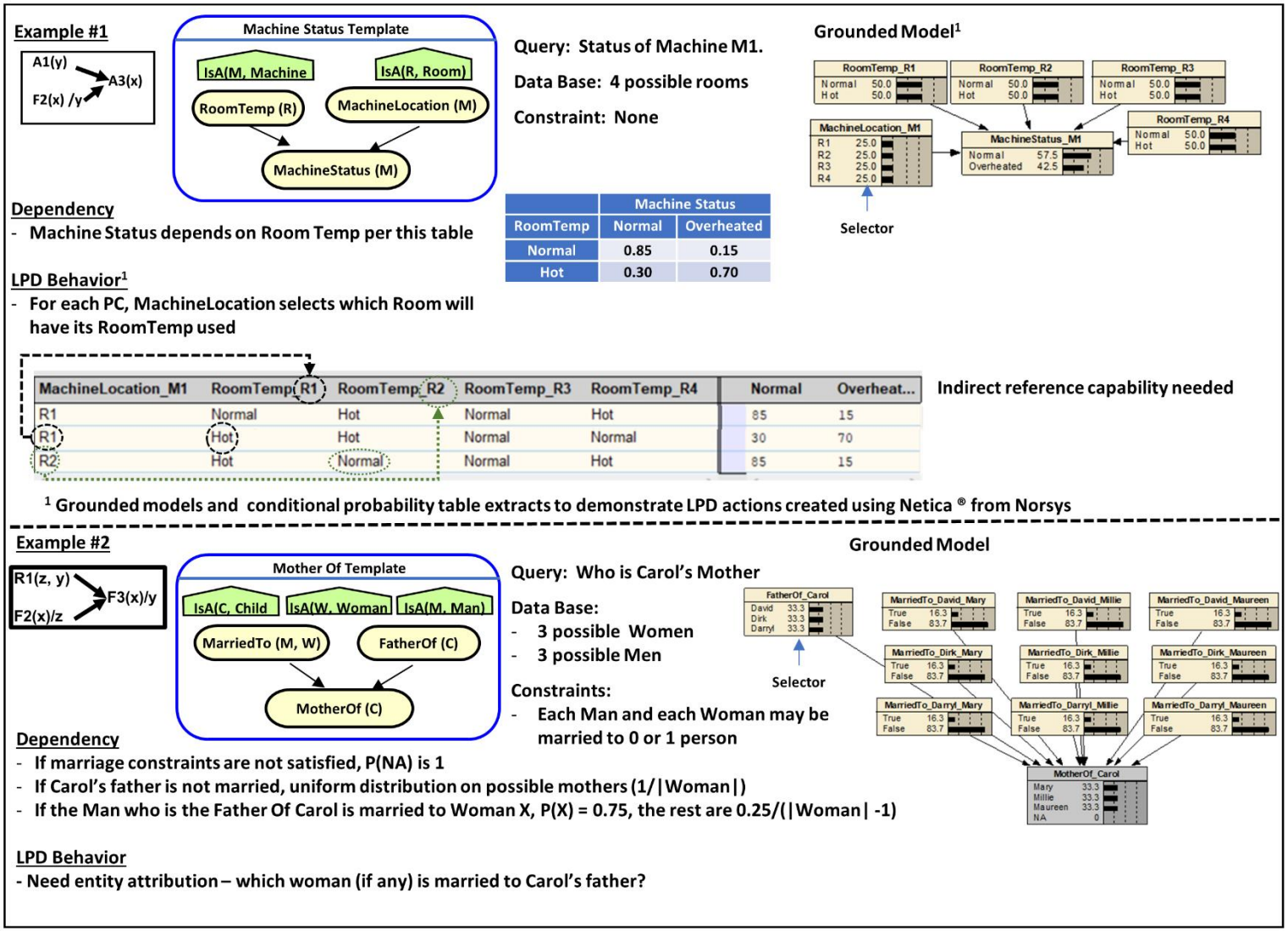


Figure 41: Select-One pattern examples

A key LPD behavior is to perform indirect referencing. Here, the state of the F-type selector is the room of interest. Indirect referencing allows us to say

State(RoomTemp(State(MachineLocation)))

to indicate we want to know the room temperature of the room that is the state of the *MachineLocation* RV.

While example #1 is an attribute-to-attribute example, where the selector licenses the dependency, one can create similar patterns with dependencies that include relationships. This is useful when doing probabilistic social network analysis. In example #2, there is a dependency model where a R-type RV influences an F-type child RV, with a F-type RV enabling the dependency. The specific example models the probability that a particular woman is the mother of a child, depending on whether she is married to the father of the child. The template shows three entity classes. The knowledge base identifies three men as possible fathers and three women who could each be married to one of the men. The lists of possible mothers and fathers is collectively exhaustive. A key constraint is that each man and each woman can be married to at most one person. In the grounded model, the *MotherOf* RV (the query node) includes an *NA* state. This model uses an embedded constraint node approach. *FatherOf* has the same OV as *MotherOf*, so only one instance is created. *FatherOf* is the selector. The variables in *MarriedTo* are free, so it will instantiate |z| * |y|[22] instance RVs.

[22] |x| is cardinality of the class from which x is drawn

This example has two interesting features. First, the marriage constraint applies individually to six different people, resulting in 6 constraints, one for each person in the knowledge base. The constraints are partial constraints across the parent set. Figure 42 visualizes the constraints needed. Each row in the parent set corresponds a specific man, and each column to a woman. In the LPD, constraints are checked first, and the constraint is applied for each man and woman (row and column respectively). If there are more than one marriage in any row or column, that parent configuration has 100% probability of *NA*. Constraints can eliminate a significant number of parent configurations from consideration. Here, there are 1536 parent configurations; constraint violations eliminate 1434.

The second interesting feature is that the LPD requires parent entity attribution, knowledge of the specific parent entities that match a specified condition to properly assign the child RV's state probabilities. For all PCs that meet the constraints, the state of the selector RV, *FatherOf*, is determined, and the *True* states of the three *MarriedTo* RVs with that man are counted. In this case, the count will be either zero or one, as the constraint has already *NA* the PCs that have more than one *MarriedTo* being *True*. If the count is zero, then each of the possible mothers will have a uniform

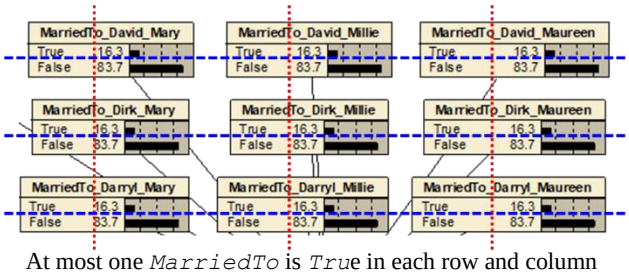


Figure 42: Multiple constraints

probability, 1/|Woman|[23]. If the count is 1, then a 75% probability is assigned to the woman who is married to the father, while the remaining women each receive a uniform share of the remaining probability, e.g. 0.25 / (|Woman| -1). In this case, the LPD needs to know which specific woman is the married one, because the child RV states also come from the Woman class. Entity attribution is required whenever the child RV is an F-type RV, as its states are entities.

In the ***Select Match pattern***, the selector identifies a state of interest and the LPD examines the other parent type to identify which ones match the selector state. In this case, the selector may be either an F-type or an A-type RV. This pattern requires entity attribution. The selector RV has the same OV as the child RV, so only one instance of the selector variable is instantiated. Figure 43 highlights two examples of this pattern. In the first example, two F-type RVs collectively influence an F-type child RV, and both parent types have the same state variable (**o**). In the example, which room a particular machine is in (the *MachineLocation* RV) depends on matching information from both the room choices and the selected machine. There are three classes involved: machine, room, and organization, with one machine entity, four room entities and two organization entities. For this model, there is a requirement that a machine can only be in a room whose tenant is the same organization that owns the machine. This becomes the selector condition the entities of the other RV type must match. *Owner_m1* is the selector, and the

[23] |Woman| is the cardinality of the class Woman

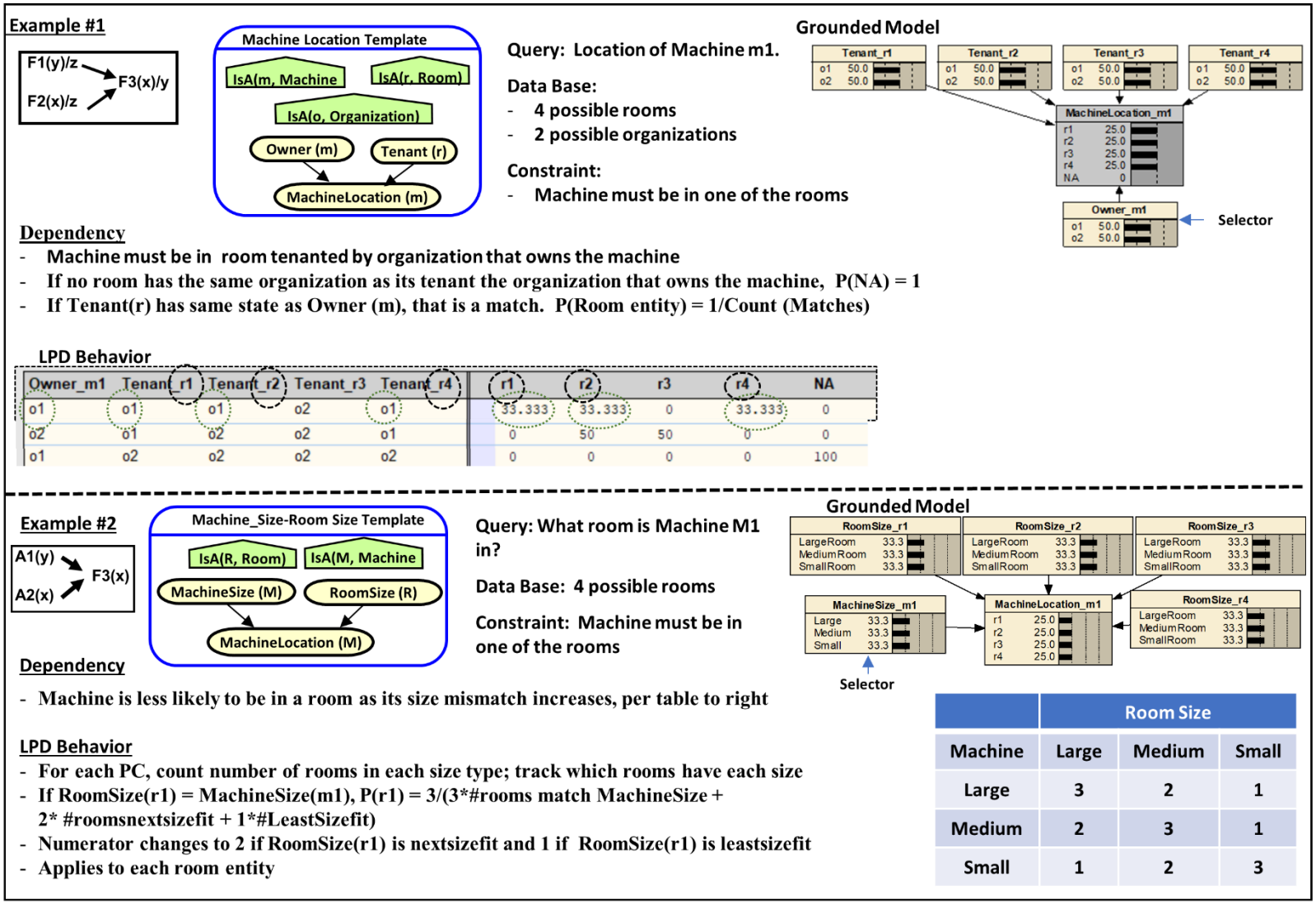


Figure 43: Select Match pattern examples

states of the child RV are from the same class (room) as the OV for the RV *Tenant*. We are looking for the behavior

COUNT((State(*RV2*)= *State*(*RV1*))

In the LPD excerpt, one sees three parent configurations, each with the state of *Owner_m1* being *o1*. In the first line, the tenant of rooms *r1, r2* and *r4* is also organization *o1*. The child RV probability is then 1/Count for each of the rooms that met this condition (*r1,r2,r4*). The second LPD behavior line shows *Owner_m1* state being *o2,* with only tenant *r2* and *r3* having the same state. The grounded model shows that *MachineLocation_m1* has an *NA* with probability 0, indicating the existence of an embedded constraint in the LPD that the machine must be in one of the four rooms. However, there are two parent configurations where this is not possible. In the first, machine *m1* is owned by organization *o1* while the four rooms are tenanted by *o2*, while the second parent configuration is reversed. In both cases, the probability of the machine being in any room is zero. The embedded constraint indicates that these two parent configurations have no meaning in this model. Line 3 of the LPD excerpt shows one of the two disallowed PCs.

In the select match pattern, it is possible for an attribute to be a selector. This depends on exploiting an embedded dependency between two attributes associated with different entities. Example #2 uses the A1-A2-F3 dependency model, with each attribute having a different OV, and the child RV having one parent OV as its OV and the other parent OV as its state variables. Let us assume there are three different machine sizes (say

large, medium and small) and three different room sizes (large room, medium room and small room). The machines can be in any type of room, but a machine is three as likely to be in a room that matches its size than in the most different room. It is twice as likely to be in a room that is nearest to its size. In the example, there are four possible rooms. In the LPD, the probability of being in a specific room depends on the total number of rooms, and on how many rooms have the same size type as the machine, versus a different size type. The probability distribution is determined via a functional probability specification.

$$\text{P(room y)} = \frac{Match\ Indicator}{3*\#\ Rooms\ Match+2*\#Rooms\ Next\ size+1*LeastSizet\ Match}$$

Match Indicator = 3 if size of Room y matches machine size, 2 if size is next best size, otherwise 1.

Equation 4: Probability of specific room depends on its size match and number of rooms with each size

Note that this LPD requires that every possible size combination be counted before the probability assignment is made. This is called state counting capability, e.g., how many rooms of each size are there. In addition, it also requires entity attribution.

