Automated Reasoning with Uncertainties

Flávio S. Corrêa da Silva

Ph.D.
University of Edinburgh
1992

Abstract

In this work we assume that uncertainty is a multifaceted concept which admits several different measures, and present a system for automated reasoning with multiple representations of uncertainty. Our focus is on problems which present more than one of these facets and therefore in which a multivalued representation of uncertainty and the study of its possibility of computational realisation are important for designing and implementing knowledge-based systems.

We present a case study on developing a computational language for reasoning with uncertainty, starting with a semantically sound and computationally tractable language and gradually extending it with specialised syntactic constructs to represent measures of uncertainty, preserving its unambiguous semantic characterisation and computability properties. Our initial language is the language of normal clauses with SLDNF as the inference rule, and we select three facets of uncertainty, which are not exhaustive but cover many situations found in practical problems: *vagueness*, *statistics* and *degrees of belief*. To each of these facets we associate a specific measure: *fuzzy measures* to vagueness, *probabilities on the domain* to statistics and *probabilities on possible worlds* to degrees of belief.

The resulting language is *semantically sound* and *computationally tractable*, and admits relatively efficient implementations employing $\alpha - \beta$ *pruning* and *caching*.

Acknowledgements

My supervisors were Paul Chung (during the first two years), Jane Hesketh (during the final year) and Dave Robertson (all the way through it). I cannot overstate how important, useful and pleasant their guidance and friendship was to develop this project.

Didier Dubois and Henri Prade kindly accepted me as a visiting student at the IRIT - Université Paul Sabatier (Toulouse), during November 1990. This research owes much to the technical advice and stimulating discussions with both Dubois and Prade, and with Jérôme Lang and Sandra Sandri.

Many people have generously offered me important advice and suggestions along the development of this work. I would like to thank especially Fahiem Bacchus, John Fox and V.S. Subrahmanian. Others have been patient enough to read early drafts of this thesis and make many insightful comments and suggestions. Those were Álvaro Fernandes, Alex Gammerman and Tom Kane.

Thanks to my examiners J. Paris and A. Bundy for being so severe as they were with their corrections and suggestions for change, consequently forcing me to change significantly the final quality of this work.

It was once said that the best of all presents is the presence of our friends. It is a pleasure to have so much to thank to so many friends. In special, I would like to thank Brian Ross, Álvaro, Regina; Dave, Margot, Jane, Paul, Anissa; Alan Black, Dave Moffat, Ian Lewin, Suresh, Am, Joke, Emilio, Blanca, Franziska, Rob Scott, Keiichi, Carla, Nelson, Sheila Rock and Lais.

Thanks to my family.

The financial support for the development of this PhD project came from CNPq - Conselho Nacional de Desenvolvimento Científico e Tecnológico (Brazil) - grant nr. 203004/89.2.

Declaration

I declare that I composed this thesis myself in its entirety and that it describes my own research.

Flávio Soares Corrêa da Silva
Edinburgh
March 22, 1993

Contents

List of Figures

List of Tables

"One should be always a little improbable"

O. Wilde

1

Introduction

When reasoning we frequently use uncertain information, i.e. information that is incomplete, vague, only partially reliable or based on statistical associations. Hence, when building automated reasoning systems we frequently need tools and mechanisms to represent uncertainty.

In this work we present a *system for automated reasoning with multiple representations of uncertainty:* uncertainty is a multifaceted concept, and because of this there are several techniques for measuring it. Our focus in this work is on problems which present more than one of these facets, a situation in which it is important to differentiate kinds of uncertainty. Our interest is on a multivalued representation of uncertainty and the study of its possibility of computational realisation, therefore, despite understanding that in many cases different measures of uncertainty can be amalgamated into a single formalism, we are not interested in doing so.

We present a case study on developing a computational language for reasoning with uncertainty, starting with a semantically sound and computationally tractable language and gradually extending it with specialised syntactic constructs to represent measures of uncertainty, preserving its unambiguous semantic characterisation and computability properties. Our initial language is the language of *normal clauses with SLDNF as the inference rule* (i.e. the language of pure PROLOG[1] [Kun89]), which:

[1] this language is reviewed in chapter 2

1. is expressive enough to represent a significant portion of first-order logic,
2. admits computationally tractable implementations, and
3. has a well defined formal semantics.

We select three specific facets of uncertainty for our study, which are not exhaustive but cover many situations found in practical problems. These facets are:

1. *vagueness*, which describes the extent to which a non-categorical statement is true - a vague predicate is one whose truth-value admits intermediate values between *true* and *false* (e.g. the predicate "fat" qualifying the weight of a person);
2. *statistics*, which describes the likelihood of selecting an element or class of elements belonging to the domain of discourse;
3. *degrees of belief*, which describe the belief apportioned to statements represented by sentences in our language.

To each of these facets we associate a specific measure. We associate *fuzzy measures* to vagueness [DP88, Rus90b, Zad88], *probabilities on the domain* to statistics [Bac90c, Hal90] and *probabilities on possible worlds* to degrees of belief [Che88, CdSB91, KJ87, Nil86, Pea88, Sha76].

The expected contribution of this work is the empirical confirmation that automated reasoning with multiple representations of uncertainty can be done in a practical sense. Nonetheless, the language which was constructed to provide this confirmation is expected to be useful as a tool to implement knowledge-based systems. We also expect this work to be an instance of a general methodology to construct pluralistic uncertainty management systems, by adding on formally specified uncertainty measures to an existing semantically sound knowledge representation mechanism in an "as modular as possible" way.

The resulting language in our experiment is *semantically sound* and *computable*. In order to make it computationally efficient, we explore the applicability of some standard optimisation techniques, such as $\alpha - \beta$ *pruning* and *caching*, to the inference mechanisms employed in the language.

1.1 Relation with Similar Works in the Area

Our approach can be compared with some other eclectic approaches for managing uncertainty as follows:

In [Cla90] we find the distinction between symbolic and numerical kinds of uncertainty, where numerical kinds are those based on real numbers and symbolic kinds are those based on some discrete collection of symbols representing states of uncertainty. This classification contrasts with our view in the sense that it regards as of secondary importance the following aspects:

- whether uncertainty measures have a clearly specified semantics;
- whether the measures are of fuzziness or of probability.

Also, it does not account for measures attached to objects in the domain.

The conclusion in that article supports our view that representations of uncertainty must be powerful enough to treat different measures of uncertainty simultaneously:

> '(...) The problem (...) is that many domains of interest to artificial intelligence are composed of a mixture of quantitative and qualitative relations. So no uncertainty management technique may be unequivocally appropriate. This raises the need to intelligently combine different uncertainty management techniques and suggests that an important area of research is the use of both symbolic and quantitative representations of uncertainty in the same application.' [Cla90]

The classification in [Saf87] identifies the following kinds of uncertainty, identified by their measures:

- Bayesian;
- Dempster-Shafer;
- Explicit (i.e. symbolic);
- Others, including multi-valued and fuzzy logics.

The emphasis in that work is on selecting the appropriate algebra of uncertainty, as opposed to our emphasis on treating the combination of different algebras. That paper generated a collection of responses [Cla88], in which many technical aspects were further explored and clarified, including the differentiation between measures of fuzziness and of probability.

Languages to represent and reason with probabilities on the domain are presented in [Bac90c, Hal90]. These works also analyse the problem of representing and reasoning with probabilities on possible worlds and propose hybrid languages to deal with probabilities on the domain and on possible worlds. The main difference between them is the set of constraints imposed on the languages to achieve computability[2]. The difference between probabilities on the domain and on possible worlds is better understood with some of the examples found in [Hal90]: an example of a statement declaring probabilities on the domain is the statement "the probability that a randomly chosen bird will fly is greater than 0.9"; an example of a statement declaring probabilities on possible worlds is the statement "the probability that Tweety (a particular bird) flies is greater than 0.9"; and an example of a statement declaring probabilities on both the domain and possible worlds is the statement "the probability that Tweety flies is greater than the probability that a randomly chosen bird will fly". The main difference between the results presented in those works

[2] these constraints are reviewed in chapter 4

and ours is that those results are mainly theoretical and pay little attention to implementation issues, whereas ours is directed towards building a programmable language.

None of these two works refers to the problem of dealing with vague (i.e. fuzzy) predicates. This problem is addressed in [DP88], where possibilistic measures are used instead of probabilities on possible worlds and possibilistic and fuzzy measures are treated within a single framework. The analysis of probabilities of fuzzy events is developed in [Kle82, Sme82], where probabilities are treated as expected truth-values. This analysis is developed outwith the context of applying the measures to logical statements, but it constitutes the basis upon which we construct our language.

1.2 Plan of the Work

In chapter 2 we review the main concepts of fuzzy set theory, probability theory and logic programming which are used throughout the rest of the work. The material presented in that chapter is not original and is presented in order to set the notation and to make this work self-contained.

In chapter 3 we introduce a logic programming language that can treat fuzzy predicates, and that treats negation by finite failure.

In chapter 4 we extend this language to deal with probabilities on the domain. The language implements a significant subset of the logic L_p [Bac88, Bac90a, Bac90c], extended with fuzzy predicates. The logic L_p was known to have computable subsets, but we are not aware of any previous implementations of it.

In chapter 5 we introduce the concepts of possible worlds and degrees of belief to the language. These concepts are introduced in a way that brings close relations between the formalism presented here and well-known formalisms like Incidence Calculus [Bun85, CdSB91], Probabilistic Logic [Nil86] and the Dempster-Shafer Theory of Evidence [Sha76, FH89b].

In chapter 6 we explore some implementation issues.

Finally, chapter 7 summarises and concludes this work.

2

Preliminaries

In this chapter we review the concepts of fuzzy and probability measures and of logic programming that we are going to use in the rest of the work.

First we introduce the concept of *fuzzy sets and relations*, to be used later in the interpretation of fuzzy sentences, then we review the basic concepts of *probability theory* and its extensions to fuzzy events. Finally, we review those concepts of logic programming that we need to define our own language, which include models for interpretation and execution of logic programs.

Throughout this and the following chapters we include some clarifying examples to illustrate specific points of our work. These examples are not essential to the understanding of the theory, but they are expected to help making the reading of this work more amenable. Additionally, anecdotal examples are included to suggest how this language could be used in "practical" situations.

2.1 Probability and Fuzzy Measures

2.1.1 Fuzzy Measures

Given a countable[1] set D, *fuzzy set theory* was developed to treat vaguely defined subsets by allowing degrees of membership. A fuzzy membership function measures the degree to which an element belongs to a subset or, alternatively, the degree of similarity between the class (subset) to which an element belongs and a reference class. Formally, a fuzzy subset F of a referential set D is defined by an arbitrary mapping $\mu_F : D \to [0, 1]$, in which, for an element $d \in D$, $\mu_F(d) = 1$ corresponds to the intuitive notion that $d \in F$ and $\mu_F(d) = 0$ to the notion that $d \notin F$ [DP89].

Example 2.1 *The set of middle-aged people can be defined by the fuzzy membership function* $\mu_{ma}(x)$, *where* x *ranges over a representation of the age of individuals (in years).* $\mu_{ma}(x)$ *is presented in Figure 2.1.*

Set-theoretic operations can be extended to fuzzy sets. In [DP89] the requirements for operations on fuzzy sets to be considered extended set operations are presented as follows: let f and g be conventional unary and binary operations on 2^D - the set of subsets of D - and let $\overline{f}$ and $\overline{g}$ be their extensions on the set $\overline{2^D}$ of fuzzy subsets of D. The extensions should be such that

1. they are *closed*, i.e. the results of operations on sets are also sets ($F_1, F_2 \in \overline{2^D} \Rightarrow \overline{f}F_1 \in \overline{2^D}$, $F_1 \overline{g} F_2 \in \overline{2^D}$), and

2. they are *reducible to the conventional operations*, i.e. the results of the extended operations on conventional sets coincide with the ones of conventional operations ($F_1, F_2 \in 2^D \Rightarrow \overline{f}F_1 \equiv fF_1$, $F_1 \overline{g} F_2 \equiv F_1 g F_2$).

[1] actually, fuzzy set theory can be defined on uncountable sets. We restrict our attention to countable sets, here, as this is the kind usually needed to characterise domains of discourse for knowledge-based systems

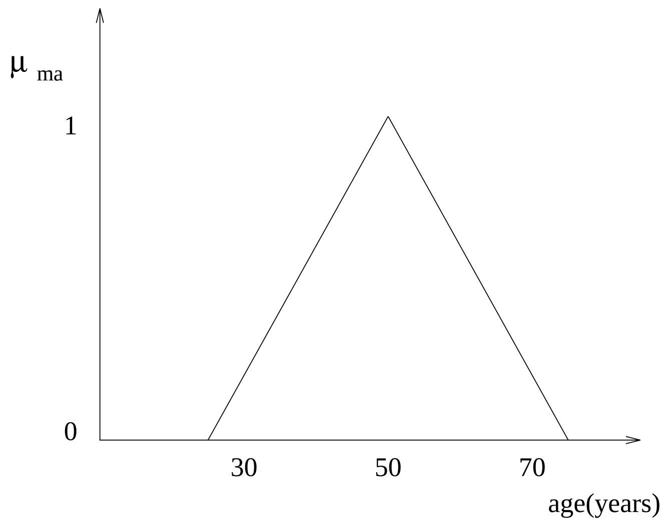

Figure 2.1: Fuzzy Membership Function for the Class "Middle-Aged"

Triangular norms and conorms have been proved to obey these requirements as extensions to the operations of intersection and union, respectively [Kle82]. A *triangular norm* is any function $T : [0,1] \times [0,1] \rightarrow [0,1]$ such that:

- $T(x,1) = x$ (boundary condition);
- $x_1 \leq x_2, y_1 \leq y_2 \Rightarrow T(x_1,y_1) \leq T(x_2,y_2)$ (monotonicity);
- $T(x,y) = T(y,x)$ (commutativity);
- $T(T(x,y),z) = T(x,T(y,z))$ (associativity).

The *conorm of a triangular norm* is the function $S : [0,1] \times [0,1] \rightarrow [0,1]$ defined by:

$$S(x,y) = 1 - T(1-x, 1-y).$$

Furthermore, following [DP89], any function $C : [0,1] \rightarrow [0,1]$ such that $C(\mu_F(d)) = 1 - \mu_F(d)$ obeys the requirements as extension of complementation.

However, not all algebraic properties of set operations are necessarily shared by triangular norms and conorms. In fact, as presented in [Kle82], the only norms and conorms that are also *distributive* and *idempotent*[2] are $T = min$ and $S = max$ - known as Zadeh's triangular norms and conorms. Henceforth, in order to keep fuzzy set operations as close as possible to conventional set operations, we adopt the following functions as our extended set operations of intersection, union and complementation[3]:

---

[2] i.e. that obey the following rules:

1. $\left\{ \begin{array}{l} S(x,T(y,z)) = T(S(x,y),S(x,z)) \\ T(x,S(y,z)) = S(T(x,y),T(x,z)) \end{array} \right\}$ (distributivity);
2. $T(x,x) = x$ and $S(x,x) = x$ (idempotency).

Conventional set union and intersection are both distributive and idempotent

[3] these are the most commonly used definitions of fuzzy set operations

- intersection: $\mu_{A\cap B}(x) = min\{\mu_A(x), \mu_B(x)\}$;
- union: $\mu_{A\cup B}(x) = max\{\mu_A(x), \mu_B(x)\}$;
- complementation: $\mu_{\neg A}(x) = 1 - \mu_A(x)$.

Example 2.2 *Given the class of middle-aged people - represented by $\mu_{ma}(x)$ as before (Figure 2.1) - and the class of fat people - represented by $\mu_f(y)$ with y ranging on a representation of the weight of individuals (in kg) (Figure 2.2) - and given that both measures refer to the same set of individuals, we can derive the classes of NON-fat, middle-aged AND fat, and middle-aged OR fat people. These are presented pictorially in Figures 2.3, 2.4 and 2.5, respectively. Observe that binary set operations correspond to evaluations in the Cartesian product space of the corresponding membership functions.*

2.1.2 Probability Measures

Given a finite set D - to be identified later with the domain of discourse of our language - we define a particular collection of systems of subsets of D, which we call the collection of *algebras* of D, and measures on these algebras which we denominate *probability measures.* Since the interpretations of predicates of arity greater than 1 belong to cartesian products of members of a partition of D, we extend the concept of algebras and probability measures to partitions and cartesian products of members of partitions of D.

Given a finite set D, an *algebra* χ_D on D is a set of subsets of D such that:

- $D \in \chi_D$;
- $A \in \chi_D \Rightarrow \neg A \in \chi_D$;
- $A, B \in \chi_D \Rightarrow A \cup B \in \chi_D$.