The third selector-based pattern is the ***Child Select pattern***. It applies to certain dependencies that have a R-type RV as the child. In this pattern, one OV in the R-type child RV is the OV of the selector, while the other OV is used to determine whether the parent configuration supports a state of `True`. Consider example #1 in Figure 44. Whether a belt is owned by a particular organization depends on which machine it is on

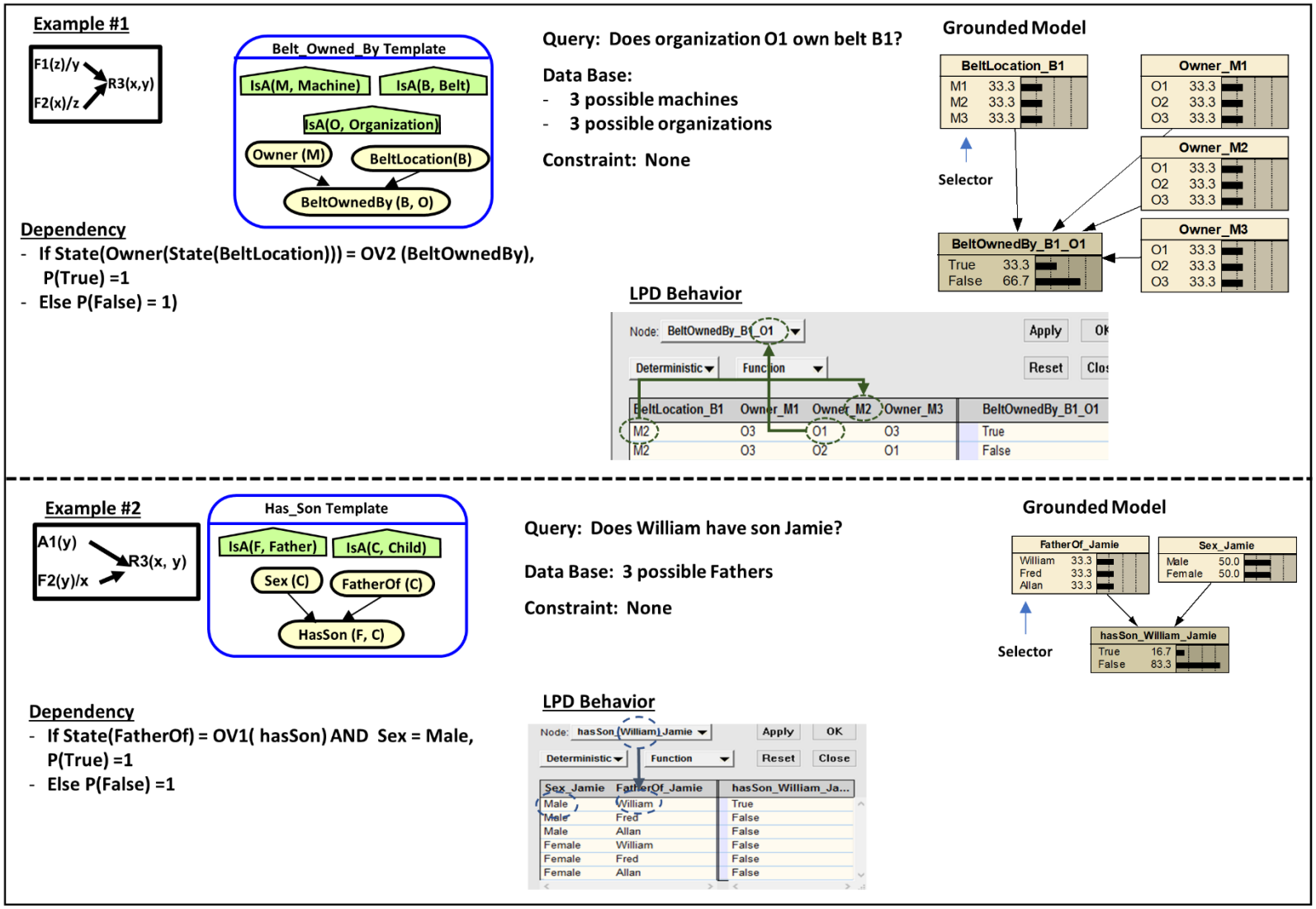


Figure 44: Child Select pattern example

and who owns that machine. The premise is that the owner of a machine owns all the parts on it. The grounded model shows three machines and three possible owning organizations. *BeltLocation* is the selector. The LPD acts very much as in the Select-One pattern, in that the selector identifies one instance of the other parent RV type as having influence.

However, rather than just using the state of that RV instance in the LPD, the LPD compares the state of the parent RV *Owner* to the second OV of the child RV. If they match, the child RV state is *True*. Otherwise, it is *False*. In the figure, the LPD language uses two forms of indirect referencing. The first, *State(Owner(State (BeltLocation*))), says that the OV entity of the owner RV is the state of the *BeltLocation*, where *BeltLocation* states are *M1*, *M2* and *M3*. The second, OV2(*BeltOwnedBy*), identifies the entity instance in the second OV position (counted from the left) of the RV inside the parentheses. The full LPD can be read as

> "If the state of *Owner*, whose entity is the state of *BeltLocation*, equals the entity of the second OV position of *BeltOwnedBy*, then the probability of *True* is 1

An indirect referencing capability is needed when dealing with entity states from a possibly vary list.

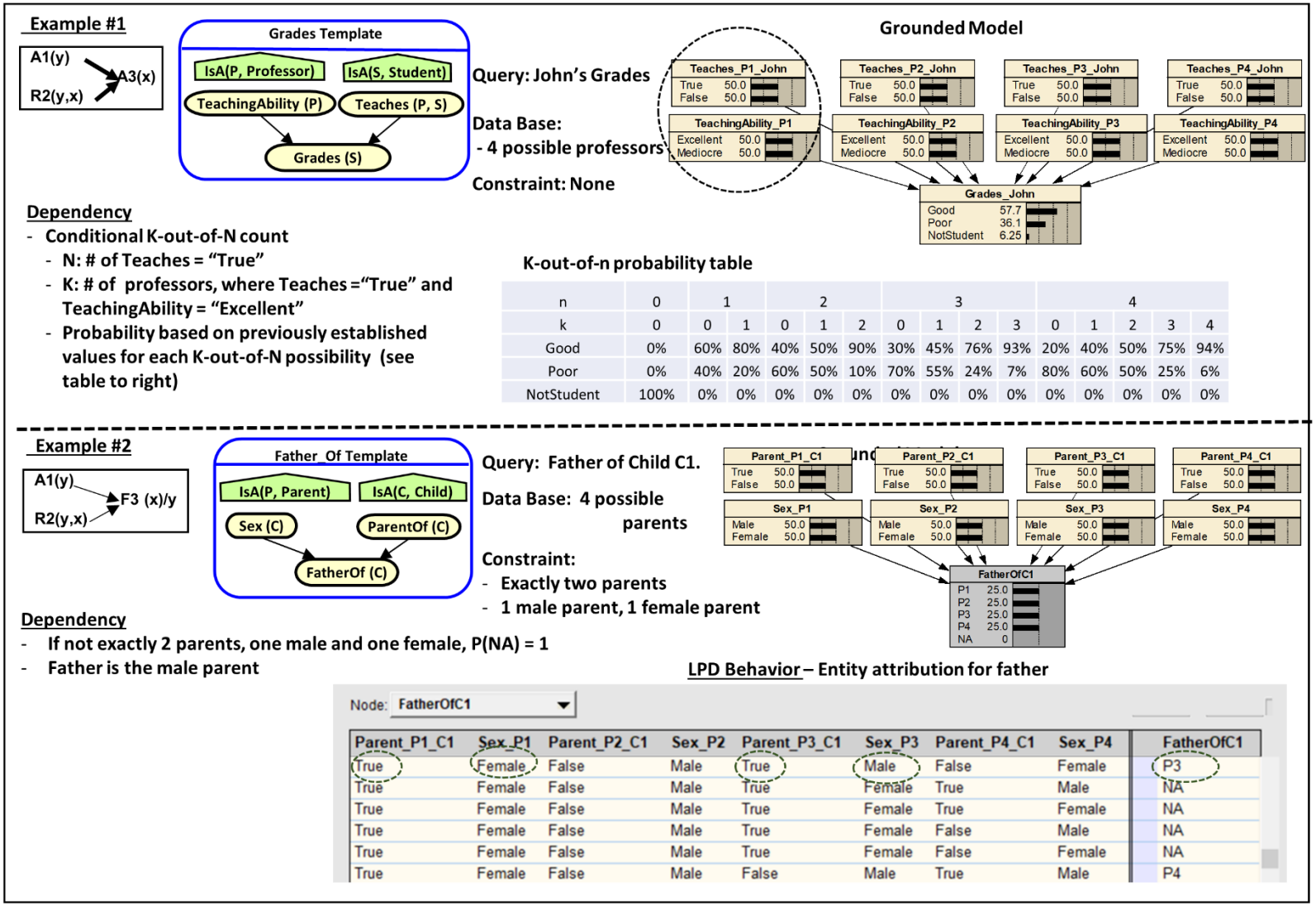
Example #2 shows a more direct variant. In this case, the selector itself must match the child RV's OV. The model addresses whether a person (William) has a son (Jamie). The available information is that there are three people who are potential fathers of a child, and the possible sex of the child. There are two conditions that must be met. First, the selector *FatherOf* must itself match a criterion. For the child RV to be

True, the possible father must be the same as the instance of the Father OV in the child RV. Second, the sex of the child must be male. In this case, one OV in the child RV specifies the entity of interest (the Child class OV), while the other OV is a condition that must be met by the selector.

B.2.2 Existential Selector Patterns

The second pattern category, Existential Selector, involve either a R-type or F-type RV. If it is a R-type RV, one or both of its OVs will be the same as for the child OV. If the child RV has only one OV (A-type or F-Type), one of the R-type's OV will be free. If there is a F-Type RV, it is that RV's state variable, not OV, that will match the child RV's OV. These patterns differ from the selector patterns in that they instantiate multiple instances of the selector RV, instead of only one. Each existential selector determines whether a specific instance of the second RV type has an effect or not. Uncertainty of a relationship's existence is a form of property uncertainty, hence the existential name (Poole 2011). These patterns use a variety of counting behaviors. There are three existential selector patterns: Existential-Paired, Multi-Existential and Child Existential.

In the ***Existential Paired pattern***, a distinct instance of an existential selector is paired with an instance of the second RV type in the template. Example #1 of Figure 45 gives a paradigmatic example. The premise is that student's overall grades depend partially on his professors' teaching abilities. Assume there are four possible professors, and each professor has either *excellent* or *mediocre* teaching abilities. Only the ones that teach the student will influence his grades (*Teaches* is the selector). Hence, there is a pairing between each instance of the Relationship RV *Teaches(**p,s**)* and the



n	0	1		2			3				4				
k	0	0	1	0	1	2	0	1	2	3	0	1	2	3	4
Good	0%	60%	80%	40%	50%	90%	30%	45%	76%	93%	20%	40%	50%	75%	94%
Poor	0%	40%	20%	60%	50%	10%	70%	55%	24%	7%	80%	60%	50%	25%	6%
NotStudent	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%

Parent_P1_C1	Sex_P1	Parent_P2_C1	Sex_P2	Parent_P3_C1	Sex_P3	Parent_P4_C1	Sex_P4	FatherOfC1
True	Female	False	Male	True	Male	False	Female	P3
True	Female	False	Male	True	Female	True	Male	NA
True	Female	False	Male	True	Female	True	Female	NA
True	Female	False	Male	True	Female	False	Male	NA
True	Female	False	Male	True	Female	False	Female	NA
True	Female	False	Male	False	Male	True	Male	P4

Figure 45: Existential Paired pattern examples

attribute RV *TeachingAbility(***p***)* for that professor entity. A commonly used LPD is a k-out-of-n aggregator model, where n is the total number of the existential selector instances that are true. For those professors that teach the student, k is the number of teachers with excellent teaching ability. Note the *NotAStudent* state for cases where no professor teaches the purported student. A probability value is attached to each k-out-of-n state, as established by expert judgment, data learning, etc. While this example has binary states for both the attribute and child RVs, it can readily be extended to multiple states. A k-out-of-n count then becomes a state count.

The second example is a refinement, where the entity associated with *ParentOf* can be refined to *FatherOf*, if the entity is male. A key item to making this model work is to ensure two constraints are properly accounted for. First, there must be exactly two biological parents. Second, one parent must be male and one female.

One can also see that if the relationship existential selector variable has an enforced many-to-one constraint, then this pattern gives an equivalent result as the reference uncertainty multiplexor in the Select One pattern. For example, the *SonOf(***f,c***)* RV, where **f** is the biological father class and **c** is the child class, is constrained to having only one father per child instance. This constrained relationship could also be modeled as *FatherOf(***c***)*.

In the **Multi-Existential pattern**, each selector instance is applied against multiple instances of the second RV type. Rather than pairing between specific instances, it works across subsets (or all) of the instances. Example #1 in Figure 46 models the case that a woman has one or more children, based on whether the woman is married to a man

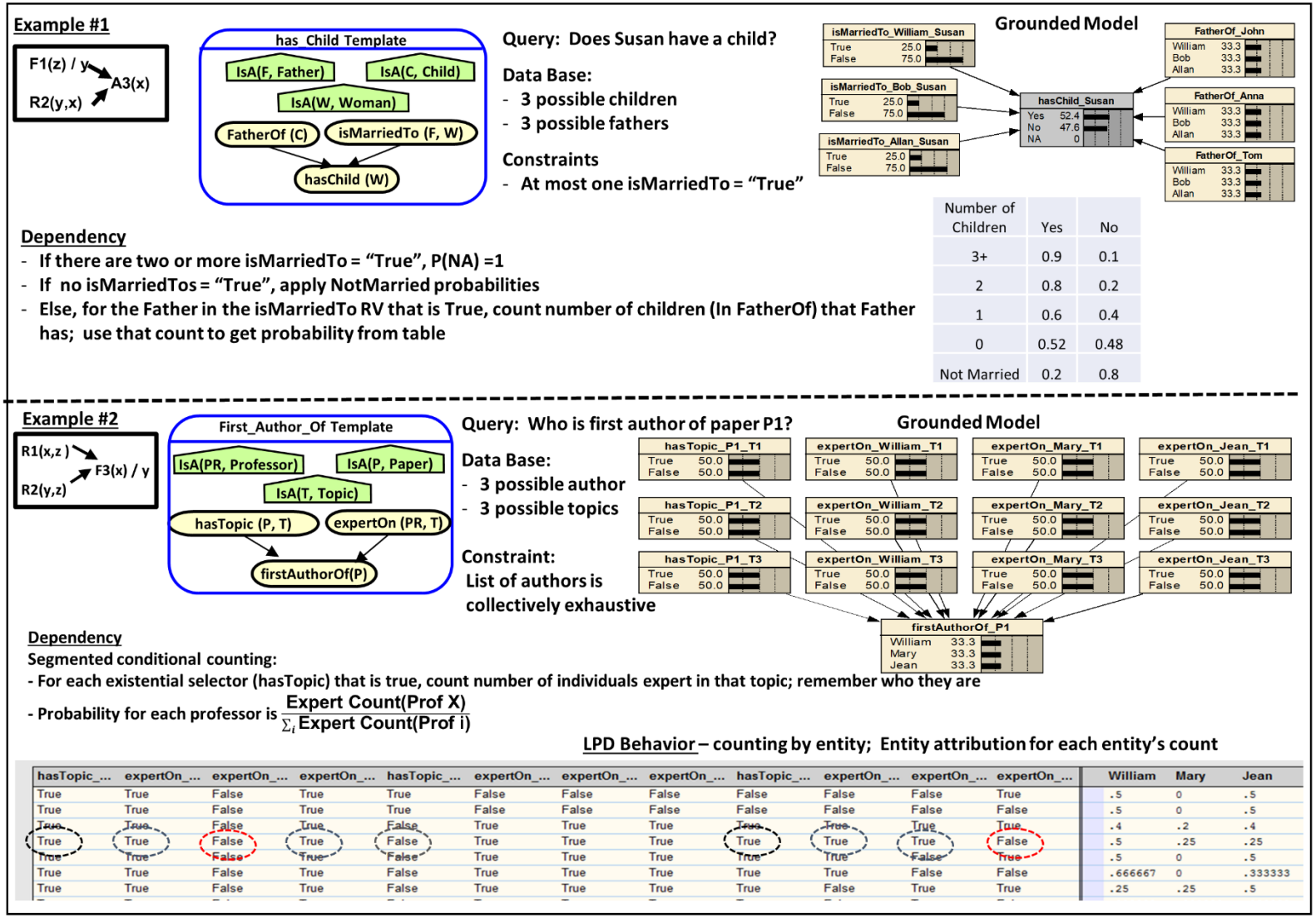


Figure 46: Multi-Existential pattern examples

and then counts how many children that man has. There is also a default probability value in case the woman is not married. There is a constraint that the woman of interest can be married to one man or not be married at all. This is modeled by an embedded constraint in *hasChild*, as indicated by the *NA* state. Note that in the grounded model, there is a 25% probability that the woman is not married. *isMarriedTo* acts as an existential selector, but there is no one-to-one pairing between *isMarriedTo* and *FatherOf*. Rather, for each instance where *isMarriedTo* is *True*, one counts how many *FatherOf* RVs have the same entity as its state as the husband in *isMarriedTo*. A probability value is based on that count. Example #2 shows the pattern when one relationship influences the state of another relationship. Here, one is assessing who the first author is on a paper, based on the paper's topics and the areas of topical expertise of various possible authors. This example includes a simplifying constraint that the list of possible first authors is complete. The existential selector is the *hasTopic* RV, which identifies the possible topics of a paper of interest. For each topic, there is an associated probability that a particular professor has expertise in that topic. The LPD here counts the number of professors that have expertise in a particular topic area. The higher the count, the less likely any particular professor is the first author. However, if the paper has multiple topics, then added weight is given to those professors who have expertise in more topics.