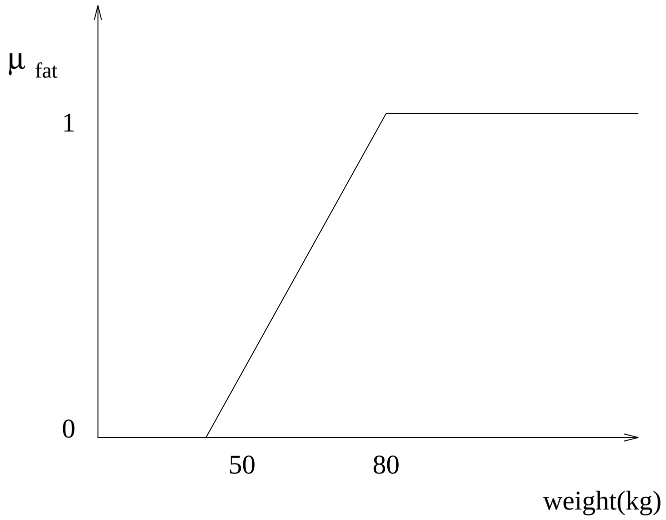

Figure 2.2: Fuzzy Membership Function for the Class “Fat”

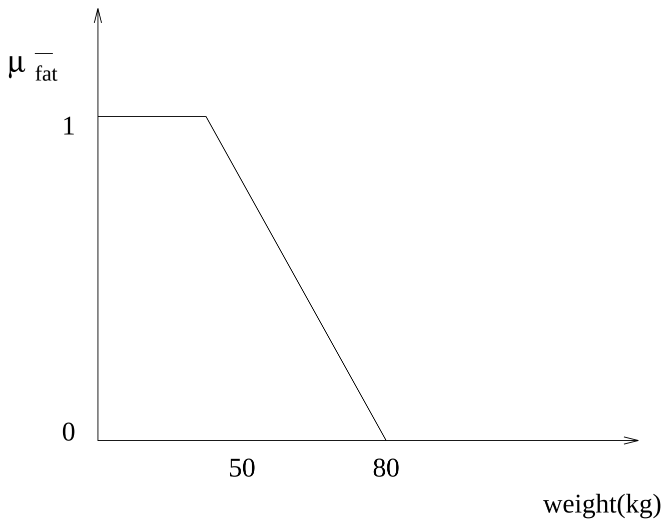

Figure 2.3: Fuzzy Membership Function for the Class “NON-Fat”

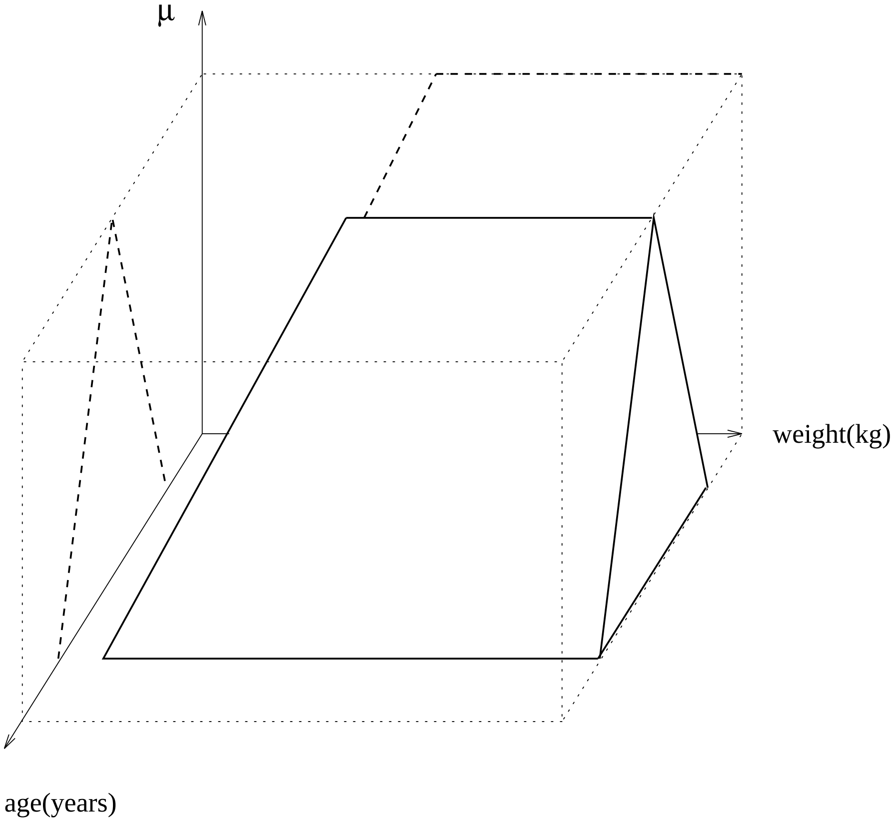

Figure 2.4: “Middle-Aged” AND “Fat”

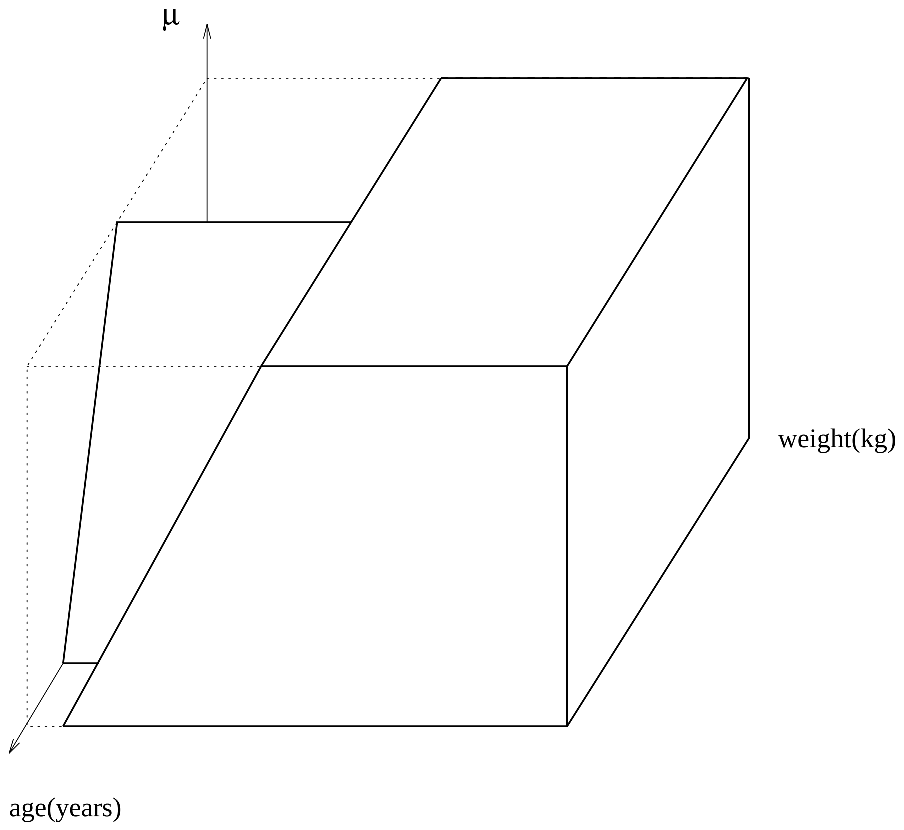

Figure 2.5: “Middle-Aged” OR “Fat”

Any subset of D is called an *event* on D. Events belonging to χ_D are called *measurable events*.

The *basis* χ'_D of an algebra χ_D is the subset of χ_D such that:

- $\{\} \notin \chi'_D$;
- $A, B \in \chi'_D, A \neq B \Rightarrow A \cap B = \{\}$;
- $K \in \chi_D \Rightarrow \exists\, A_1, ..., A_n \in \chi'_D : K = \bigcup_1^n A_i$.

A *probability measure* on χ_D is a function $\mathcal{P} : \chi_D \rightarrow [0, 1]$ such that:

- $\mathcal{P}(D) = 1$ (total probability);
- $A \cap B = \{\} \Rightarrow \mathcal{P}(A \cup B) = \mathcal{P}(A) + \mathcal{P}(B)$ (finite additivity).

As presented in [FH89b], once $\mathcal{P}$ is defined for χ'_D it can be extended to the whole algebra by finite additivity. This is useful as we can specify a probability measure by defining its value only for the elements of χ'_D.

Given two measurable events $A, B \in \chi_D$, the *conditional probability* $\mathcal{P}(A|B)$[4] is defined as[5]:

$$\mathcal{P}(A|B) = \begin{cases} \dfrac{\mathcal{P}(A \cap B)}{\mathcal{P}(B)}, & \mathcal{P}(B) \neq 0 \\ 0, & \mathcal{P}(B) = 0 \end{cases}$$

Two measurable events A, B are called *independent* iff:

$$\mathcal{P}(A|B) = \mathcal{P}(A)$$

which, as a corollary, gives that:

[4] to be read as "the probability of the event A given the event B"

[5] notice that, as a corollary of the definition of an algebra, $A, B \in \chi_D \Rightarrow A \cap B \in \chi_D$ [Dud89]

$$\mathcal{P}(A \cap B) = \mathcal{P}(A) \times \mathcal{P}(B)$$

if A and B are independent.

The set D can be partitioned into m subsets $D_1, ..., D_m$ such that

- $D_i \cap D_j = \{\}, i, j = 1, ..., m$.
- $D = \bigcup_1^m D_i$.

We can have independent algebras[6] χ_{Di} and probability measures $\mathcal{P}_i$ for each set D_i. If we assume that all events in each χ_{Di} are *pairwise independent*, we can extend measures to cartesian products of the sets D_i of a partition of D: the cartesian product of a collection of bases of algebras of elements $D_1, ..., D_m$ of a partition of D is the basis $\vec{\chi}'$ of an algebra of the cartesian product of the sets $D_1, ..., D_m$, and the measure $\vec{\mathcal{P}}$ on the corresponding algebra $\vec{\chi}$ is defined as:

- $\vec{\mathcal{P}} : \vec{\chi} \rightarrow [0, 1]$.
- $\vec{\mathcal{P}}(\vec{A}) = \prod_1^m \mathcal{P}_i(A_i)$.

  where

  - $\vec{A} = [A_1, ..., A_m]$,
  - $A_i \in \chi_{Di}$,
  - $\mathcal{P}_i$ is the probability measure defined on χ_{Di}.