The ***Existential Child pattern*** is like the Child Select pattern discussed above, in that there is a dependency between the child RV's OV instance(s) and the selectors' states. Instead of a single selector, there are multiple existential selectors. Example #1 in

Figure 47 assesses whether two people are married based on the number of common children they have. In the example, there are three children and three possible mothers and fathers. There is an assumption that the list of mothers and fathers is complete. For each child, the LPD checks whether the states of *MotherOf* and *FatherOf* match the respective mother or father OV in *MarriedTo*. The count of common children is then applied to a probability table to get the probability for each possible PC from this model's parent set.

Example #2 looks like an Existential Paired pattern but has a context specific independence (CSI) twist. Here, the problem is to determine the status of a specific machine (M1), based on the status of a cooling fan belt on that machine. There is uncertainty about which belt is on which machine (*BeltLocation*). There is also uncertainty about the status of each belt. There is a constraint on this problem (The *NA* in *EngineStatus*). There must be at least one belt on the machine. In each PC, the LPD looks at the location and status for each belt. Because one is interested only in machine *M1*, the LPD checks whether the state of *BeltLocation* matches the OV in the child RV, *EngineStatus*. This gives the LPD a CSI twist. If the machine is not *M1*, it is ignored. If it is, then the belt status of the associated belt affects *Engine Status* states per the probabilities of the table.

Although this research focused on single and dual RV-type parents to identify the patterns, many real-world problems require more complex implementations of the patterns. Selection patterns are widely used, especially within probabilistic relational

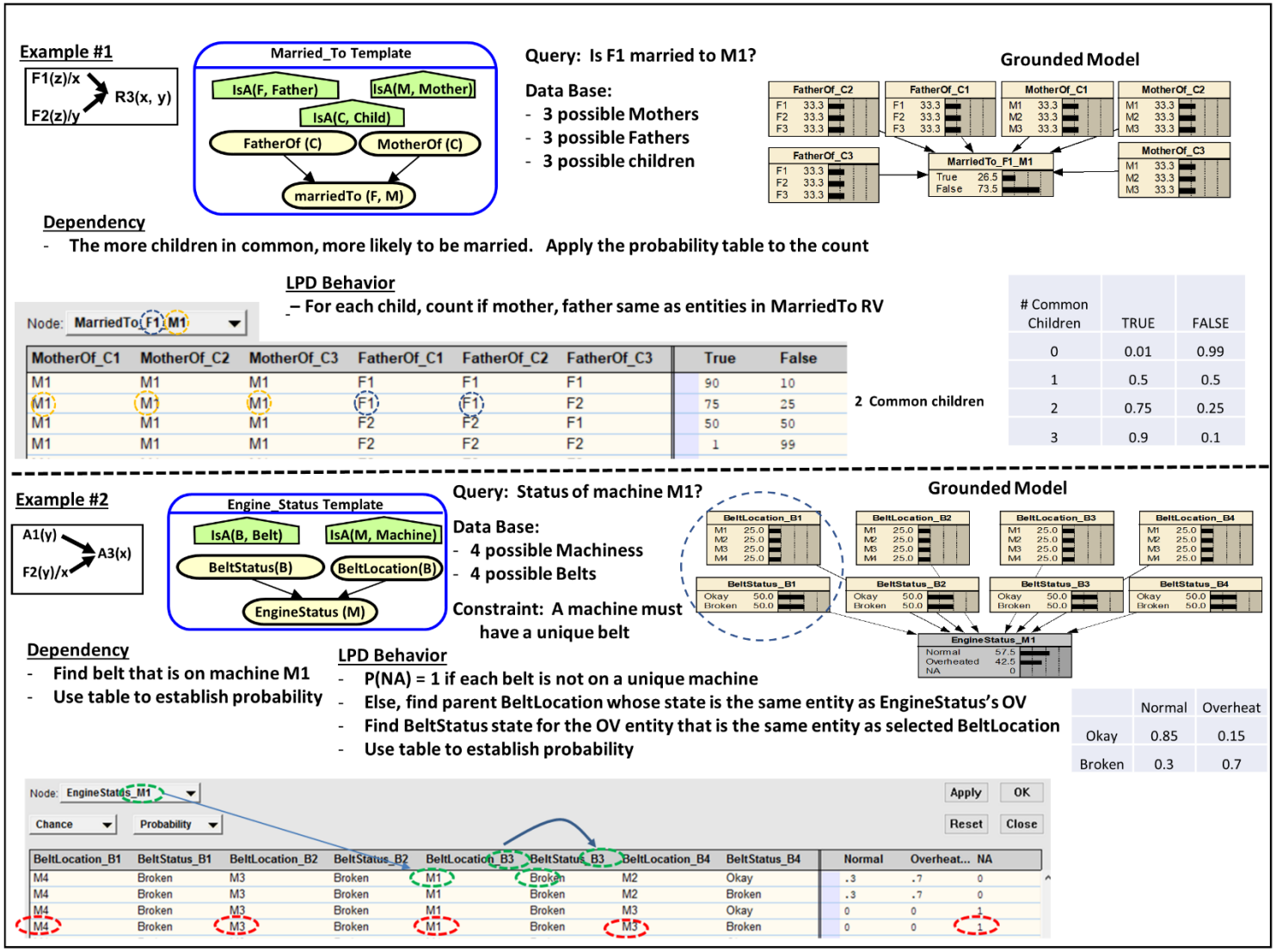


Figure 47: Existential Child pattern examples

models. One can extend these patterns to a multiple RV-Type problem. One example is a three RV-type model, where two parent RV-types are used as existential selectors. The original model was developed by Kazemi et al (Kazemi et al. 2017). They explored the performance of various aggregators on a common problem. The intent was to understand performance differences, rather than developing the most capable system for the problem. The problem is to determine the gender of a movie rater, given a data base of movie ratings given by raters whose gender are known. The template is in Figure 48. This example uses one aggregator they developed, which has a double selection process[24]. The data base includes entities for Movie and Rater, and has multiple relations and attributes. For this model, one relationship and one attribute are used: *Rates*(**r,m**), a Boolean relation that states whether a particular rater has rated a specific movie, and the gender attribute of the raters. In this model, the RV for *Rates*(**r,m**) is used twice – once to identify the set of movies the rater being queried about has rated, and another time to identify which of those movies the rest of the raters have rated. The context node *~(X=Y)* ensures that information about the query entity is not included in the LPD model.

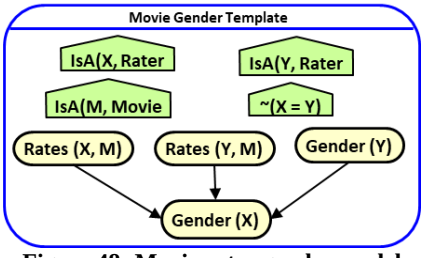


Figure 48: Movie rater gender model

[24] They explored six different aggregator models, which used different mixes of data in the knowledge base. I selected one for this demonstration.

The probability equation is

$$P(Gender(x) = Female \mid E)$$
$$= \frac{c + \sum_m [I(Rates(x,m)) * \sum_y (I(Rates(y,m) * I(Gender(y) = F))]}{2c + \sum_m [I(Rates(x,m)) * \sum_y I(Rates(y,m))]}$$

Equation 5: Movie rater probability equation

Where **m** is the set of movies and I is an indicator function with value 1 if the RV instance has state *True* or the specified state is true (e.g. *gender = female*), else it is 0. The constant c represents a pseudo count as a prior and is established via a learning algorithm prior to using the model.

The model is a conditional double count model. For each movie, if rater x has not rated it, it is ignored. This is the first selector (*Rates(***X, M**). If rater x has rated it, the second selector guides the counting of both the number of female raters (denominator), and total number of raters (numerator).

B.2.3 Embedded Dependency Patterns

The third pattern category are **Embedded Dependency patterns**. These all are single parent type patterns. There is not a second RV type that enables the linkage between the first RV type and the child RV. The research identified two different embedded dependency patterns: conversion / inversion patterns and existence patterns.

The **Conversion pattern** switches an F-type RV to an equivalent set of R-type RVs, and vice-versa. It is useful when the knowledge base has information in one form,

but when the other form is more appropriate for the model. An example of each is shown in Figure 49. The F-to-R conversion is a simplified model converting a functional representation of possible causes of death (assumed to be collectively exhaustive) to a series of Relationship RVs, one for each cause. The LPD is guided by the child's OV. If the state of the parent *CauseOfDeath* matches the Cause class OV in *DeathCausedBy*, then the child's state is *True*. Otherwise, it is *False*.

In the R-to-F conversion, one is taking a constrained (exactly one) group of Relationship RVs and converting them to a single functional F-type RV. The LPD is identical to an embedded constraint node. All PCs with more than one parent true or with no parent true map to the *NA* state. For remaining PCs, the state of the child is set to the Parent entity that has a state *True*.

The **Inversion pattern** provides the inverse of a relationship, which is not necessarily the same relationship with the variable order reversed. For example, *hasFriend*(**x,y**) is normally considered a symmetric relationship and is its own inverse. But *hasChild*(**x,y**) and *hasParent*(**y,x**) are each an antisymmetric relationship, but are considered inverses of each other. If the inverse of a relationship is a different relationship, the modeler needs to understand the nature of the inversion. The naming of the RVs and determining whether the two RVs have the same semantic meaning is the modeler's responsibility. The pattern only ensures that the uncertainties among the states is preserved between the RVs. Trivially, one can always create an inverse by switching the voice (active / passive) of the verb phrase that defines the relationship. For example,

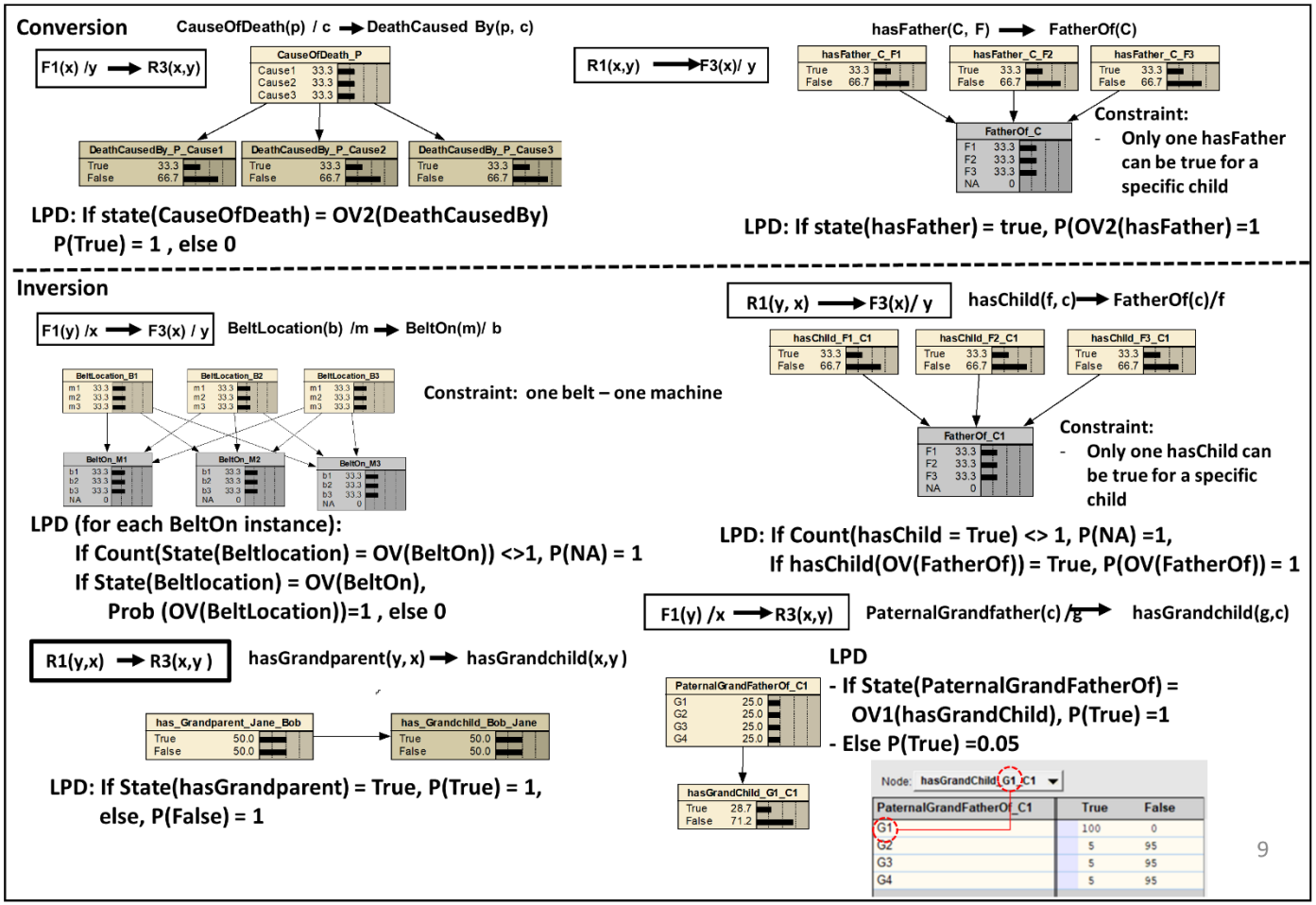


Figure 49: Conversion / Inversion patterns

hasFather(**x,y**) and *isFatherOf*(**y,x**) are formally inverses of each other. But often one is interested in an inverse that describes the relationship differently, such as *hasChild*(**x,y**) and *hasParent*(**y,x**). In those cases, the dependency between the two may not be as tightly bound.

Because relationships can be expressed either as R-type (always) and F-type (for a functional relationship), there are four possible implementations, as shown in Figure 49. In the F-F case (*BeltLocation*(**b**) / **m** to *BeltOn* (**m**) / **b**), there must be a one-to-one relationship between the entities in the OV and state variable of the two RVs. The *NA* constraint in the child RV enforces this requirement. Note the effect of having this constraint on the grounded model. While the query would be on one entity, the query-specific construction algorithm will instantiate a child RV for each entity in the OV class. This occurs because the finding on each *BeltOn* RV instance that *NA* must have zero probability means that each *BeltOn* instance becomes d-connected to every other *BeltOn*. The LPD is straightforward – the specific belt on a machine is the parent with the same state as the OV of that machine. The LPD includes a count command, which counts the number of instances that meets the criterion between the parentheses.

The R-R case (*hasGrandparent*(**y,x**) to *hasGrandchild*(**x,y**)) is the most flexible, as it puts the least restrictions on what can be modeled. In the example, the LPD is simply that the probabilities of *hasGrandparent*(**y,x**) are the probabilities of *hasGrandchild*(**x,y**). In the R – F case, there is a requirement that the R-type parent must be functional for the OV that is also the OV of the child variable (**child** in the example). If a child can have two biological fathers, this violates the mutually exclusive na-

ture of *FatherOf*. The embedded constraint exists to enforce this requirement. For those PCs that meet the constraint, the LPD looks for which *hasChild* instance is *True*, and assigns a probability of 1 to the child state that is the same entity as the parent OV. The last case, F – R, is an example where the inverse has additional characteristics. Instead of the grandparent / grandchild inverses, one now has an F-type parent called *PaternalGrandfatherOf* with a R-type child RV (*hasGrandchild*). *hasGrandchild* is not tied so tightly to the parent RV. It is possible for the paternal grandfather to be one person, while the OV in *hasGrandchild* is true for another person. This is because a human child has two grandfathers, of which one is the paternal grandfather. The LPD table provides a probability for this case.

Existence patterns are single parent patterns that describe dependencies between attributes and relationships. In these patterns, the same entity has both the attribute and the relationship under consideration, and the state of one influences the state of the other. Figure 50 gives three examples. In the first, an attribute (a person's sex) affects whether that person can be the mother of another person. In the LPD, all male persons have zero probability, while females have a uniform probability whose value depends on their count. In this example, the list of possible mothers is assumed to be complete.

In the second example, a set of relationships *BossOf* influence whether a manager is a busy person. In this case, the LPD looks at a set of employees and counts if a particular manager is their boss. The higher the count, the higher the degree of busyness. In this example, the states of busyness are a sequence of qualitative levels of busyness,

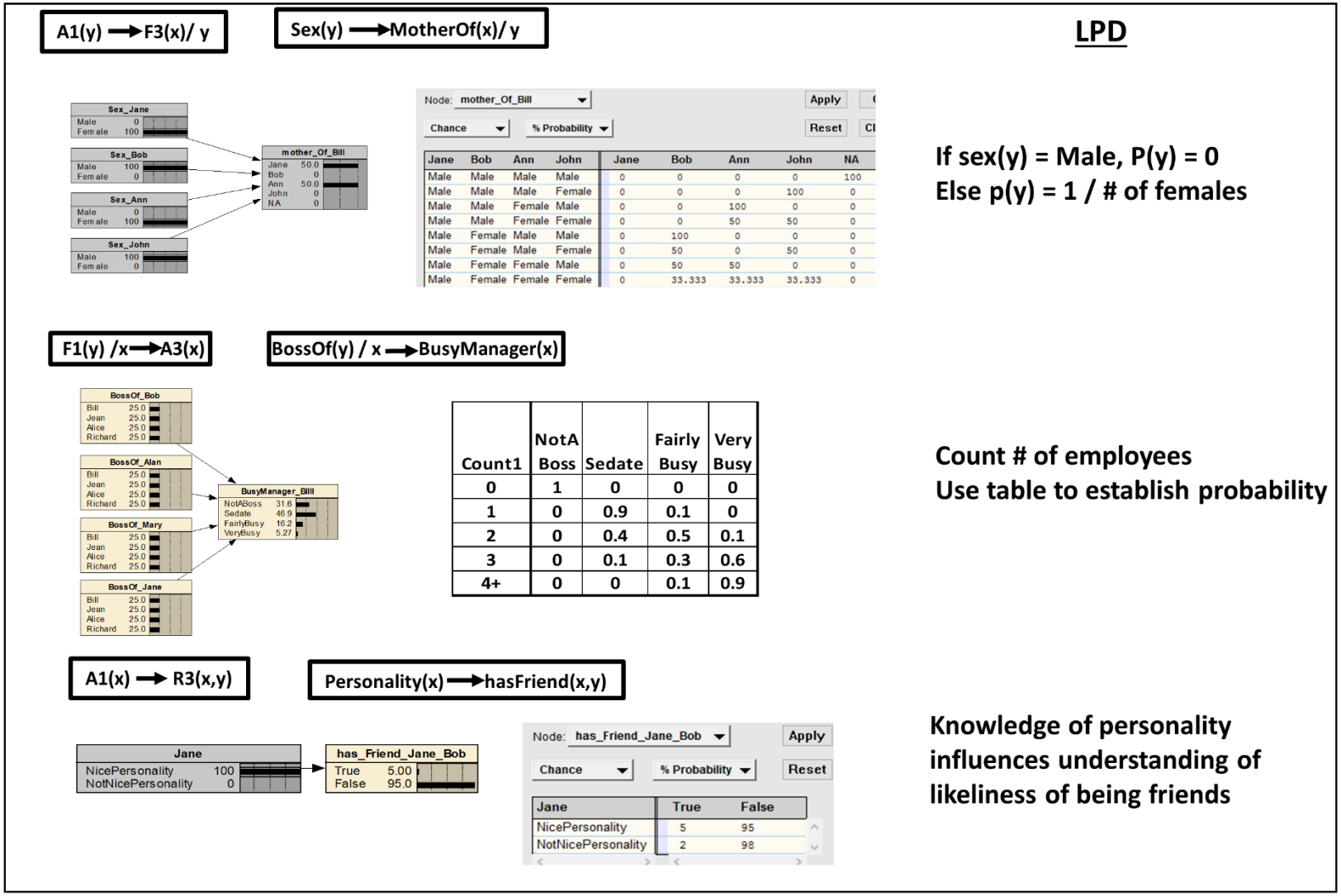


Figure 50: Existence patterns

rather than a yes/no assessment. In the third example, an attribute (*Personality*) influences the likelihood that a person will be friends with another person. The LPD is a straightforward implementation, with conditional probabilities for each of the states of the parents.

APPENDIX C
DECISION TEMPLATE EVALUATION BACKGROUND AND RESULTS

To assess the value of a decision template to a model developer, an experiment was conducted at George Mason University (GMU). A graduate level class on heterogeneous information fusion and decision support had a course project requirement to develop a fusion / decision support system. This appendix provides the background information that describes the top-level project requirements, the decision templates provided to the students, and the analysis results of the collected evaluation information.

C.1 Project Background Information

The objective of the class project is to build a fusion / decision support system for a notional fighter aircraft self-defense system. The system supports a fighter aircraft pilot by fusing the information received from various aircraft sensor systems about threats to the aircraft and recommends appropriate actions to the pilot.

The scenario is that two countries, Blueland and Redland, are at war with each other. They are adjacent countries on a large island, which they also share with a neutral third country, Grayland. A Blueland Air Force fighter unit is preparing to launch a small strike on a target within Redland, using four BlueFighter aircraft. They will face Redland aircraft, the XFighter, YFighter and RedBomber. All are equipped with air-to-air infrared (IR) guided missiles, and the two fighters also have radar guided missiles. For this scenario, there are no cannons on any aircraft. In addition, Redland also has two types of ra-

dar-guided antiaircraft artillery (AAA) systems, four types of radar guided surface to air missile (SAM) systems, and an IR guided man portable SAM system. Grayland previously bought X and Y fighters from Redland and uses them to aggressively maintain its neutrality. Blueland also has IR missile armed BlueBombers. All three countries have unarmed military and civil aircraft. The scenario has several simplifying assumptions to keep the modeling reasonable for the available time.

Each BlueFighter aircraft has a fusion / decision support system to aid the pilot in avoiding or defeating threats. The system architecture naturally decomposes to a fusion element and a decision support element. Each aircraft has six sensors to detect and identify threats, which provide the inputs to the fusion element:

- Radar system (RS) - reports on the numbers, speed, range and heading of aircraft detected in the sector forward of the aircraft
- Radar warning receiver (RWR) - reports the type, operating mode, signal strength, and heading to various radar systems detected in the environment
- Missile warning receiver (MWR) – detects missile launch in the aircraft's vicinity
- Identify friend and foe (IFF) - a system that uses a cryptographically encoded signal to query other aircraft to determine if they are friendly
- Navigation system (NS) – provides the aircraft's location and altitude
- Pilot reporting (PR) – Pilot input that a missile threat was visually detected.

Based on this information, the fusion element determines three things and passes them to the decision support element:

- Identity of the type of threat

- Whether the BlueFighter aircraft is heading into or away from the threat
- Level of threat posed to the aircraft.

The decision support element then determines the appropriate actions to the threat. Four concurrent decisions are made:

- Whether to conduct an extreme defensive maneuver, called a break, to avoid an incoming missile threat with concurrent electronic countermeasures (ECM), or to deviate the flight path to move away from the threat - with or without ECM, or do nothing. Maneuvering negatively affects reaching the target at the scheduled time
- Whether to engage an aircraft threat with an air to air missile, or conduct a feint by placing the radar into missile launch mode without actually launching the missile (thereby activating the threat aircraft's RWR and forcing the pilot to react as if a missile were launched), or doing nothing. There are no air-to-surface missiles
- Whether to turn the radar off to reduce the electronic signature of the aircraft and thereby reduce its detectability to opposing radar detection systems. This also reduces the pilot's situational awareness of threat aircraft
- Whether to turn the IFF off, for the same reason as turning the radar off.

There are several prohibited combinations of choices.

- No missile launch or feint can be done if a break maneuver is selected
- No missile launches can occur if the threat aircraft is not in range, or if there are no missiles available to reach the threat aircraft
- The radar cannot be turned off if a missile launch or feint is selected
- If the IFF is turned on, the radar cannot be turned off.

The system was implemented as a decision graph, using the Netica® modeling tool. The benchmark decision graph model (Figure 51) has 43 nodes, with 22 nodes in the fusion element and 21 nodes in the decision support element.

C.2 Project Decision Templates and Evaluation Requirements

The class was divided into six teams, each with two or three people. Each team received a project design document with the scenario and background material necessary to develop the model, and information to derive the conditional probability tables and the utility node values. In addition, two decision template packages were developed, one for the fusion element (Figure 52 and Figure 53) and one for the decision support element (Figure 54 through Figure 56). The templates are at the specific decision problem level. They give a complete decomposition from the high-level system diagram to the individual decision graph nodes used in the system. Along with the architectural decomposition, supporting documentation graphic structures were developed including an objective hierarchy, decision list, consequence table, and a threat response table. The last two tables were included in the design document given to all students, not in the decision template. The consequence table was given as a set of probabilities of successfully defeating a threat given the defensive responses used. The threat response table, shown in Table 11, gives the optimal defensive responses to different threat levels. The threat level is determined by the following five columns next to threat level. The project design document included a significant amount of information that normally would be part of the decision template. This information was not replicated in the decision template. The template provided additional detailed design information useful in completing the model. Half of

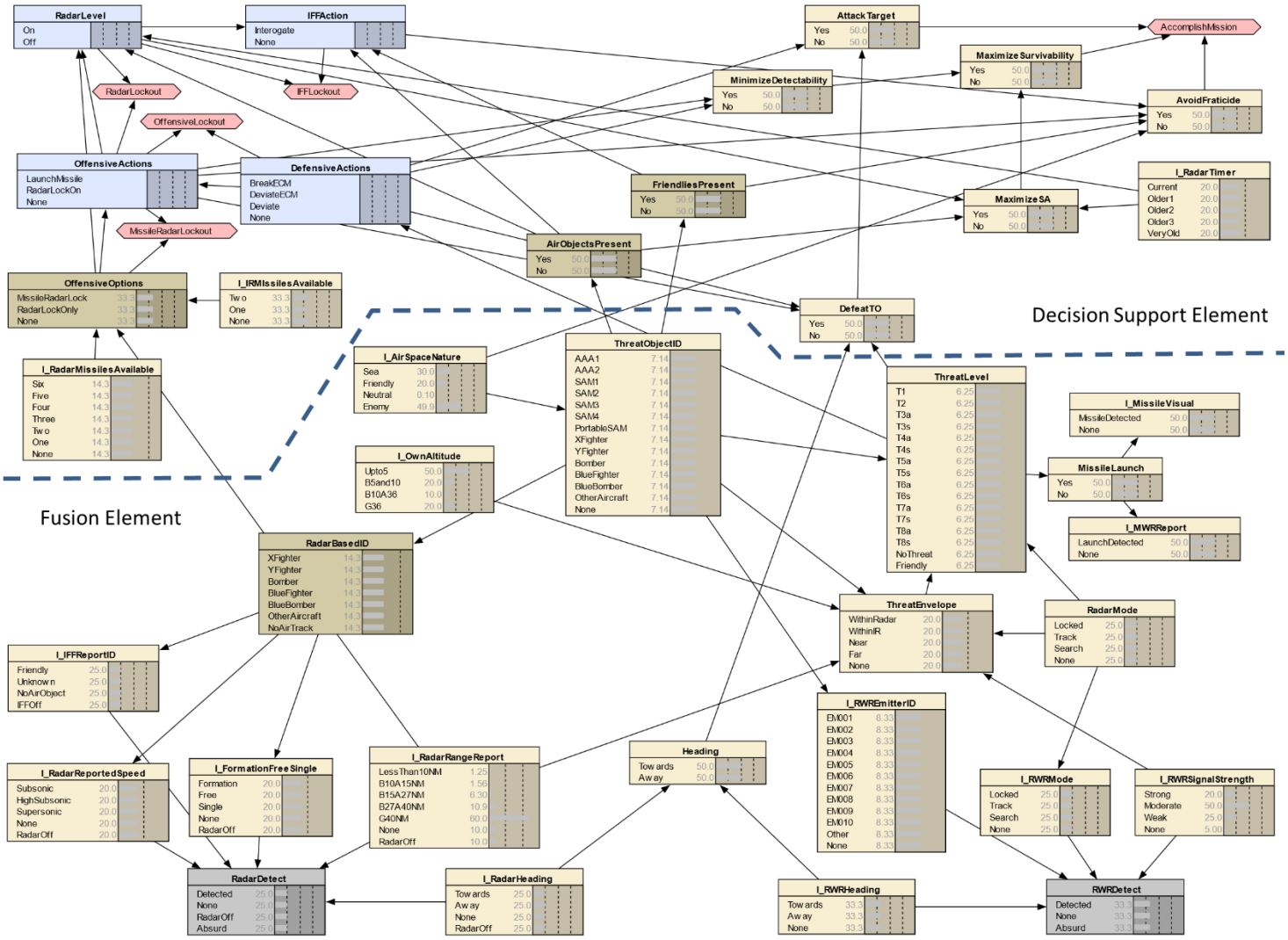


Figure 51: Benchmark model for the fighter aircraft system. Dashed line delineates the two main elements of the system

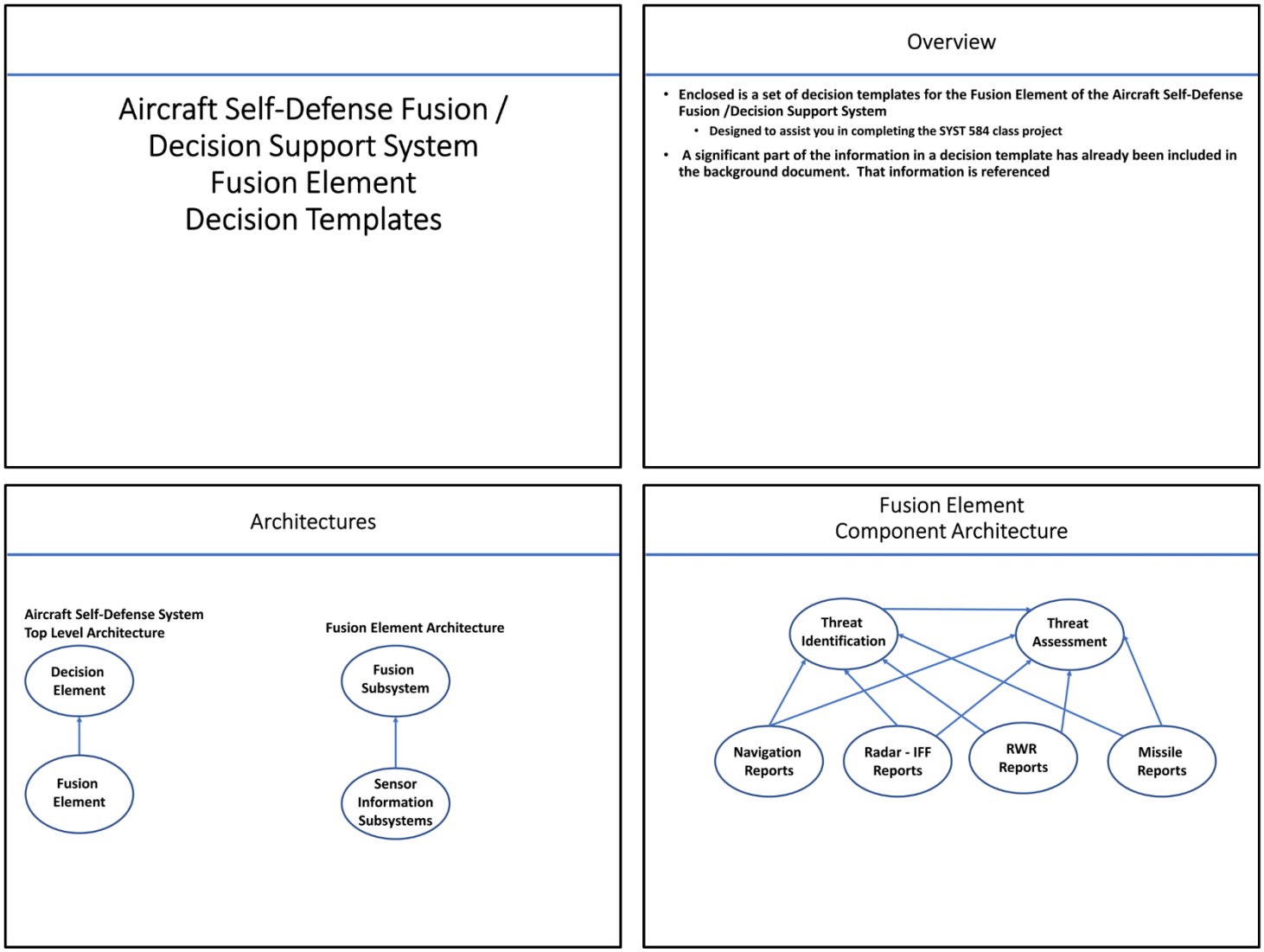


Figure 52: Fusion element decision template (part 1 of 2)