Probability measures can be extended to *non-measurable events*, i.e. sets $A_j \in 2^D \setminus \chi_D$. Given D, χ_D and $\mathcal{P}$, we define the *inner and outer extensions to* $\mathcal{P}$ ($\mathcal{P}_*$ and $\mathcal{P}^*$, respectively) as [Dud89, GS82, FH89b]:

[6]two algebras χ_1 and χ_2 are *independent* iff each event $X_i \in \chi_1$ is independent of every event $Y_j \in \chi_2$ and vice-versa

- $\mathcal{P}_*, \mathcal{P}^* : 2^D \rightarrow [0,1]$

- $\mathcal{P}_*(A) = sup\{\mathcal{P}(X) : X \subseteq A, X \in \chi_D\} = \mathcal{P}(\bigcup X : X \subseteq A, X \in \chi'_D)$

- $\mathcal{P}^*(A) = inf\{\mathcal{P}(X) : A \subseteq X, X \in \chi_D\} = \mathcal{P}(\bigcup X : X \cap A \neq \{\}, X \in \chi'_D)$

Inner and outer measures can be extended to cartesian products of a partition of D. Given a collection $D_1, ..., D_m$ of elements of a partition of D, and given also the algebras χ_{Di} and probability measures $\mathcal{P}_i$ of each $D_i, i = 1, ..., m$, we have:

- $\mathcal{P}_{m*}, \mathcal{P}^*_m : 2^{D_1 \times ... \times D_m} \rightarrow [0,1]$

- $\mathcal{P}_{m*}(A) = sup\{\mathcal{P}_m(X) : X \subseteq A, X \in \vec{\chi}\} = \mathcal{P}_m(\bigcup X : X \subseteq A, X \in \vec{\chi}')$.

- $\mathcal{P}^*_m(A) = inf\{\mathcal{P}_m(X) : A \subseteq X, X \in \vec{\chi}\} = \mathcal{P}_m(\bigcup X : X \cap A \neq \{\}, X \in \vec{\chi}')$.

The measures $\mathcal{P}_{m*}$ and $\mathcal{P}^*_m$ can be regarded as approximations from below and from above to the probabilities of non-measurable events: if we could evaluate the probability $\mathcal{P}_m(A)$, then we would have that $\mathcal{P}_{m*}(A) \leq \mathcal{P}_m(A) \leq \mathcal{P}^*_m(A)$. Indeed, for measurable events we have that $\mathcal{P}_{m*}(A) = \mathcal{P}_m(A) = \mathcal{P}^*_m(A)$.

As presented in [FH89a],the best approximations we have for conditional probabilities of non-measurable events can be given by the following expressions:

- $\mathcal{P}_*(A|B) = \begin{cases} \dfrac{\mathcal{P}_*(A \cap B)}{\mathcal{P}_*(A \cap B) + \mathcal{P}^*(\neg A \cap B)}, & \mathcal{P}_*(A \cap B) + \mathcal{P}^*(\neg A \cap B) \neq 0 \\ 0, & \mathcal{P}_*(A \cap B) + \mathcal{P}^*(\neg A \cap B) = 0 \end{cases}$

- $\mathcal{P}^*(A|B) = \begin{cases} \dfrac{\mathcal{P}^*(A \cap B)}{\mathcal{P}^*(A \cap B) + \mathcal{P}_*(\neg A \cap B)}, & \mathcal{P}^*(A \cap B) + \mathcal{P}_*(\neg A \cap B) \neq 0 \\ 0, & \mathcal{P}^*(A \cap B) + \mathcal{P}_*(\neg A \cap B) = 0 \end{cases}$

For the case of measures on $\vec{\chi}$, these expressions can be stated as:

- $\mathcal{P}_{m*}(A|B) = \begin{cases} \dfrac{\mathcal{P}_{m*}(A \cap B)}{\mathcal{P}_{m*}(A \cap B) + \mathcal{P}^*_m(\neg A \cap B)}, & \mathcal{P}_{m*}(A \cap B) + \mathcal{P}^*_m(\neg A \cap B) \neq 0 \\ 0, & \mathcal{P}_{m*}(A \cap B) + \mathcal{P}^*_m(\neg A \cap B) = 0 \end{cases}$

- $\mathcal{P}_m^*(A|B) = \begin{cases} \dfrac{\mathcal{P}_m^*(A \cap B)}{\mathcal{P}_m^*(A \cap B) + \mathcal{P}_{m*}(\neg A \cap B)}, & \mathcal{P}_m^*(A \cap B) + \mathcal{P}_{m*}(\neg A \cap B) \neq 0 \\ 0, & \mathcal{P}_m^*(A \cap B) + \mathcal{P}_{m*}(\neg A \cap B) = 0 \end{cases}$

2.1.3 Probabilities of Fuzzy Events

A sentence containing vague predicates defines a fuzzy set of elements of the domain of discourse (or of elements of the cartesian product of members of one of the partitions of the domain of discourse). Hence, if we allow fuzzy predicates in our language we must be prepared to specify the probability of fuzzy events.

In [Kle82] the concept of algebra is extended to fuzzy sets and in [Pia88, Sme82, Tur88] the definition of the probability of a fuzzy event is presented, reputed as originally by L. Zadeh [Zad68].

A *fuzzy algebra* on D is defined by analogy with the concept of an algebra. It is a set χ_D^F of fuzzy subsets of D, such that:

- $\mu(a : a \in A) = \text{constant} \Rightarrow A \in \chi_D^F$;
- $A \in \chi_D^F \Rightarrow \neg A \in \chi_D^F$;
- $A, B \in \chi_D^F \Rightarrow A \cup B \in \chi_D^F$

Given an algebra χ_D and a probability measure $\mathcal{P}$ on χ_D, the probability of the fuzzy subset $A \in \chi_D^F$ is defined for every measurable A (i.e. for every $A \in \chi_D$), and is given by the Lebesgue-Stieltjes integral

$$\mathcal{P}^F(A) = \int_D \mu(a : a \in A) d\mathcal{P}$$

From the computational point of view, we can access upper and lower bounds for this integral, related to the extreme values of the membership function in $A \cap A_i, A_i \in \chi'_D$:

$$\sum_{A_i \subseteq A} \mathcal{P}(A_i) \times min\{\mu(a_i) : a_i \in A_i \cap A\}$$
$$\leq \mathcal{P}^F(A) \leq$$
$$\sum_{A_i \cap A \neq \{\}} \mathcal{P}(A_i) \times max\{\mu(a_i) : a_i \in A_i \cap A\}$$

These expressions can be extended to the non-measurable cases and to $2^{D_1 \times \ldots \times D_m}$, where $D_1, \ldots, D_m$ form a partition of D. Given a non-measurable fuzzy event A, we have:

$$\sum_{A_i \subseteq A} \mathcal{P}_m(A_i) \times min\{\mu(a_i) : a_i \in A_i \cap A\}$$
$$\leq \mathcal{P}_m^F(A) \leq$$
$$\sum_{A_i \cap A \neq \{\}} \mathcal{P}_m(A_i) \times max\{\mu(a_i) : a_i \in A_i \cap A\}$$

And for the case of conditional probabilities, we have:

- $\mathcal{P}_{m*}^F(A|B) = \begin{cases} \dfrac{\mathcal{P}_{m*1}}{\mathcal{P}_{m*2} + \mathcal{P}_{m3}^*}, \mathcal{P}_{m*2} + \mathcal{P}_{m3}^* \neq 0 \\ 0, otherwise \end{cases}$
- $\mathcal{P}_m^{F*}(A|B) = \begin{cases} min\{1, \dfrac{\mathcal{P}_{m1}^*}{\mathcal{P}_{m2}^* + \mathcal{P}_{m*3}}\}, \mathcal{P}_{m2}^* + \mathcal{P}_{m*3} \neq 0 \\ 0, otherwise \end{cases}$

  where

  - $\mathcal{P}_{m*1} = \sum_{A_i \subseteq [A \cap B]} \mathcal{P}_m(A_i) \times min\{\mu(a_i) : a_i \in A_i \cap [A \cap B]\}$
  - $\mathcal{P}_{m*2} = \sum_{A_i \subseteq [A \cap B]} \mathcal{P}_m(A_i) \times max\{\mu(a_i) : a_i \in A_i \cap [A \cap B]\}$
  - $\mathcal{P}_{m*3} = \sum_{A_i \subseteq [\neg A \cap B]} \mathcal{P}_m(A_i) \times min\{\mu(a_i) : a_i \in A_i \cap [\neg A \cap B]\}$
  - $\mathcal{P}_{m1}^* = \sum_{A_i \cap [A \cap B] \neq \{\}} \mathcal{P}_m(A_i) \times max\{\mu(a_i) : a_i \in A_i \cap [A \cap B]\}$
  - $\mathcal{P}_{m2}^* = \sum_{A_i \cap [A \cap B] \neq \{\}} \mathcal{P}_m(A_i) \times min\{\mu(a_i) : a_i \in A_i \cap [A \cap B]\}$
  - $\mathcal{P}_{m3}^* = \sum_{A_i \cap [\neg A \cap B] \neq \{\}} \mathcal{P}_m(A_i) \times max\{\mu(a_i) : a_i \in A_i \cap [\neg A \cap B]\}$

Observe that we use $\mathcal{P}_{m*2}$ in the denominator of $\mathcal{P}_{m*}^F(A|B)$ instead of $\mathcal{P}_{m*1}$. The reason for that is that, since $\mathcal{P}_{m*1}$ underevaluates the inner measure $\mathcal{P}_{m*}(A \cap B)$, we cannot guarantee that $\mathcal{P}_{m*1} + \mathcal{P}^*_{m3} \geq \mathcal{P}_{m*}(A \cap B) + \mathcal{P}^*_m(\neg A \cap B)$. Hence we use $\mathcal{P}_{m*2}$, since it is clear that $\mathcal{P}_{m*2} + \mathcal{P}^*_{m3} \geq \mathcal{P}_{m*}(A \cap B) + \mathcal{P}^*_m(\neg A \cap B)$. A similar argument explains the use of $\mathcal{P}^*_{m2}$ instead of $\mathcal{P}^*_{m1}$ in the denominator of $\mathcal{P}^{F*}_m(A|B)$.

2.2 Logic Programming with Negation

The language presented here is defined after [Kun89]. The class of logic programs supported by this language is that of *normal non-cyclical programs* which are *strict with respect to queries* and *allowed* (see definitions below). The symbols of the language are:

- variables $x, y, ...$;
- constants $a, b, ...$;
- n-ary predicates p, q[7], ...;
- the connectives '$\leftarrow$' ("if"), '$\neg$' ("not"), ',' ("and").

A *term* is a variable or a constant, an *atom* is a predicate application on terms, and a *literal* is an atom (positive literal) or the negation of an atom (negative literal). A *normal clause* is an expression $p \leftarrow q_1, ..., q_n$ where p is an atom and $q_1, ..., q_n$ are literals, $n \geq 0$. p is called the *head* of the clause and $q_1, ..., q_n$ is called the *body* of the clause. When $n = 0$ the clause is called a *unit clause*. A *query clause* is an expression $q_1, ..., q_n$ where $n \geq 0$. A *normal program* is a finite non-empty set of normal clauses.

Let Pr_P be the set of predicates in the program P. The *immediate dependency relation* $\sqsupseteq$ is defined as follows:

- given $p, q \in Pr_P, p \sqsupseteq q$ iff there is a clause in P in which p occurs in the head and q occurs in the body.

[7] predicates of different arities always have different names

The *dependency relation* $\geq$ is defined as the least transitive reflexive relation on Pr_P extending $\sqsupseteq$: $p \geq q$ means that p depends hereditarily on q.

Signed dependencies are defined as follows:

- $p \sqsupseteq_{+1} q$ iff there is a clause in P in which p occurs in the head and q occurs in a *positive* literal in the body;
- $p \sqsupseteq_{-1} q$ iff there is a clause in P in which p occurs in the head and q occurs in a *negative* literal in the body;
- $\geq_{+1}$ and $\geq_{-1}$ are the least pair of relations satisfying:
  - $p \geq_{+1} p$,
  - $p \sqsupseteq_i q \wedge q \geq_j r \Rightarrow p \geq_{i \times j} r$.

A program P is called *call-consistent* iff it does not have any p such that $p \geq_{-1} p$. If P also does not have any p such that $p \geq_{+1} p$ then it is called *non-cyclical*. P is called *strict with respect to a query* φ iff there are no $p \in P, q \in \varphi$ such that $q \geq_{-1} p$ and $q \geq_{+1} p$. P is called *allowed* iff every variable occurring in each clause of P occurs in at least one positive literal in the body of the clause.

An *instance* of an expression[8] ξ is the expression ξ' obtained by replacing all occurrences of a variable x in ξ by a term different from x. The operation that generates instances is called *substitution*. Essentially, a substitution is a mapping from variables to terms. A *ground instance* of an expression ξ is any variable-free instance of ξ. Given a program P, *ground*(P) stands for the set of all ground instances of the clauses in P.

A substitution σ of two expressions ξ_1 and ξ_2 is a *unifier* iff $\xi_1\sigma = \xi_2\sigma$. It is a *most general unifier* (mgu) iff for any other unifier π of ξ_1 and ξ_2, $\xi_i\pi$ are instances of $\xi_i\sigma$.

[8] an expression is a term, a literal or a clause

Assuming first-order logic with equality as the underlying language, the *completion* of a program P ($Comp(P)$) is defined by the rules and axioms presented in figure 2.6 [Tur89].

The *semantic model* of a program P is defined in terms of its completion $Comp(P)$. The *domain* of $Comp(P)$ consists of the non-empty set D of constants occurring in P. The interpretation of a predicate $p \in Pr_P$ is a function $I(p) : D^n \rightarrow \{\top, \bot\}$ where n is the arity of p, $\top$ stands for "true" and $\bot$ stands for "false". The interpretation of the equality and the truth tables of the connectives occurring in $Comp(P)$ are defined as usual (see, for example, [Men87]). Any interpretation that takes every expression occurring in $Comp(P)$ to the value $\top$ is a *model of P*.

Now it is possible to introduce an inference procedure for this language. The procedure is *SLDNF*[9]. First we must introduce some notation. In what follows:

- φ_i are literals;
- p_i, q_i are positive literals;
- g_i are positive ground literals;
- δ_i, ψ_i are (possibly empty) conjunctions of literals;
- σ, π are substitutions;
- R stands for "returns": $\psi R \sigma$ holds iff *SLDNF* succeeds on ψ with the substitution σ as an answer, in which case we say that ψ belongs to the *success set R* of the program;
- F stands for "fails": ψF holds iff *SLDNF* fails, in which case we say that ψ belongs to the *finite failure set F* of the program;

[9] *SLDNF* stands for *L*inear Resolution with a *S*election Rule for *D*efinite Clauses, extended with *N*egation by Finite *F*ailure. "Linear" indicates that each inference step uses the most-recently resolved clause as an input, "selection rule" indicates the use of some fixed rule to select the other inputs of each inference, "definite clauses" defines the class of clauses initially tractable by the procedure (a definite clause is a normal clause in which all literals are positive), and "negation by finite failure" indicates that these clauses are extended to accomodate negation - resulting in what we are calling normal clauses - and that negation is interpreted in the specific way presented in the following paragraphs

- Rules:

  Denoting by Def_p the definition of the predicate p in the program:

  - Def_p = the set of clauses in P with p in the head,

  1. $\dfrac{Def_p = \{\}}{\forall \vec{x}[\neg p(\vec{x})]}$
  2. $\dfrac{Def_p = \{p(\vec{t_i}) \leftarrow \psi_i : i = 1, ..., k\} \neq \{\}}{\forall \vec{x}[p(\vec{x}) \leftrightarrow \bigvee_{i=1}^{k} \exists((\vec{x} = \vec{t_i}) \wedge \psi_i)]}$

     where

     (a) $\vec{x}, \vec{t_i}$ are tuples, with the proper arity, of variables ($[x_1, ..., x_m]$) and terms ($[t_{1i}, ..., t_{mi}]$), respectively;
     (b) $\vec{x} = \vec{t_i}$ stands for $x_1 = t_{1i} \wedge ... \wedge x_m = t_{mi}$;
     (c) the scope of the existential quantifier is the variables occurring in the bodies of the clauses in Def_p;
     (d) ψ_i are (possibly empty) conjunctions of literals; and
     (e) the connective $\leftrightarrow$ stands for equivalence.

- Axioms:

  1. equality axioms [Men87]:

     (a) $\forall x(x = x)$ (reflexivity);
     (b) $x_1 = x_2 \rightarrow (C(x_1, x_1) \rightarrow C(x_1, x_2))$ (substitutivity).
     where x, x_1, x_2 are variables, $C(x_1, x_1)$ is a clause and $C(x_1, x_2)$ is the same clause with some (but not necessarily all) occurrences of x_1 replaced by x_2;

  2. $t(x) \neq x$ for each term in which x occurs.

Figure 2.6: Completion $Comp(P)$ of a program P

- *true* stands for the empty query clause;

- *yes* stands for the identity substitution.

The procedure is defined by the inductive rules presented in figure 2.7.

The soundness and completeness of *SLDNF* with respect to this model of completion is obtained from the literature. In [Apt87] we have that *SLDNF* is *sound*, i.e. that given a query ψ and a program P, if (using our notation) $\psi R\sigma$ then $Comp(P) \models \psi\sigma$, and if ψF then $Comp(P) \models \neg\psi$, where $Comp(P) \models \Phi$ means that Φ is a semantic consequence of $Comp(P)$.

The *completeness* result can be found in [Kun89]: for the classes of programs and queries considered in our work (actually, [Kun89] treats slightly more general classes of programs and queries, allowing e.g. cycles and functions), if $Comp(P) \models \psi\sigma$ then $\psi R\sigma$, and if $Comp(P) \models \neg\psi$ then ψF.