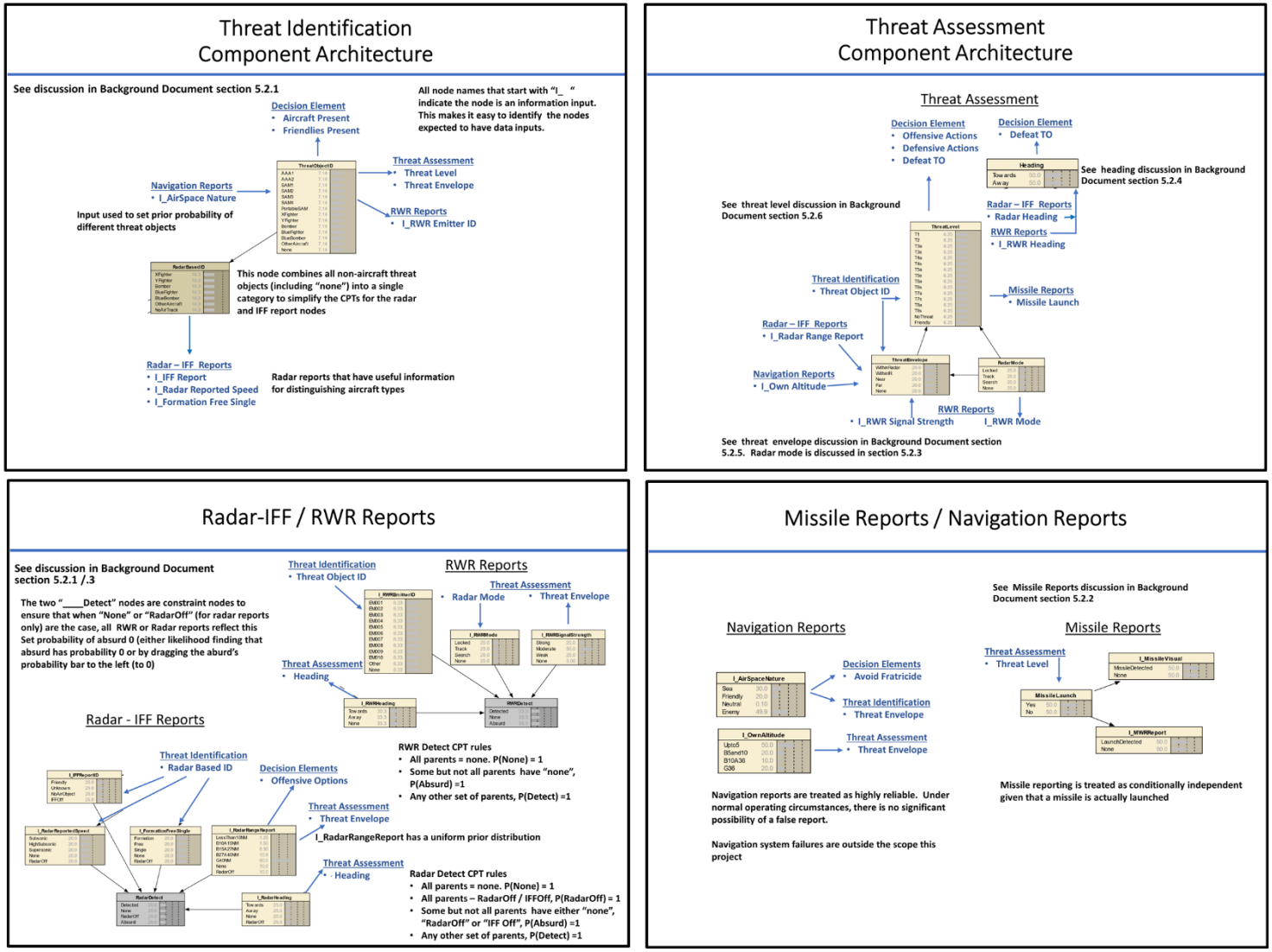


Figure 53: Fusion element decision template (part 2 of 2)

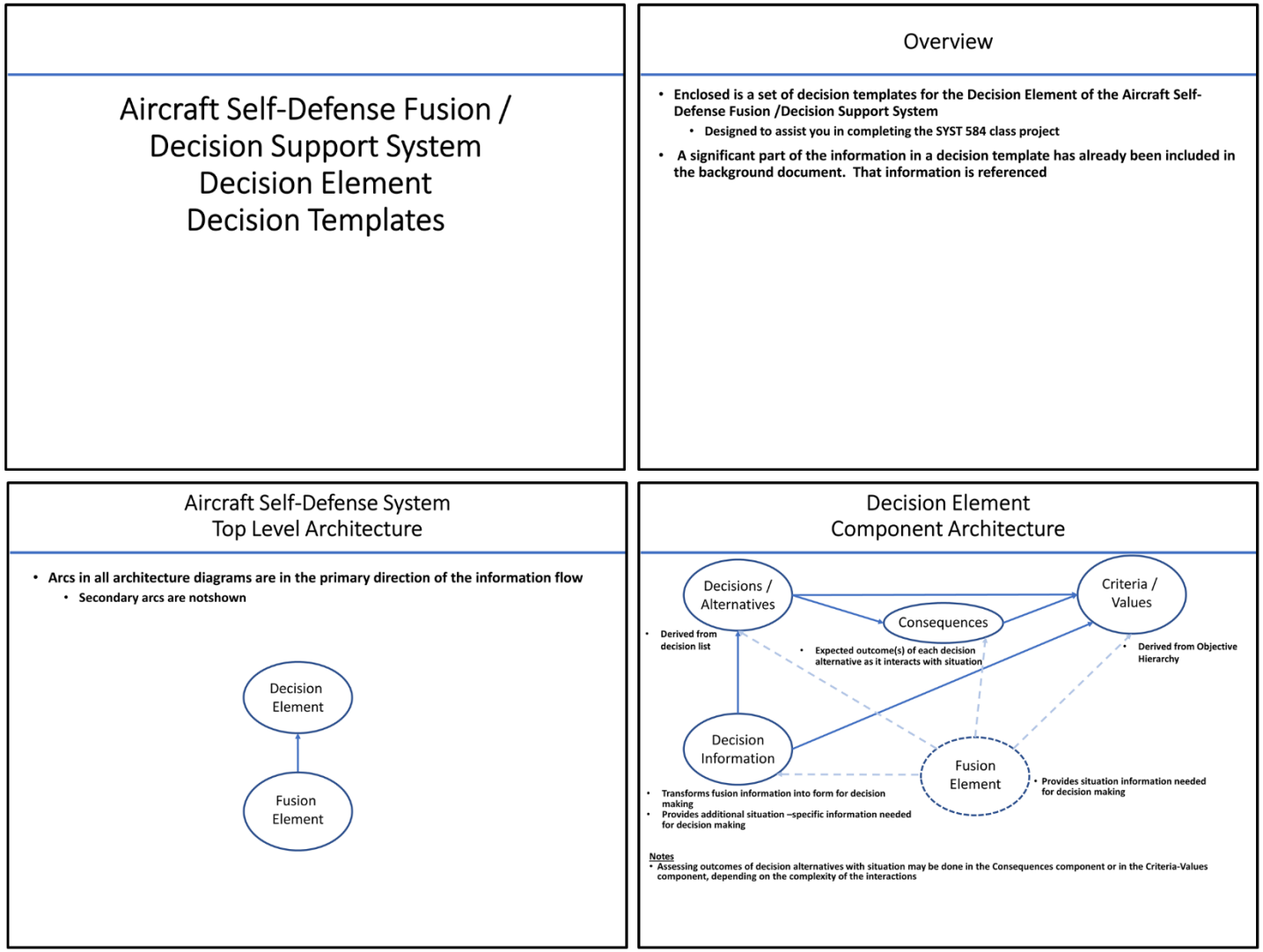


Figure 54: Decision element decision template (part 1 of 3)

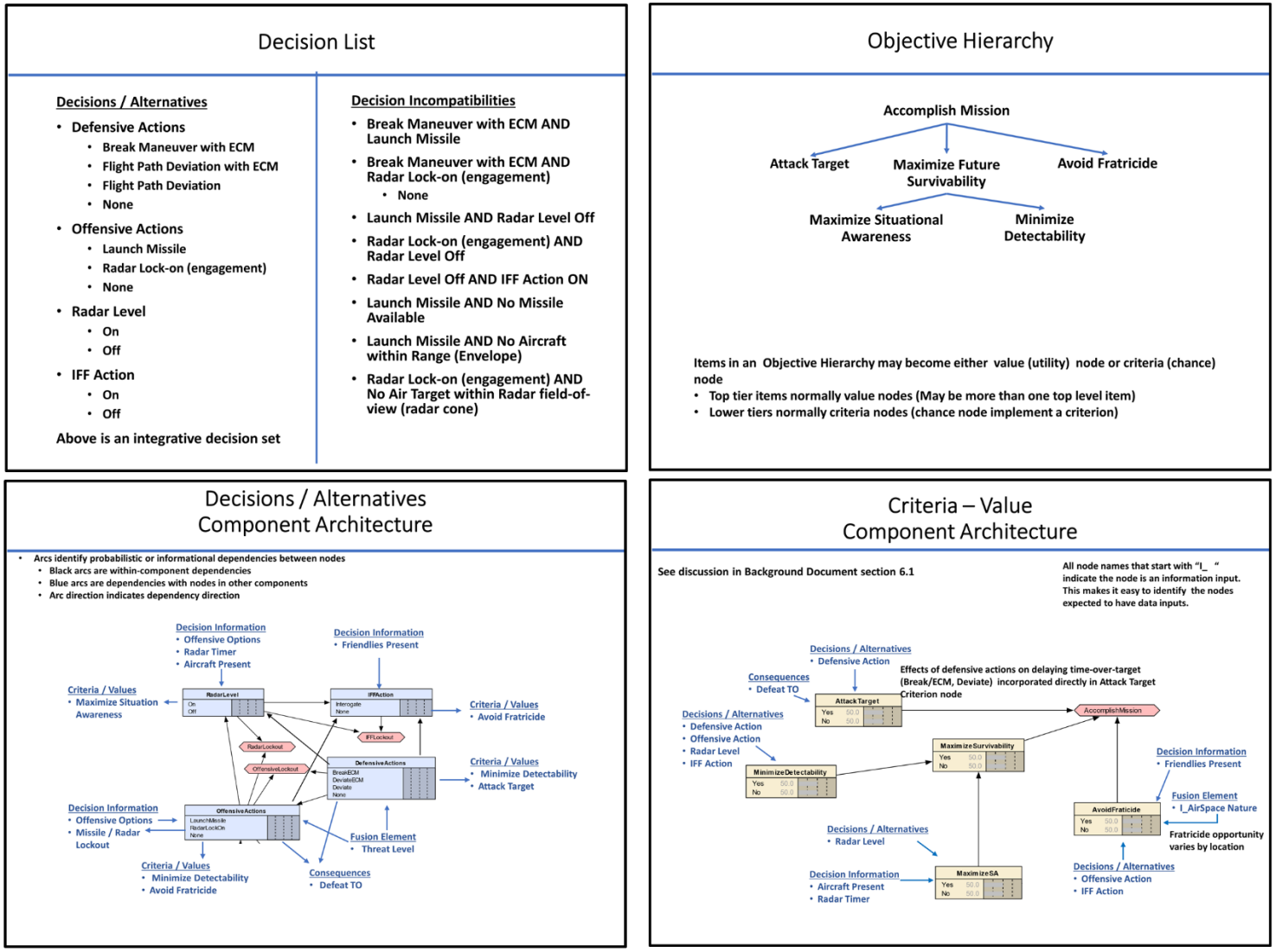


Figure 55: Decision support element decision template (part 2 of 3)

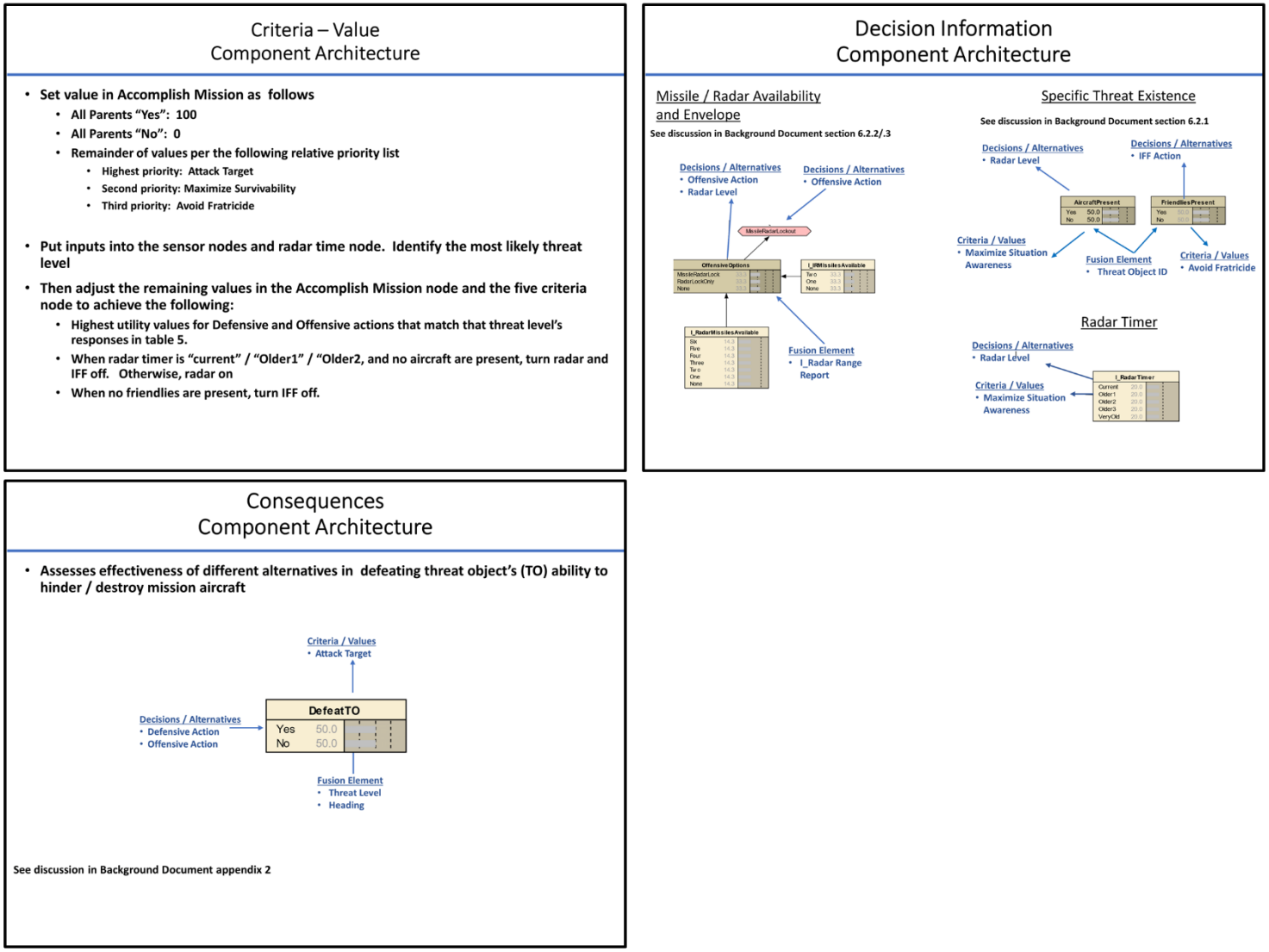


Figure 56: Decision support element decision template (part 3 of 3)

Table 11: Threat response table

Threat Level	Missile Launch	Radar Mode	Envelope	Threat Entity	Heading	Response
1ps (missile attack)	Yes	None	In IR Envelope	Portable SAM	Any	Break w/ ECM
1ar (missile attack)	Yes	Locked	In Radar Envelope	X, YFighter, any Radar SAM	Any	Break w/ ECM
1ai (missile attack)	Yes	Any	In IR Envelope	X, YFighter, Bomber	Any	Break w/ ECM
1g (gun attack)	No	Locked	In Envelope	Any AAA	Any	Break w/ ECM
2	No	Any	In IR Envelope	X,YFighter, Bomber	Any	Launch Missile, Deviate
3a	No	Locked	In Radar Envelope	X,YFighter	Any	Launch Missile, ECM, Deviate
3s	No	Locked	In Radar Envelope	Any Radar SAM	Any	Deviate w/ ECM
4a	No	None, Search, Track	In Radar Envelope	X,Yfighter, Bomber	Any	Launch Missile, ECM, Deviate
4s	No	None, Search, Track	In Radar Envelope	Any AAA or Radar SAM	Any	Deviate w/ ECM
5a	No	Locked	Near	X,YFighter	Any	Launch Missile, ECM, Deviate
5s	No	Locked	Near	Any AAA or Radar SAM	Any	Deviate , ECM
6a	No	Track	Near	X,YFighter, Bomber	Away - Ignore / Toward - Deviate with engage	
6s	No	Track	Near	Any AAA or Radar SAM	Away - Ignore / Toward - Deviate	
7a	No	None, Search	Near	X,YFighter, Bomber	Away - Ignore / Toward - Deviate with engage	
7s	No	None, Search	Near	Any AAA or Radar SAM	Away - Ignore / Toward - Deviate	
8a	No	Any	Far	X,YFighter, Bomber	Away - Ignore / Toward - Deviate	
8s	No	Any	Far	Any AAA or Radar SAM	Away - Ignore / Toward - Deviate	
No Threat	No	Any	Any	Other Aircraft, None	Any	Ignore
Friendly	No	Any	Any	Blue Fighter, Blue Bomber	Any	Ignore

the teams received the fusion element decision template; the other half received the decision support element decision template. Students were asked to do two things:

- Track the amount of time they spent on the project, separated into four categories: general preparation, fusion element development, decision element development, and testing and report writing
- Complete a seven-question evaluation after they turned in their projects

The time collection / evaluation questionnaire is shown in Figure 57 (front page) and Figure 58 (back page).