This defines a rich subset of first-order logic with a computationally efficient inference procedure and a formally specified declarative semantics.

Example 2.3 *Consider the following program P:*

$p_1(x) \leftarrow p_2(x), \neg p_3(x).$
$p_4(x) \leftarrow p_3(x).$
$p_5(x) \leftarrow p_4(x), \neg p_1(x).$
$p_6(x) \leftarrow p_1(x).$
$p_7(x) \leftarrow p_5(x).$
$p_6(x) \leftarrow p_2(x), \neg p_7(x).$
$p_2(a).$
$p_3(a).$
$p_3(b).$

And the queries:

$p_6(a).$
$p_7(b).$

The predicates in this program have a dependency relation as shown in figure 2.8, where $p_i \rightarrow p_j$ *iff* $p_i \sqsubseteq p_j$.

1. $true\ R\ yes.$

2. $$\frac{(q,\delta), \exists[p \leftarrow \psi] : \sigma = mgu(q,p), (\psi,\delta)\sigma R \pi}{(q,\delta)\ R\ (\sigma\pi)}$$

3. $$\frac{(\neg g,\delta), gF, \delta R \sigma}{(\neg g,\delta)\ R\ \sigma}$$

4. (a) $$\frac{(q,\delta), \neg\exists[p \leftarrow \psi] : \exists mgu(q,p)}{(q,\delta)\ F}$$

   (b) $$\frac{(q,\delta), \forall[p_i \leftarrow \psi_i] : \exists\sigma = mgu(q,p_i) \Rightarrow (\psi_i,\delta)\sigma\ F}{(q,\delta)\ F}$$

5. $$\frac{(\neg g,\delta), g\ R\ yes}{(\neg g,\delta)\ F}$$

Figure 2.7: *SLDNF*

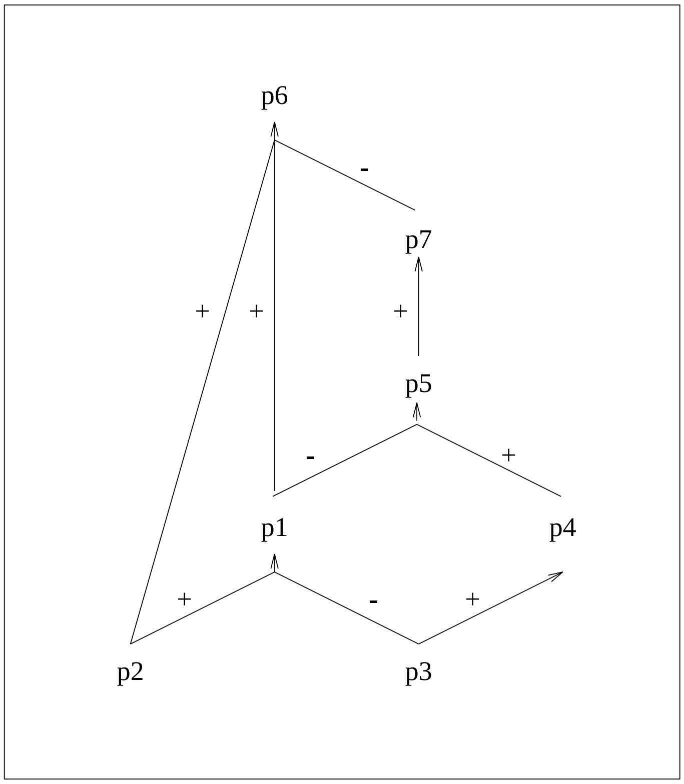

Figure 2.8: Dependency relation for the predicates in the program P

The complete and/or-trees for these queries, indicating whether each node belongs to the success (R) or the finite failure (F) sets, are as follows (figures 2.9 and 2.10), in which linked branches represent "and-ed" dependencies and unlinked branches represent "or-ed" dependencies.

These trees compute the values $\perp$ *for* $p_6(a)$ *(since it belongs to the finite failure set of the program) and* $\top$ *for* $p_7(b)$ *(since it belongs to the success set of the program).*

Anecdotal Example 2.1 *We know the following people*[10]*:*

Alice;	*Andy;*	*Carla;*	*Dave;*	*Flavio;*	*Ian;*
Jane;	*Joanne;*	*Kathleen;*	*Otavio;*	*Robert;*	*William.*

We also know that:

1. *Andy, Carla, Dave, Flavio, Ian, Jane, Joanne and Robert work or study at the Department of Artificial Intelligence at the University of Edinburgh;*
2. *Ian, Joanne, Kathleen and William are blond;*
3. *Kathleen and William are Robert's parents;*
4. *Alice and Otavio are Carla's parents;*
5. *Andy, Carla, Dave, Flavio, Ian, Jane, Joanne and Robert are young adults;*
6. *Flavio, Otavio, Ian and William are overweight; and*
7. *Carla and Kathleen have variable blood pressure.*

When analysing these individuals' fitness for engaging in aerobic exercise sessions and their susceptibility to dehydration, the following statements were accepted as true:

[10] all characters portrayed in this and the following anecdotal examples are purely fictional and any resemblance to any non-fictional people is merely coincidental.

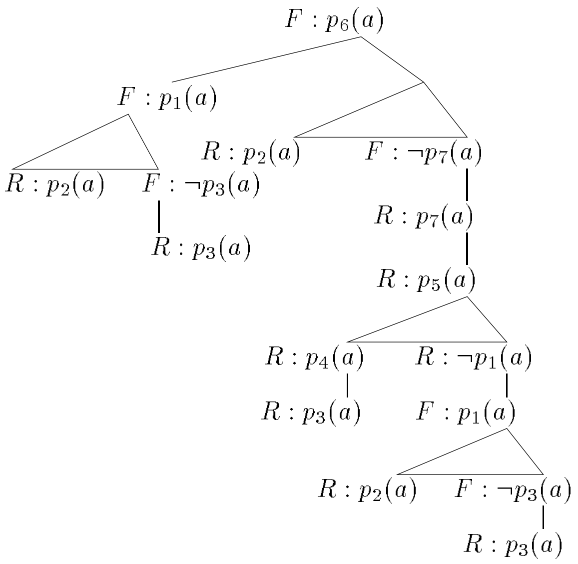

Figure 2.9: and/or-tree for $p_6(a)$

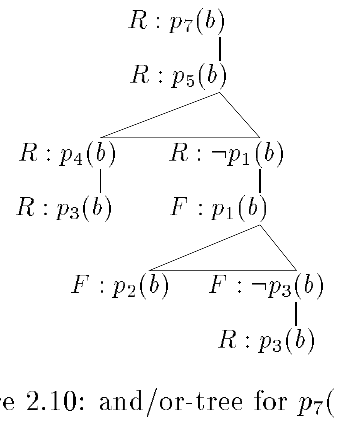

Figure 2.10: and/or-tree for $p_7(b)$

1. *If a person is susceptible to get sunburnt and is not a young adult, then this person is susceptible to dehydration.*

2. *If a person is blond, then this person is susceptible to get sunburnt.*

3. *If a person has blond parents, then this person is blond.*

4. *If a person is overweight, then this person is susceptible to dehydration.*

5. *If a person is a young adult and does not need special care, then this person is fit for aerobics.*

6. *If a person is overweight or potentially hypertense, then this person needs special care.*

7. *If a person has variable blood pressure or is not a young adult and has a parent who has variable blood pressure, then this person is potentially hypertense.*

The questions are:

1. *Is Carla fit for aerobics?*

2. *Is there anyone who is susceptible to dehydration?*

The problem can be stated as a logic program as follows, where x, y, z are variables and the other terms are constants:

$dehyd(x) \leftarrow sunb(x), \neg young(x).$
$sunb(x) \leftarrow blond(x).$
$blond(x) \leftarrow parent(x, y), parent(x, z), blond(y), blond(z).$
$dehyd(x) \leftarrow overw(x).$
$fit(x) \leftarrow young(x), \neg spcare(x).$
$spcare(x) \leftarrow overw(x).$
$spcare(x) \leftarrow hyper(x).$
$hyper(x) \leftarrow varblpres(x).$
$hyper(x) \leftarrow parent(x, y), varblpres(y), \neg young(x).$

$deptai(andy).$	$deptai(carla).$
$deptai(dave).$	$deptai(flavio).$
$deptai(ian).$	$deptai(jane).$
$deptai(joanne).$	$deptai(robert).$
$blond(ian).$	$blond(joanne).$
$blond(kathleen).$	$blond(william).$
$parent(robert, kathleen).$	$parent(robert, william).$
$parent(carla, alice).$	$parent(carla, otavio).$
$young(andy).$	$young(carla).$
$young(dave).$	$young(flavio).$
$young(ian).$	$young(jane).$
$young(joanne).$	$young(robert).$
$overw(flavio).$	$overw(otavio).$
$overw(ian).$	$overw(william).$
$varblpres(carla).$	$varblpres(kathleen).$

The first query is evaluated as represented in the and/or graph in figure 2.11, concluding that "Carla is not fit for aerobics".