SYST 584 Data Fusion / Decision Support Study Evaluation

Team Identifier:_____

Which Decision Template did you get (circle one)? Fusion Element Decision Element

Regarding the decision template you received, please circle the response that best fits your judgement for that question

1. The decision template provided was clear and understandable

1	2	3	4	5
Poor	Fair	Average	Good	Excellent

2. The information in the decision template was useful in completing the project

1	2	3	4	5
Not helpful at all	Marginally helpful	Helpful for some areas	Helpful for many areas	Very Helpful

3. The decision template contributed to understanding the overall requirements of the project

1	2	3	4	5
Contradicted or confused the project requirements	Provided some confusion	No significant effect	Contributed some understanding	Contributed significantly

4. Consider the effort you put in developing the fusion element of the model, and the effort you put in developing the fusion element. You received a template for one of them. Compared to the element for which no template was provided, the template made the effort for the element it described:

1	2	3	4	5
Harder to perform	Somewhat harder	No significant difference	Easier	Much easier

The following questions assess the project as a whole

5. The project's level of difficulty was

1	2	3	4	5
Very Easy	Easy	Average	Hard	Very Hard

Figure 57: Front page of project evaluation form

6. The material provided for the project (Background information and template) was

1	2	3	4	5
Significantly incomplete	Missing some key information	Generally adequate to complete the project	Mostly complete and comprehensive	Very complete and comprehensive

7. The project was useful in developing skills in heterogenous data fusion

1	2	3	4	5
Not useful	Marginally useful	Somewhat useful	Useful	Very useful

If you collected the amount of time you spent on this project, please enter the information below. If you did not, ignore this section

Total Time (hours) you (individual team member) spent on:

- General preparation: _____
- Developing and Testing Fusion Element: _____
- Developing Decision Support Element:_____
- Integration, Evaluation and Report Writing: _____

Any comments you have on the decision template or the project (Optional):

Figure 58: Back page of project evaluation form

The experiment plan was reviewed and approved by the GMU Institutional Review Board process, and the experiment executed in accordance with the plan. Participation in the study was voluntary and did not affect the students' grades.

C.3 Study results and analysis

Of the 13 students in the class, 12 participated in the study. As discussed in chapter 6, both the time data and the students' evaluations were analyzed. The time data is discussed in chapter 6 and there is nothing to add here. Table 12 provides the template / project evaluation scores provided by each student. The student identifier indicated which template they received (D – Decision, F- Fusion). The basic evaluation scheme was to compare the amount of time students spent on the element for which they had a decision template versus the amount of time on the element for which they did not. Since the model itself was roughly balanced between the two elements, it was expected that they should take about the same amount of time to develop.

The group average and standard deviation for each question are shown at the bottom. As discussed in Section 6.4, there are two individuals for whom there are questions

Table 12: Questionnaire results

	Template Questions				Project Questions				
	1	2	3	4	5	6	7	Average Template	Average Project
Student									
D1	4	5	4	4	4	3	4	4.3	3.7
D2	4	5	5	4	4	4	4	4.5	4.0
D3	4	5	5	4	5	4	4	4.5	4.3
D4	4	5	5	5	4	3	5	4.8	4.0
D5	4	3	4	5	4	4	4	4.0	4.0
D6	5	5	5	1	5	5	5	4.0	5.0
D7	5	5	4	4	3	3	4	4.5	3.3
F1	3	3	4	2	3	3	3	3.0	3.0
F2	5	4	5	4	3	4	4	4.5	3.7
F3	2	3	4	4	5	2	3	3.3	3.3
F4	1	3	2	1	5	4	2	1.8	3.7
F5	-	-	-	4	3	4	4	4.0	3.7
Average	3.7	4.2	4.3	3.5	4.0	3.6	3.8	3.9	3.8
Std Dev	1.3	1.0	0.9	1.4	0.9	0.8	0.8	0.9	0.5

Average by Template Group		
Template	Average Template	Average Project
Decision	4.4	4.0
Fusion	3.2	3.5

about the input. D6 may have misunderstood question #4, since the data given there is significantly inconsistent with the other answers. F4 indicated there was a specific circumstance that made completing the project difficult. In the analysis that follows, no data points were eliminated for being outliers, but in several cases, elimination of those two data points slightly changes the results. This will be pointed out in the discussion.

The primary observations regarding the scores are covered in section 6.4. This appendix will cover some additional correlation analysis that was done. This looked for nonzero correlations among the different scores (see Figure 59 for results). The number next to a dot indicates that more than one answer fell on that chart point. The top row assessed correlations between the student's averaged template score and the three project-related scores. The one in the top left looks at the student's average template ratings with how they rated the project difficulty. The chart includes a trend line showing that the average template assessment goes down as the assessment of project difficulty increased, although there is a low coefficient of determination (r^2). When the lowest student template score was removed, the trend line is flat.

The second graph, in the top middle, plots the student's average template score versus their assessment of the overall quality of the total information package (design document and template). One would expect that there would be positive trend between how they scored the template and how they scored the total package. If the students viewed the decision templates the same as they viewed the entire decision package, the trend line would follow the red line on the chart. Seven of 12 students rated the templates

higher than they did the total package, while two rated them lower. There is a small positive correlation, but it appears to be insignificant.

The graph in the top right plots the average template score against the student's assessment of the project's learning value. As discussed in section 6.4, there was a strong correlation between a student's assessment of the project's learning value and the assessment of the decision templates.

The bottom row of Figure 59 looks for correlations between project-level assessment questions. No significant correlations were found. The summary results are presented in section 6.4.

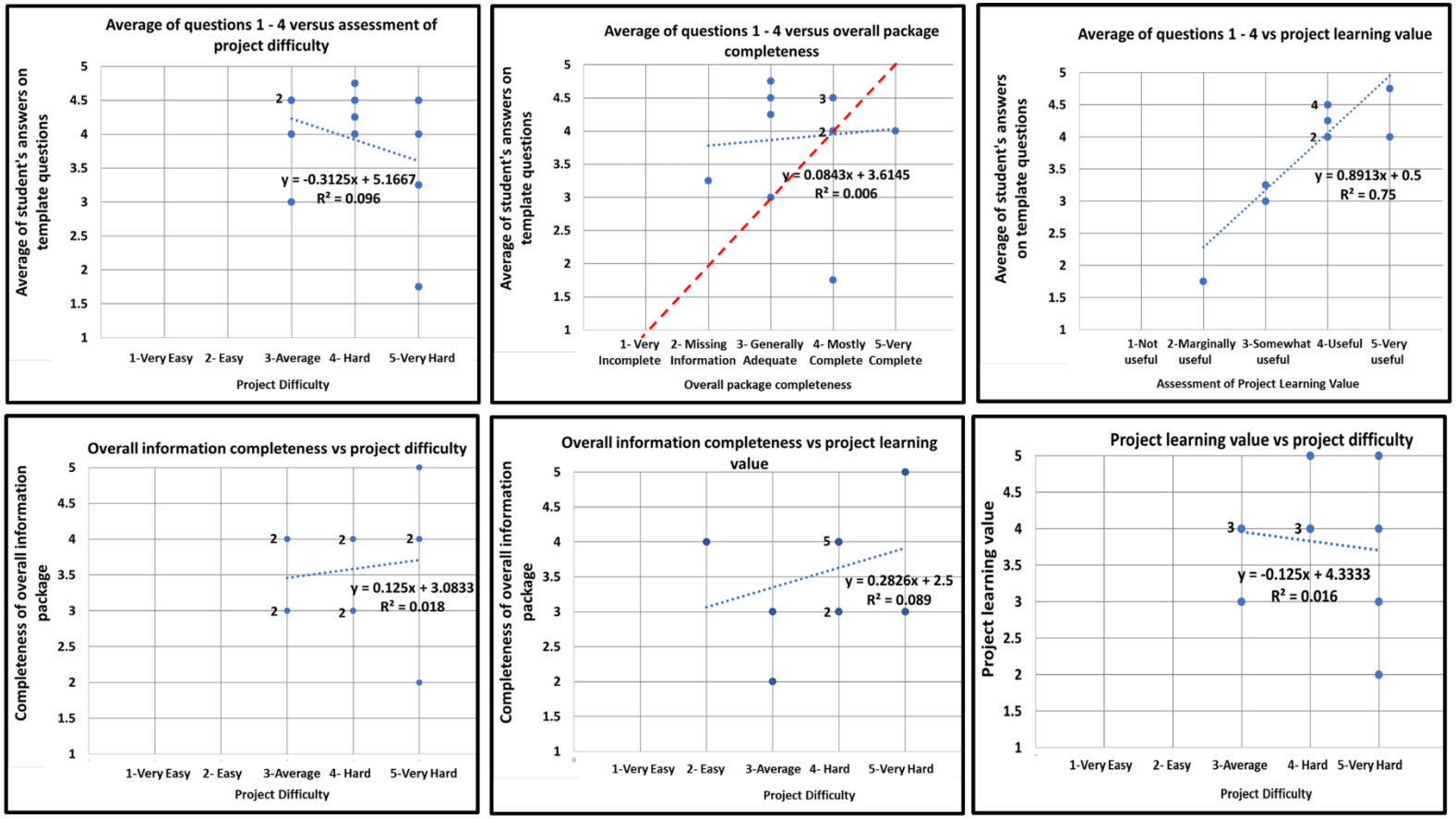


Figure 59: Correlation analysis between average template scores versus different project scores (top row) and among project scores (bottom row)

REFERENCES

Ackoff, Russell Lincoln. 1974. Redesigning the Future: A Systems Approach to Societal Problems. New York: Wiley.

Adomavicius, Gediminas, and Alexander Tuzhilin. 2015. "Context-Aware Recommender Systems." In Recommender Systems Handbook, 191–226. Springer.

Alegre, Unai, Juan Carlos Augusto, and Tony Clark. 2016. "Engineering Context-Aware Systems and Applications: A Survey." Journal of Systems and Software 117: 55–83.

Almond, Russell G., Joris Mulder, Lisa A. Hemat, and Duanli Yan. 2009. "Bayesian Network Models for Local Dependence among Observable Outcome Variables." Journal of Educational and Behavioral Statistics 34 (4): 491–521.

Al-Rousan, Thamer, Shahida Sulaiman, and Rosalina Abdul Salam. 2009. "Project Management Using Risk Identification Architecture Pattern (RIAP) Model: A Case Study on a Web-Based Application." In Proceedings of the 16th Asia-Pacific Software Engineering Conference, 449–56. IEEE.

American Institute of Chemical Engineers, ed. 2008. Guidelines for Hazard Evaluation Procedures. 3rd ed. Hoboken, N.J: CCPS/AIChE/Wiley Interscience.

Anand, Vibha, and Stephen M. Downs. 2008. "Probabilistic Asthma Case Finding: A Noisy or Reformulation." In AMIA Annual Symposium Proceedings, 2008:6. American Medical Informatics Association.

Baader, Franz, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, eds. 2003. The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press.

Bannach, David, Oliver Amft, and Paul Lukowicz. 2008. "Rapid Prototyping of Activity Recognition Applications." IEEE Pervasive Computing 7 (2).

Barclay, Scott, Rex V. Brown, Clinton W III Kelly, Cameron R. Peterson, Lawrence D. Phillips, and Judith Selvidge. 1977. "Handbook for Decision Analysis." TR-77-6-30. Decisions and Designs Inc, Mclean VA. http://www.dtic.mil/docs/citations/ADA049221.

Beamon, Bridget, and Mohan Kumar. 2010. “HyCoRE: Towards a Generalized Hierarchical Hybrid Context Reasoning Engine.” In Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference On, 30–36. IEEE.

Belton, Valerie, and Theodor Stewart. 2002. Multiple Criteria Decision Analysis: An Integrated Approach. Springer Science & Business Media.

———. 2010. “Problem Structuring and Multiple Criteria Decision Analysis.” In Trends in Miltiple Criteria Decision Analysis, edited by Matthias Ehrgott, Jose Rui Figueria, and Salvatore Greco, 209–39. Springer Science+Business Media.

Berkeley, Dina, and Patrick Humphreys. 1982. “Structuring Decision Problems and the ‘Bias Heuristic.’” Acta Psychologica 50 (3): 201–52.

Bermejo, Inigo, Francisco Javier Díez, Paul Govaerts, and Bart Vaerenberg. 2013. “A Probabilistic Graphical Model for Tuning Cochlear Implants.” In Conference on Artificial Intelligence in Medicine in Europe, 150–155. Springer.

Bernoulli, Daniel. 1738. “Exposition of a New Theory on the Measurement of Risk (Translated and Republished 1954).” Translated by Louise Sommer. Econometrica 22 (1): 23.

Bettini, Claudio, Oliver Brdiczka, Karen Henricksen, Jadwiga Indulska, Daniela Nicklas, Anand Ranganathan, and Daniele Riboni. 2010. “A Survey of Context Modelling and Reasoning Techniques.” Pervasive and Mobile Computing 6 (2): 161–180.

Bonczek, R. H., C. W. Holsapple, and Andrew B. Whinston. 1981. Foundations of Decision Support Systems. Operations Research and Industrial Engineering. New York: Academic Press.

Brachman, Ronald J., and Hector J. Levesque. 2004. Knowledge Representation and Reasoning. Amsterdam; Boston: Morgan Kaufmann.

Brennan, Alan, Stephen E. Chick, and Ruth Davies. 2006. “A Taxonomy of Model Structures for Economic Evaluation of Health Technologies.” Health Economics 15: 1295–1310.

Brézillon, Patrick. 1999. “Context in Artificial Intelligence: I. A Survey of the Literature.” Computers and Artificial Intelligence 18 (4): 321–40.

Brézillon, Patrick, and Juliette Brézillon. 2008. “Context-Sensitive Decision Support Systems in Road Safety.” Information Systems and E-Business Management 6 (3): 279–93.

Brown, Rex V., and Jacob W. Ulvila. 1976. “Selecting Analytic Approaches for Decision Situations. A Matching of Taxonomies.” http://www.dtic.mil/cgi-bin/GetTRDoc?Location=U2&doc=GetTRDoc.pdf&AD=ADA033349.

Buchanan, Leigh, and Andrew O’Connell. 2006. “A Brief History of Decision Making.” Harvard Business Review 84 (1): 32.

Buzan, Tony, and Barry Buzan. 1996. The Mind Map Book: How to Use Radiant Thinking to Maximize Your Brain’s Untapped Potential. New York: Plume.

Carvalho, Rommel N., Kathryn B. Laskey, and Paulo CG Costa. 2017. “PR-OWL–a Language for Defining Probabilistic Ontologies.” International Journal of Approximate Reasoning 91: 56–79.

Carvalho, Rommel Novaes. 2011. “Probabilistic Ontology: Representation and Modeling Methodology.” Ph.D Thesis, George Mason University.

Clemen, Robert T. 1997. Making Hard Decisions: An Introduction to Decision Analysis. 2 edition. Belmont, Calif: Duxbury.

Clemen, Robert T., and Terence Reilly. 2014. Making Hard Decisions with Decision-Tools. 3rd edition. South-Western Cengage Learning.

Consortium, World Wide Web. 2012. “OWL 2 Web Ontology Language Document Overview.”

Cooper, Gregory F. 1988. “A Method for Using Belief Networks as Influence Diagrams.” In Proceedings of the Fourth Conference on Uncertainty in Artificial Intelligence, 55–63.

Costa, P. C. G., and K. B. Laskey. 2005. “Multi-Entity Bayesian Networks without Multi-Tears.” The Volgenau School of Information Technology and Engineering, George Mason University.

Costa, Paulo Cesar G. 2005. Bayesian Semantics for the Semantic Web. PhD Thesis -- George Mason University.

Coutaz, Joëlle, and Gaëtan Rey. 2002. “Foundations for a Theory of Contextors.” In Computer-Aided Design of User Interfaces III, 13–33. Springer.

Dean, James W., Mark P. Sharfman, and Cameron M. Ford. 1991. “Strategic Decision-Making: A Multiple Context Framework.” Advances in Information Processing in Organizations 4: 77–110.

Devlin, Keith. 2008. “Situation Theory and Situation Semantics.” In Handbook of the History of Logic, edited by Dov Gabbay and John Woods, 7:601–64. Elsevier.

Dey, Anind K. 2001. “Understanding and Using Context.” Personal and Ubiquitous Computing 5 (1): 4–7.

Díez, Francisco Javier, Manuel Luque, Caroline Leonore König, and Iñigo Bermejo. 2014. “Decision Analysis Networks.” CISIAD-14-01. Madrid, Spain: UNED.

Dillon, S. M. 2002. “Understanding the Decision Problem Structuring of Executives.” Waikato, New Zealand: University of Waikato.

Dıez, F. Javier, and Marek J. Druzdzel. 2006. “Canonical Probabilistic Models for Knowledge Engineering.” CISIAD-06-01. Madrid, Spain: UNED.

Domingos, Pedro, and Daniel Lowd. 2009. Markov Logic: An Interface Layer for Artificial Intelligence. Vol. 3..

Enderton, Herbert B. 2001. A Mathematical Introduction to Logic. 2nd ed. San Diego: Harcourt/Academic Press.

Endsley, Mica R. 1995. “Toward a Theory of Situation Awareness in Dynamic Systems.” Human Factors 37: 32–64.

Ericson, Clifton. 1999. “Fault Tree Analsyis: A History.” In Proceedings of the 17th International Systems Safety Conference.

Etzioni, Amitai. 1989. “Humble Decision Making.” Harvard Business Review, 122–26.

Fenton, Norman, Martin Neil, and David A. Lagnado. 2013. “A General Structure for Legal Arguments about Evidence Using Bayesian Networks.” Cognitive Science 37 (1): 61–102.

Fenton, Norman, Martin Neil, David Lagnado, William Marsh, Barbaros Yet, and Anthony Constantinou. 2016. “How to Model Mutually Exclusive Events Based on Independent Causal Pathways in Bayesian Network Models.” Knowledge-Based Systems 113: 39–50.

Fenz, Stefan, Simon Parkin, and Aad van Moorsel. 2011. “A Community Knowledge Base for IT Security.” IT Professional 13 (3): 24–30.

Franco, L. Alberto, and Gilberto Montibelller. 2009. “Problem Structuring for Multi-Criteria Decision Analysis Interventions.” London : LSE, Dep. of Operational Research..

Friend, J. K., and Allen Hickling. 2005. Planning Under Pressure : The Strategic Choice Approach. Amsterdam: Routledge.

Gerven, M. A. J. van, Peter JF Lucas, and Th P. van der Weide. 2008. “A Generic Qualitative Characterization of Independence of Causal Influence.” International Journal of Approximate Reasoning 48 (1): 214–236.

Getoor, Lise, Nir Friedman, Daphne Koller, Avi Pfeffer, and Ben Taskar. 2007. “Probabilistic Relational Models.” In Introduction to Relational Statistical Models, edited by Lise Getoor and Ben Taskar. The MIT Press.

Getoor, Lise, and John Grant. 2006. “PRL: A Probabilistic Relational Language.” Machine Learning 62 (1): 7–31.

Golestan, Keyvan, Ridha Soua, Fakhri Karray, and Mohamed S. Kamel. 2016. “Situation Awareness within the Context of Connected Cars: A Comprehensive Review and Recent Trends.” Information Fusion 29: 68–83.

Goodman, Noah, Vikash Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. 2012. “Church: A Language for Generative Models.” ArXiv Preprint ArXiv:1206.3255. https://arxiv.org/abs/1206.3255.

Granger, Charles H. 1964. The Hierarchy of Objectives. Vol. 42. 3. Harvard Business Review.

Gregory, R., L. Failing, M. Harstone, G. Long, T. McDaniels, and D. Ohlson. 2012. Structured Decision Making: A Practical Guide to Environmental Management Choices. Chichester, West Sussex ; Hoboken, N.J: Wiley-Blackwell.

Grimm, Stephan. 2010. “Knowledge Representation and Ontologies.” In Scientific Data Mining and Knowledge Discovery, edited by Mohamed Medhat Gaber, 111–37. Berlin, Heidelberg: Springer Berlin Heidelberg.

Grosan, Crina, and Ajith Abraham. 2011. Intelligent Systems: A Modern Approach. Vol. 17. Berlin, Heidelberg: Springer Berlin Heidelberg.

Grünig, Rudolf, and Richard Kühn. 2013. Successful Decision-Making: A Systematic Approach to Complex Problems. Translated by Anthony Clark, Claire O’Dea, and Maude Montani. 3rd ed. 2013 edition. New York: Springer.

Gundersen, Odd Erik. 2013. “Situational Awareness in Context.” In Proceedings of the 8th International and Interdisciplinary Conference, CONTEXT 2013, edited by Patrick Brézillon, Patrick Blackburn, and Richard Dapoigny, 8175:274–87. Annecy, France: Springer Berlin Heidelberg.

Han, Paul K. J., William M. P. Klein, and Neeraj K. Arora. 2011. “Varieties of Uncertainty in Health Care.” Medical Decision Making 31 (6): 828–838.

Heckerman, David, Chris Meek, and Daphne Koller. 2007. “Probabilistic Entity-Relationship Models, PRMs, and Plate Models.” Introduction to Statistical Relational Learning, 201–238.

Helsper, Eveline M., and Linda C. van der Gaag. 2005. “Generic Knowledge Structures for Probabilistic-Network Engineering.” UU-CS-2005-014. Utrecht, The Nederlands: Institute of Information and Computing Sciences, Utrecht University.

Herman, Ivan, Ben Adida, Manu Sporny, and Mark Birbeck, eds. 2015. “RDFa 1.1 Primer - Third Edition. Rich Structured Data Markup for Web Documents.” W3C Consortium. https://www.w3.org/TR/2015/NOTE-rdfa-primer-20150317/.

Howard, Catherine. 2010. “Knowledge Representation and Reasoning for a Model-Based Approach to Higher Level Information Fusion.”

Howard, Catherine, and Markus Stumptner. 2014. “A Survey of Directed Entity-Relation–Based First-Order Probabilistic Languages.” ACM Computing Surveys (CSUR) 47 (1): 4.

Howard, R. A. 1966. “Decision Analysis: Applied Decision Theory.” In Proceedings of the Fourth International Conference on Operational Research, 55–71. New York: Wiley-Interscience.

Howard, Ronald A., and James E. Matheson. 1984. “Influence Diagrams.” In Readings on the Principles and Applications of Decision Analysis, edited by Ronald A. Howard and James E. Matheson, 2:720–762. Menlo Park, CA: Strategic Decisions Group.

Jensen, Finn. 1996. An Introduction To Bayesian Networks. Taylor & Francis.

Jensen, Finn V., Thomas D. Nielsen, and Prakash P. Shenoy. 2006. “Sequential Influence Diagrams: A Unified Asymmetry Framework.” International Journal of Approximate Reasoning 42 (1): 101–118.

Jensen, Finn V., and Thomas Dyhre Nielsen. 2007. Bayesian Networks and Decision Graphs. 2nd ed. Information Science and Statistics. New York: Springer.

———. 2013. “Probabilistic Decision Graphs for Optimization under Uncertainty.” Annals of Operations Research 204 (1): 223–248..

Jensen, Frank, Finn Verner Jensen, and Soren L. Dittmer. 1994. “From Influence Diagrams to Junction Trees.” In Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence, 367–73. http://arxiv.org/abs/1302.6824.

Jones, Dawna. 2014. Decision-Making for Dummies. Hoboken, NJ: John Wiley & Sons, Inc.

Kazemi, Seyed Mehran, David Buchman, Kristian Kersting, Sriraam Natarajan, and David Poole. 2014. “Relational Logistic Regression.” In KR. http://www.aaai.org/ocs/index.php/KR/KR14/paper/download/8013/7968.

Kazemi, Seyed Mehran, Bahare Fatemi, Alexandra Kim, Zilun Peng, Moumita Roy Tora, Xing Zeng, Matthew Dirks, and David Poole. 2017. “Comparing Aggregators for Relational Probabilistic Models.” In ArXiv:1707.07785 [Cs, Stat]. Sydney, Australia. http://arxiv.org/abs/1707.07785.

Keefer, Donald L., Craig W. Kirkwood, and James L. Corner. 2004. “Perspective on Decision Analysis Applications, 1990–2001.” Decision Analysis 1 (1): 4–22.

Keeney, Ralph L. 1992. Value-Focused Thinking : A Path to Creative Decisionmaking. Cambridge, MA, USA: Harvard University Press.

Keeney, Ralph L., and Howard Raiffa. 1976. Decisions with Multiple Objectives : Preferences and Value Tradeoffs. Wiley Series in Probability and Mathematical Statistics. New York: Wiley.

Kersting, Kristian. 2012. “Lifted Probabilistic Inference.” In ECAI, 33–38.

Kimmig, Angelika, Lilyana Mihalkova, and Lise Getoor. 2015. “Lifted Graphical Models: A Survey.” Machine Learning 99 (1): 1–45.

Kjaerulff, Uffe B., and Anders L. Madsen. 2013. Bayesian Networks and Influence Diagrams: A Guide to Construction and Analysis. 2nd ed. Information Science and Statistics. New York, NY: Springer.

Klein, Gary A., R. Calderwood, and A. Clinton-Cirocco. 1988. Rapid Decision Making on the Fire Ground /. Alexandria, Va : U.S. Army Research Institute for the Behavioral and Social Sciences.

Klein, Gary, Roberta Calderwood, and Anne Clinton-Cirocco. 2010. “Rapid Decision Making on the Fire Ground: The Original Study Plus a Postscript.” Journal of Cognitive Engineering and Decision Making 4 (3): 186–209.

Kleindorfer, Paul R., Howard Kunreuther, and Paul JH Schoemaker. 1993. Decision Sciences: An Integrative Perspective. Cambridge University Press.

Koller, Daphne, and Nir Friedman. 2009. Probabilistic Graphical Models: Principles and Techniques. MIT press.

Koller, Daphne, Alon Levy, and Avi Pfeffer. 1997. “P-CLASSIC: A Tractable Probabilistic Description Logic.” AAAI/IAAI 1997: 390–397.

Koller, Daphne, and Avi Pfeffer. 1997. “Object-Oriented Bayesian Networks.” In Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence, 302–13. Providence, Rhode Island, USA.

König, Caroline Leonore. 2012. “Representing Asymmetric Decision Problems with Decision Analysis Networks.” Madrid, Spain: Universidad Nacional de Educación a Distancia (UNED).

Kornyshova, Elena, and Rébecca Deneckère. 2010. “Decision-Making Ontology for Information System Engineering.” In Lecture Notes in Computer Science, 104–117. Springer.

Kruchten, Philippe. 2004. “An Ontology of Architectural Design Decisions in Software Intensive Systems.” In 2nd Groningen Workshop on Software Variability, 54–61.

Kwakkel, Jan H., Warren E. Walker, and Marjolijn Haasnoot. 2016. “Coping with the Wickedness of Public Policy Problems: Approaches for Decision Making under Deep Uncertainty.” Journal of Water Resources Planning and Management 142 (3).

Laskey, Kathryn Blackmond. 2008. “MEBN: A Language for First-Order Bayesian Knowledge Bases.” Artificial Intelligence 172 (2): 140–178.

Laskey, Kathryn Blackmond, and Suzanne M. Mahoney. 1997. “Network Fragments: Representing Knowledge for Constructing Probabilistic Models.” In Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence, 334–41.

Latfi, Fatiha, Bernard Lefebvre, and Céline Descheneaux. 2007. “Ontology-Based Management of the Telehealth Smart Home, Dedicated to Elderly in Loss of Cognitive Autonomy.” In OWLED. Vol. 258.

Lipshitz, Raanan, and Orna Strauss. 1997. “Coping with Uncertainty: A Naturalistic Decision-Making Analysis.” Organizational Behavior and Human Decision Processes 69 (2): 149–163.

Liu, Ou, Qijia Tian, and Jian Ma. 2004. “A Fuzzy Description Logic Approach to Model Management in R&D Project Selection.” PACIS 2004 Proceedings, 6.

López, Claudia, Hernán Astudillo, and Luiz Marcio Cysneiros. 2008. “Semantic-Aided Interactive Identification of Reusable NFR Knowledge Fragments.” In Lecture Notes in Computer Science, 5333:324–33. Monterrey, Mexico: Springer Berlin Heidelberg.

Lucas, Peter JF. 2005. “Bayesian Network Modelling through Qualitative Patterns.” Artificial Intelligence 163 (2): 233–263.

Lukasiewicz, Thomas. 2008. “Expressive Probabilistic Description Logics.” Artificial Intelligence 172 (6): 852–883.

Madsen, Anders L., and Dennis Nilsson. 2001. “Solving Influence Diagrams Using HUGIN, Shafer-Shenoy and Lazy Propagation.” In Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence.

Magrini, Alessandro, Davide Luciani, and Federico Mattia Stefanini. 2016. “A Generalization of the Noisy-MAX Parameterization for Biomedical Applications.”

Mahoney, Suzanne M., and Kathryn Blackmond Laskey. 1998. “Constructing Situation Specific Belief Networks.” In Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI1998).

Manheim, Marvin L., and Fred L. Hall. 1968. “Abstract Representation of Goals: A Method for Making Decisions in Complex Problems.” MASSACHUSETTS INST OF TECH CAMBRIDGE DEPT OF CIVIL ENGINEERING.

Matsumoto, Shou. 2019. “PR-OWL Decision: A Framework for Decision Making with Probabilistic Ontologies.” ProQuest Dissertations Publishing.

Matsumoto, Shou, Rommel N. Carvalho, Paulo CG Costa, Kathryn B. Laskey, Laécio L. Dos Santos, and Marcelo Ladeira. 2011. “There’s No More Need to Be a Night OWL: On the PR-OWL for a MEBN Tool Before Nightfall.” In Introduction to the Semantic Web: Concepts, Technologies and Applications, edited by Gabriel P. C. Fung. iconceptpress.