The second query can be solved by several different ways. For example, it could be evaluated as represented in figure 2.12, concluding that Kathleen is susceptible to dehydration, hence "someone is susceptible to dehydration".

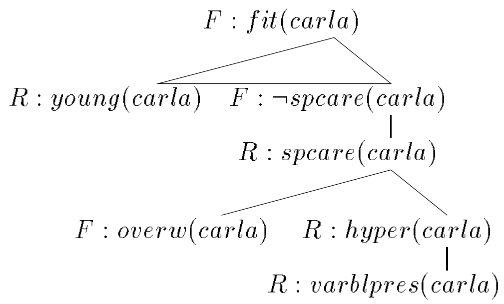

Figure 2.11: and/or-tree for "is Carla fit for aerobics?"

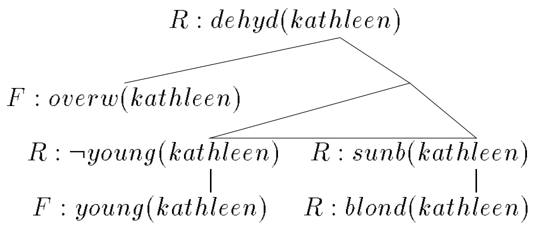

Figure 2.12: one possible and/or-tree for "is anyone susceptible to dehydration?"

3

A Language Supporting Fuzzy Predicates

The relationship between fuzzy logics and the resolution principle is well established. Since [Lee72], one of the pioneering works in the area, several proposals have been made, aiming at richer languages in respect of both the logical and the fuzzy relations supported.

In [Lee72] the language is limited to *definite clauses* [Apt87, Hog90] *allowing fuzzy predicates with truth-values always greater than 0.5*[1]. The semantics of the relevant connectives is defined according to Zadeh's triangular norms and conorms and resolution is extended to propagate truth-values in a way that is sound and complete with respect to the Herbrand interpretation of sets of clauses. Several implementations based on [Lee72] have been proposed, e.g. the ones described in [Hin86, IK85, Orc89].

More recent developments [Fit88, Fit90, KS91, Sha83, vE86] have focused on fixpoint semantics, either working with definite programs or approaching the definition of negation by means other than finite failure. We adopt negation by finite failure in this work, in order to have the more conventional languages which are based on this principle (e.g. pure PROLOG) as proper subsets of our language. This choice is corroborated by the results found in [Tur89, CL89, Fit85, Kun87,

[1] the limitations on the types of clauses and range of truth-values are conditions imposed to obtain soundness for the specific resolution procedure employed in [Lee72]

Kun89, Kun90], which determine large classes of *normal programs* with a well-defined declarative semantics[2].

In what follows we introduce a language to deal with fuzzy predicates. First we present the language, then its model theory and inference procedure.

3.1 Reasoning with Fuzzy Predicates

Fuzzy predicates can be defined by analogy with the concept of fuzzy sets. The interpretation of predicates can be generalised to a function $I(p) : D^n \rightarrow [0,1]$, with the extreme values corresponding to the previous values $\top$ and $\bot$ (namely, $\top \equiv 1$ and $\bot \equiv 0$). This function can be construed as a fuzzy membership function and the logical connectives can be interpreted as fuzzy set operators - '$\neg$' corresponding to complementation, '$\vee$' corresponding to union, '$\wedge$' corresponding to intersection, and '$\leftrightarrow$' corresponding to set-equivalence. Intuitively, the semantics of a closed formula becomes a "degree of truth", rather than simply one value out of $\{\top, \bot\}$. Let τ denote this value and $\mathcal{T}(\psi, \tau)$ state that "the truth-degree of ψ is τ". This evaluation can be made operational using an *extended SLDNF (e-SLDNF)* procedure, to be related to the model of an *extended completion* of a program P (*e-Comp*(P)). We assume that the *unit clauses* (and only them) in the program express truth-degrees, that is, unit clauses are of the form $\mathcal{T}(p, \tau)$, where $\tau > 0$.[3]

The *extended completion* of a program P (*e-Comp* (P)) is defined as presented in figure 3.1.

Two classes of formulae can be identified in *e-Comp*(P):

- *unit formulae*, generated by rule 1 or from the unit clauses occurring in P; and

[2] see chapter 2

[3] the restrictions on how to declare truth-degrees are imposed to avoid ambiguity, redundancy and conflicting declarations. The language presented here is monotonic and does not contain mechanisms to resolve truth-degrees if they are declared for unit clauses as well as larger constructs (e.g. general normal clauses)

- *equivalence formulae*, i.e. the remaining ones, all of them containing the connective $\leftrightarrow$.

The connectives occurring in *e-Comp*(P) are interpreted according to the truth-functions defined below:

Assuming that:

- $\mathcal{T}(\delta, \tau_\delta)$, and
- $\mathcal{T}(\psi, \tau_\psi)$

We have that:

- $\mathcal{T}((\delta \wedge \psi), \tau) \Rightarrow \tau = min\{\tau_\delta, \tau_\psi\}$
- $\mathcal{T}((\delta \vee \psi), \tau) \Rightarrow \tau = max\{\tau_\delta, \tau_\psi\}$
- $\mathcal{T}((\neg\delta), \tau) \Rightarrow \tau = 1 - \tau_\delta$
- $\begin{cases} \mathcal{T}((\delta \leftrightarrow \psi), 1) & \Rightarrow \tau_\delta = \tau_\psi \\ \mathcal{T}((\delta \leftrightarrow \psi), 0) & \Rightarrow \tau_\delta \neq \tau_\psi \end{cases}$

The completion of a conventional program defines a unique model for the program. For the extended completion to do the same, a necessary condition is to fix the truth-values for the unit clauses occurring in P as values greater than 0. This condition is also sufficient, as all the other formulae in *e-Comp*(P) - i.e. the equivalence formulae and the unit formulae generated by rule 1 - must have truth-values equal to 1 in the model of the program.

A model for a program containing fuzzy predicates is any interpretation for which every expression φ occurring in *e-Comp*(P) has a truth-value $\tau > 0$.

Our notation for logic programs and the *e-SLDNF* procedure is basically the notation used in figure 2.7, with the following alterations:

- Rules:

  1. $\dfrac{Def_p = \{\}}{\forall \vec{x}[\mathcal{T}(\neg p(\vec{x}), 1)]}$

  2. $\dfrac{Def_p = \{p(\vec{t_i}) \leftarrow \psi_i : i = 1, ..., k\} \neq \{\}}{\begin{array}{l}\forall \vec{x}[\mathcal{T}(p(\vec{x}), \tau) \leftrightarrow max\{\tau_i : (\vec{x} = \vec{t_i}) \wedge [(\psi_i \neq \{\} \wedge \mathcal{T}(\psi_i, \tau_i)) \\ \vee(\psi_i = \{\} \wedge \mathcal{T}(p(\vec{t_i}), \tau_i))]\} = \tau]\end{array}}$

     where

     (a) Def_p is the set of clauses in P with p in the head;

     (b) $\vec{x}, \vec{t_i}$ are tuples of variables ($[x_1, ..., x_m]$) and terms ($[t_{1i}, ..., t_{mi}]$), respectively;

     (c) $\vec{x} = \vec{t_i}$ stands for $x_1 = t_{1i} \wedge ... \wedge x_m = t_{mi}$;

     (d) ψ_i are (possibly empty) conjunctions of literals;

     (e) the connective $\leftrightarrow$ stands for equivalence.

- Axioms:

  – same as in figure 2.6.

Figure 3.1: Extended completion of a program P

- R^e stands for "returns with a truth-value greater than 0": $\psi\ R^e\ (\sigma, \tau)$ holds iff *e-SLDNF* succeeds, assigning a truth-value τ to ψ, with the substitution σ as an answer;

- F^e stands for "fails": ψF^e holds iff *e-SLDNF* fails, implying the assignment of a truth-value $\tau = 0$ to ψ.

e-SLDNF is defined inductively as presented in figure 3.2.

The intuition underlying the definitions of *e-Comp*(P) and *e-SLDNF* is that we need the truth-degrees declared in the unit clauses in a program P "transferred" to the heads of the clauses in P in a consistent way. Central to these definitions is the notion of completed database, which makes it possible to define the truth-degrees to be "transferred" as unique.

There are indications that the completeness results obtained for programs without fuzzy predicates can be mimicked for those containing fuzzy predicates (indeed, early drafts of this thesis contained the sketch of a proof of these results – later showed to be faulty – based on the supposition that fuzzy predicates could be rewritten as non-fuzzy ones and the "transferring" of truth-degrees could be dealt with independently from the evaluation of queries to non-fuzzy programs). These results have been object of research, and there are indications that they are at least as complex – and hard to achieve – as their non-fuzzy counterparts exhibited in [Kun89].