Matsumoto, Shou, Rommel Novaes Carvalho, Marcelo Ladeira, Paulo Cesar G. da Costa, Laecio Lima Santos, Danilo Silva, Michael Onishi, Emerson Machado, and Ke Cai. 2011. “UnBBayes: A Java Framework for Probabilistic Models in AI.” Java in Academia and Research, 34.

McGuinness, Deborah L., and Frank Van Harmelen, eds. 2004. “OWL Web Ontology Language Overview.” W3C Recommendation 10 (10): 2004.

Medina-Oliva, Gabriela, Philippe Weber, and Benoît Iung. 2013. “PRM-Based Patterns for Knowledge Formalisation of Industrial Systems to Support Maintenance Strategies Assessment.” Reliability Engineering & System Safety 116: 38–56.

Milch, Brian, Bhaskara Marthi, Stuart Russell, David Sontag, Daniel L. Ong, and Andrey Kolobov. 2007. “BLOG: Probabilistic Models with Unknown Objects.” In Statistical Relational Learning, edited by Lisa Getoor and Ben Taskar, 373.

Milch, Brian, and Stuart Russell. 2010. “Extending Bayesian Networks to the Open-Universe Case.” In Heuristics, Probability and Causality: A Tribute to Judea Pearl., edited by Rina Dechter, Hector Geffner, and Joseph Halpern. College Publications.

Miller, Danny, and Jamal Shamsie. 1999. “Strategic Responses to Three Kinds of Uncertainty: Product Line Simplicity at the Hollywood Film Studios.” Journal of Management 25 (1): 97–116.

Mintzberg, Henry, Duru Raisinghani, and André Théorêt. 1976. “The Structure of ‘Unstructured’ Decision Processes.” Administrative Science Quarterly 21 (2): 246–275.

Mohan, Kannan, Radhika Jain, and Balasubramaniam Ramesh. 2007. “Knowledge Networking to Support Medical New Product Development.” Decision Support Systems 43 (4): 1255–1273.

Montibeller, Gilberto, Haidee Gummer, and Daniele Tumidei. 2006. “Combining Scenario Planning and Multi-Criteria Decision Analysis in Practice.” Journal of Multi-Criteria Decision Analysis 14 (1–3): 5–20.

Neil, Martin, Norman Fenton, and Lars Nielson. 2000. “Building Large-Scale Bayesian Networks.” The Knowledge Engineering Review 15 (3): 257–284.

Neumann, John von, and Oskar Morgenstern. 1947. Theory of Games and Economic Behavior. 2nd ed. Princeton, NJ, USA: Princeton University Press.

Nickols, Fred. 2012. “Three Problem Solving Approaches.” 2012. http://www.nickols.us/three%20problem%20solving%20approaches.jpg.

Nielsen, Thomas D., and Finn V. Jensen. 2004. “Advances in Decision Graphs.” In Advances in Bayesian Networks, 137–159. Springer.

Nutt, Paul C. 1993. “The Formulation Processes and Tactics Used in Organizational Decision Making.” Organization Science 4 (2): 226–51.

O’Brien, Frances A. 2004. “Scenario Planning—-Lessons for Practice from Teaching and Learning.” European Journal of Operational Research 152 (3): 709–722.

OED. 2017. “Definition of Context.” Oxford Dictionaries | English. 2017. https://en.oxforddictionaries.com/definition/context.

Olesen, Kristian G., Uffe Kjaerulff, Frank Jensen, Finn V. Jensen, Bjoern Falck, Steen Andreassen, and Stig K. Andersen. 1989. “A Munin Network for the Median

Nerve-a Case Study on Loops." Applied Artificial Intelligence an International Journal 3 (2–3): 385–403.

Ozturk, Pinar. 1999. "Towards a Knowledge-Level Model of Context and Context Use in Diagnostic Problems." Applied Intelligence 10 (2–3): 123–37.

Öztürk, Pinar, and Agnar Aamodt. 1997. "Towards a Model of Context for Case-Based Diagnostic Problem Solving." In Context-97; Proceedings of the Interdisciplinary Conference on Modeling and Using Context, 198–208. Citeseer.

Park, Cheol Young, Kathryn Blackmond Laskey, Paulo CG Costa, and Shou Matsumoto. 2014. "Predictive Situation Awareness Reference Model Using Multi-Entity Bayesian Networks." In Information Fusion (FUSION), 2014 17th International Conference On, 1–8. IEEE.

Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph, eds. 2012. "OWL 2 Web Ontology Language Primer (Second Edition)." W3C. https://www.w3.org/TR/owl2-primer/#References.

Pasula, Hanna, Bhaskara Marthi, Brian Milch, Stuart Russell, and Ilya Shpitser. 2002. "Identity Uncertainty and Citation Matching." In Advances in Neural Information Processing Systems, 1401–1408.

Pearl, Judea. 1988. Probabilistic Reasoning in Intelligence Systems. San Mateo, CA: Morgan Kaufman.

———. 2005. "Influence Diagrams—Historical and Personal Perspectives." Decision Analysis 2 (4): 232–34.

Peterson, Martin. 2009. An Introduction to Decision Theory. Cambridge Introductions to Philosophy. New York: Cambridge University Press.

Pfeffer, Avrom J. 1999. "Probabilistic Reasoning for Complex Systems." Ph.D Thesis, Stanford CA USA: Stanford University.

Phillips, Lawrence D. 1984. "A Theory of Requisite Decision Models." Acta Psychologica 56 (1): 29–48.

Poole, David. 1997. "The Independent Choice Logic for Modelling Multiple Agents under Uncertainty." Artificial Intelligence 94 (1): 7–56.

———. 2008. "The Independent Choice Logic and Beyond." In Probabilistic Inductive Logic Programming, 222–243. Springer.

———. 2011. “Logic, Probability and Computation: Foundations and Issues of Statistical Relational AI.” In International Conference on Logic Programming and Nonmonotonic Reasoning, 1–9. Springer.

Power, Daniel J., and Ramesh Sharda. 2007. “Model-Driven Decision Support Systems: Concepts and Research Directions.” Decision Support Systems 43 (3): 1044–1061.

Pratt, John W., Howard Raiffa, and Robert Schlaifer. 1964. “The Foundations of Decision Under Uncertainty: An Elementary Exposition.” Journal of the American Statistical Association 59 (306): 353.

Raedt, Luc De, Kristian Kersting, Sriraam Natarajan, and David Poole. 2016. Statistical Relational Artificial Intelligence: Logic, Probability, and Computation. Vol. 10. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool.

Raiffa, Howard. 1997. Decision analysis: Introductory Lectures on Choice Under Uncertainty. College Custom Series. New York: McGraw Hill.

Ramsey, F.P. 1931. “Truth and Probability.” In The Foundations of Mathematics and Other Logical Essays, by F.P. Ramsey, edited by R.B. Braithwaite, 156–98. London /New York: Kegan, Paul, Trench, Trubner & Co./ Harcourt, Brace and Company. h

Regan, Helen M., Yakov Ben-Haim, Bill Langford, William G. Wilson, Per Lundberg, Sandy J. Andelman, and Mark A. Burgman. 2005. “Robust Decision-Making under Severe Uncertainty for Conservation Management.” Ecological Applications 15 (4): 1471–1477.

Reich, Yoram, and Adi Kapeliuk. 2005. “A Framework for Organizing the Space of Decision Problems with Application to Solving Subjective, Context-Dependent Problems.” Decision Support Systems 41 (1): 1–19.

Renooij, Silja, and Linda C. van der Gaag. 2002. “From Qualitative to Quantitative Probabilistic Networks.” In Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence, 422–429. Morgan Kaufmann Publishers Inc.

Richardson, Matthew, and Pedro Domingos. 2006. “Markov Logic Networks.” Machine Learning 62 (1): 107–136.

Rittel, Horst, and Melvin Webber. 1973. “Dilemmas in a General Theory of Planning.” Policy Sciences 4 (2): 155–169.

Rowe, Alan J. 1962. “Management Decision Making and the Computer.” Management International 2 (2): 9–22.

Rowe, William D. 1994. “Understanding Uncertainty.” Risk Analysis 14 (5): 743–750.

Roy, Bernard. 2005. “Paradigms and Challenges.” In Multiple Criteria Decision Analysis - State of the Art Survey, edited by J. Figueira, S. Greco, and M. Ehrgott, 3–24. Springer.

Russell, Stuart. 2015. “Unifying Logic and Probability.” Communications of the ACM 58 (7): 88–97.

Russell, Stuart J, and Peter Norvig. 2018. Artificial Intelligence: A Modern Approach. Noida, India: Pearson India Education Services Pvt. Ltd.

Ryan, Alex. 2007. “Position Paper: A Complex Systems Approach to Operational Planning.” http://cs.calstatela.edu/wiki/images/7/79/Position_Paper_on_Fragmented_Planning.pdf.

Saaty, Thomas L., and Hsu-Shih Shih. 2009. “Structures in Decision Making: On the Subjective Geometry of Hierarchies and Networks.” European Journal of Operational Research 199 (3): 867–872.

Sanjib, Kumar M. n.d. Research Methodology Self-Learning Manual. Accessed July 2, 2015. https://www.scribd.com/doc/91342515/Research-Methodology-Self-Learning-Manual.

Santos, Eunice E., Eugene Santos, Long Pan, and John T. Wilkinson. 2008. “Culturally Infused Social Network Analysis.” In Proceedings of the 2008 International Conference on Artificial Intelligence, 449–55. Las Vegas, NV, USA.

Savage, Leonard J. 1954. The Foundations of Statistics. New York, Dover Publications.

Scherpereel, Christopher. 2006. “Decision Orders: A Decision Taxonomy.” Management Decision 44 (1): 123–36.

Scheubrein, Ralph, and Stanley Zionts. 2006. “A Problem Structuring Front End for a Multiple Criteria Decision Support System.” Computers & Operations Research 33 (1): 18–31.

Schum, David A. 1994. The Evidential Foundations of Probabilistic Reasoning. Northwestern University Press.

Shachter, Ross D. 1986. “Evaluating Influence Diagrams.” Operations Research 34 (6): 871–882.

Shachter, Ross D., and Debarun Bhattacharjya. 2010. “Solving Influence Diagrams: Exact Algorithms.” In Wiley Encyclopedia of Operations Research and Management Science.

Sharfman, Mark P., and James W. Dean. 1997. “The Effects of Context on Strategic Decision Making Processes and Outcomes.” In Strategic Decisions, 179–203. Springer.

Shenoy, P. P. 1992. “Valuation-Based Systems for Bayesian Decision Analysis.” Operations Research 40: 463–84.

———. 1998. “Game Trees for Decision Analysis.” Theory and Decision 44 (2): 149–171.

Shepherd, Neil Gareth, and John Maynard Rudd. 2014. “The Influence of Context on the Strategic Decision-Making Process: A Review of the Literature.” International Journal of Management Reviews 16 (3): 340–364.

Shpitser, Ilya. 2010. “Disease Models, Part I: Graphical Models.” In Medical Imaging Informatics, 335–369. Springer.

Simon, Herbert A. 1967. “The Logic of Heuristic Decision Making.” In The Logic of Decision and Action, edited by Nicholas Rescher, 1–34. Pittsburgh, PA: University of Pittsburgh Press.

Snidaro, Lauro, Jesús García, and James Llinas. 2015. “Context-Based Information Fusion: A Survey and Discussion.” Information Fusion 25: 16–31.

Snowden, David J., and Mary E. Boone. 2007. “A Leader’s Framework for Decision Making.” Harvard Business Review 85 (11): 68.

Sorenson, L, and R.V.V Vidal. 1999. “Getting an Overview with SWOT.” DTI Working Paper. Technical University of Denmark, Lyngby Denmark.

Sowa, John F. 2000. Knowledge Representation: Logical, Philosophical, and Computational Foundations. Pacific Grove: Brooks/Cole.

Stewart, Theodor J., Simon French, and Jesus Rios. 2013. “Integrating Multicriteria Decision Analysis and Scenario Planning—Review and Extension.” Omega 41 (4): 679–688.

Stirling, A. C., and I. Scoones. 2009. “From Risk Assessment to Knowledge Mapping: Science, Precaution, and Participation in Disease Ecology.”

Teoh, Ping Chow, and Keith Case. 2005. “An Evaluation of Failure Modes and Effects Analysis Generation Method for Conceptual Design.” International Journal of Computer Integrated Manufacturing 18 (4): 279–293.

Terek, Milan. 2005. “Decision Trees and Influence Diagrams in Decision Analysis.” Journal of the Applied Mathematics, Statistics and Informatics (JAMSI) 1 (1): 121–35.

Tetlow, Phil, Jeff Z. Pan, Daniel Oberle, Evan Wallace, Michael Uschold, and Elisa Kendall. 2006. “Ontology Driven Architectures and Potential Uses of the Semantic Web in Systems and Software Engineering.” Working Draft. W3C.

Theissen, Manfred, and Wolfgang Marquardt. 2008. “Decision Models.” In Collaborative and Distributed Chemical Engineering. From Understanding to Substantial Design Process Support, 153–168. Springer.

Thomsen, Steen. 2004. “Corporate Values and Corporate Governance.” Corporate Governance: The International Journal of Business in Society 4 (4): 29–46.

Thrall, Robert M. 1984. “A Taxonomy for Decision Models.” Annals of Operations Research 2 (1): 23–27.

Vári, Anna, and János Vecsenyi. 1984. “Designing Decision Support Methods in Organizations.” Acta Psychologica 56 (1–3): 141–51.

Vlek, Charlotte S., Henry Prakken, Silja Renooij, and Bart Verheij. 2014. “Building Bayesian Networks for Legal Evidence with Narratives: A Case Study Evaluation.” Artificial Intelligence and Law 22 (4): 375–421.

Von Winterfeldt, Detlof, and Ward Edwards. 1986. Decision Analysis and Behavioral Research. Cambridge Cambridgeshire ; New York: Cambridge University Press.

Walther, Larry M. 2018. Managerial Accounting.

Watson, Stephen R., and Dennis M. Buede. 1987. Decision Synthesis: The Principles and Practice of Decision Analysis. Cambridge University Press.

Wellman, Michael P. 1990. “Fundamental Concepts of Qualitative Probabilistic Networks.” Artificial Intelligence 44 (3): 257–303.

White, D. J. 1969. Decision Theory. London: George Allen and Unwin Ltd.

Winterfeldt, Detlof von. 1980. “Structuring Decision Problems for Decision Analysis.” Acta Psychologica 45 (1): 71–93.

Winterfeldt, Detlof von, and Ward Edwards. 2007. “Defining a Decision Analytic Structure.” In Advances in Decision Analysis: From Foundations to Applications, edited by Ward Edwards, R.F Miles, and Detlof Von Winterfeldt. New York: Cambridge University Press.

Winterfeldt, Detlof von, and Barbara Fasolo. 2009. “Structuring Decision Problems: A Case Study and Reflections for Practitioners.” European Journal of Operational Research 199 (3): 857–866.

Woudenberg, Steven PD, Linda C. van der Gaag, and Carin MA Rademaker. 2015. “An Intercausal Cancellation Model for Bayesian-Network Engineering.” International Journal of Approximate Reasoning 63: 32–47.

Yet, Barbaros, Anthony Constantinou, Norman Fenton, Martin Neil, Eike Luedeling, and Keith Shepherd. 2016. “A Bayesian Network Framework for Project Cost, Benefit and Risk Analysis with an Agricultural Development Case Study.” Expert Systems with Applications 60: 141–155.

Zsambok, Caroline. 1996. “Naturalistic Decision Making: Where Are We Now?” In Naturalistic Decision Making, edited by Caroline Zsambok and Gary Klein, 3–16. New York: Lawrence Earlbaum Associates.

BIOGRAPHY

Mark A. Locher received his Bachelor of Science in Operations Research and in Aviation Science from the United States Air Force Academy in 1979. He completed a Master of Business Administration from Saint Mary's University (San Antonio, Texas) in 1982, and a Master of Science in System Engineering from George Mason University in 2012. He served for over twenty years with the United States Air Force, primarily in the systems acquisition and engineering career fields. He then worked as a senior staff support to the Office of the Secretary of Defense, Headquarters United States Air Force, and several defense agencies in the Washington DC area.