Example 3.1 *Consider the following program:*

$p_1(x) \leftarrow p_2(x), \neg p_3(x).$
$p_4(x) \leftarrow p_3(x).$
$p_5(x) \leftarrow p_4(x), \neg p_1(x).$
$p_6(x) \leftarrow p_1(x).$
$p_7(x) \leftarrow p_5(x).$
$p_6(x) \leftarrow p_2(x), \neg p_7(x).$

$\mathcal{T}(p_2(a), 0.4).$
$\mathcal{T}(p_3(a), 0.2).$
$\mathcal{T}(p_3(b), 0.1).$

And the queries:

1. $true\ R^e\ (yes, 1)$

2. (a) $$\frac{(q,\delta), max \left\{ \begin{array}{ll} \tau_i : & [p_i \leftarrow \psi_i], \sigma_i = mgu(q,p_i), (\psi_i,\delta)\sigma_i\ R^e\ (\pi_i,\tau_i) \vee \\ & \sigma_i = mgu(q,p_i), (\delta)\sigma_i\ R^e\ (\pi_i,\tau_i'), \\ & \mathcal{T}((p_i)\sigma_i, \tau_i''), min\{\tau_i', \tau_i''\} = \tau_i \end{array} \right\} = \tau}{(q,\delta)\ R^e\ (\sigma\pi, \tau)}$$
   where $\sigma\pi$ is the substitution that generates τ and the ψ_i are non-empty conjunctions.

   (b) $$\frac{(\neg g,\delta), g\ R^e\ (yes,\tau'), \tau' < 1, \delta\ R^e\ (\sigma,\tau''), min\{(1-\tau'),\tau''\} = \tau}{(\neg g,\delta)\ R^e\ (\sigma,\tau)}$$

3. $$\frac{(\neg g,\delta), gF^e, \delta\ R^e\ (\sigma,\tau)}{(\neg g,\delta)\ R^e\ (\sigma,\tau)}$$

4. (a) $$\frac{(q,\delta), \neg\exists[p \leftarrow \psi] : \exists mgu(q,p)}{(q,\delta)\ F^e}$$

   (b) $$\frac{(q,\delta), \forall[p_i \leftarrow \psi_i] : \exists\sigma = mgu(q,p_i) \Rightarrow (\psi_i,\delta)\sigma\ F^e}{(q,\delta)\ F^e}$$

5. $$\frac{(\neg g,\delta), g\ R^e\ (yes,1)}{(\neg g,\delta)\ F^e}$$

Figure 3.2: *e-SLDNF*

$p_6(a)$.
$p_7(b)$.

The complete and/or-trees for these queries are as presented in figures 3.3 and 3.4. The trees compute the truth-degrees for $p_6(a)$ and $p_7(b)$. It is worth observing that this language subsumes the language without fuzzy predicates, having that one as the special class of programs in which every clause has truth-degree 1 or 0. The truth-values obtained in example 2.3 could be derived here if the program had its unit clauses replaced by the following clauses:

$\mathcal{T}(p_2(a), 1)$.
$\mathcal{T}(p_3(a), 1)$.
$\mathcal{T}(p_3(b), 1)$.

Anecdotal Example 3.1 *On reviewing our statements on the anecdotal example 2.1, we conclude that we are not being fair when we are so categorical about statements such as one's youth or plumpness. To make things fairer, we update the unit clauses in our programs as below:*

$\mathcal{T}(deptai(andy), 1)$.	$\mathcal{T}(deptai(carla), 1)$.
$\mathcal{T}(deptai(dave), 1)$.	$\mathcal{T}(deptai(flavio), 1)$.
$\mathcal{T}(deptai(ian), 1)$.	$\mathcal{T}(deptai(jane), 1)$.
$\mathcal{T}(deptai(joanne), 1)$.	$\mathcal{T}(deptai(robert), 1)$.
$\mathcal{T}(blond(ian), 0.9)$.	$\mathcal{T}(blond(joanne), 0.9)$.
$\mathcal{T}(blond(kathleen), 0.7)$.	$\mathcal{T}(blond(william), 0.6)$.
$\mathcal{T}(blond(jane), 0.7)$.	
$\mathcal{T}(parent(robert, kathleen), 1)$.	$\mathcal{T}(parent(robert, william), 1)$.
$\mathcal{T}(parent(carla, alice), 1)$.	$\mathcal{T}(parent(carla, otavio), 1)$.
$\mathcal{T}(young(andy), 0.9)$.	$\mathcal{T}(young(carla), 0.9)$.
$\mathcal{T}(young(dave), 0.9)$.	$\mathcal{T}(young(flavio), 0.9)$.
$\mathcal{T}(young(ian), 0.9)$.	$\mathcal{T}(young(jane), 0.9)$.
$\mathcal{T}(young(joanne), 0.9)$.	$\mathcal{T}(young(robert), 0.9)$.
$\mathcal{T}(young(alice), 0.3)$.	$\mathcal{T}(young(otavio), 0.3)$.
$\mathcal{T}(young(kathleen), 0.4)$.	$\mathcal{T}(young(william), 0.3)$.
$\mathcal{T}(overw(flavio), 0.4)$.	$\mathcal{T}(overw(otavio), 0.7)$.
$\mathcal{T}(overw(ian), 0.6)$.	$\mathcal{T}(overw(william), 0.5)$.
$\mathcal{T}(overw(andy), 0.2)$.	$\mathcal{T}(overw(robert), 0.3)$.
$\mathcal{T}(varblpres(carla), 0.7)$.	$\mathcal{T}(varblpres(kathleen), 0.8)$.
$\mathcal{T}(varblpres(flavio), 0.3)$.	

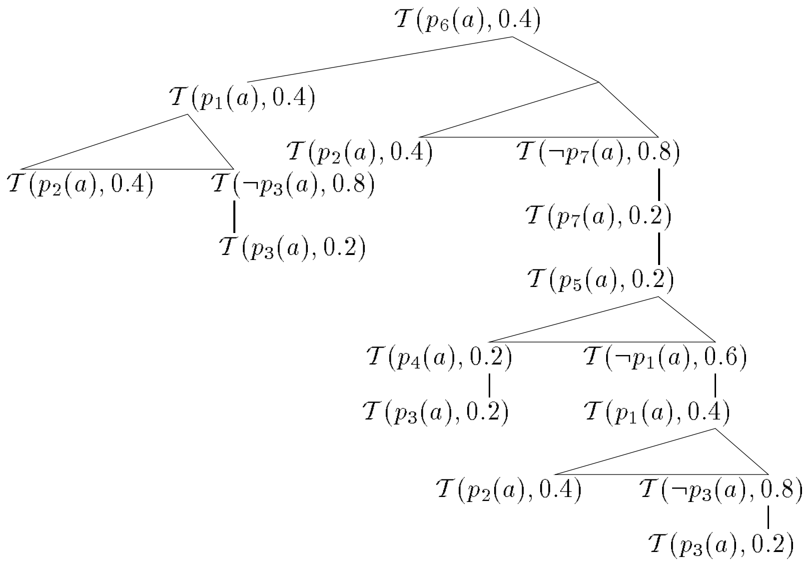

Figure 3.3: and/or-tree for $p_6(a)$

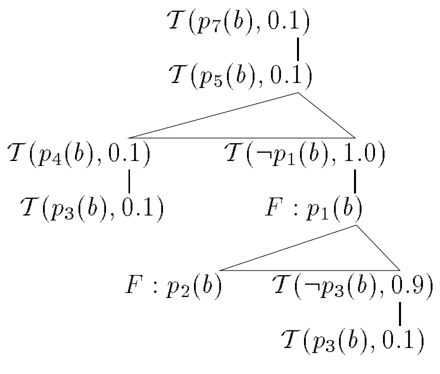

Figure 3.4: and/or-tree for $p_7(b)$

With these unit clauses, the possible ground answers for the query "Is there anyone who is susceptible to dehydration?" are[4]*:*

- $\mathcal{T}(dehyd(andy), 0.2)$.
- $\mathcal{T}(dehyd(flavio), 0.4)$.
- $\mathcal{T}(dehyd(ian), 0.6)$.
- $\mathcal{T}(dehyd(jane), 0.1)$.
- $\mathcal{T}(dehyd(joanne), 0.1)$.
- $\mathcal{T}(dehyd(kathleen), 0.7)$.
- $\mathcal{T}(dehyd(otavio), 0.7)$.
- $\mathcal{T}(dehyd(robert), 0.3)$.
- $\mathcal{T}(dehyd(william), 0.6)$.

Since the interpreter selects the answer that is the "most correct", the returned truth-degree is actually 0.7, since it is the maximum truth-degree that we can obtain for the query.

[4] Each alternative ground answer is generated by traversing an alternative and/or tree that resolves the query, i.e. by exhaustively grounding the free variables in the query with a different term. There is a direct correspondence between this and the execution of step $2(a)$ in figure 3.2, which generates exhaustively all answers to a query in order to find the one with maximum truth-degree

4

A Language Supporting Probabilities on the Domain

The problem of representing and reasoning with statistical knowledge has received some attention recently [AH89, Bac88, Bac90a, Bac90c, Hal90]. This problem can be roughly characterised as the problem of representing in a first-order language terms of the form $\mathcal{P}_{\vec{x}}(\psi)$[1]. For example, given a program like the one in example 3.1, we may have interest in *statistical information* such as "what is the probability that there is a constant $\hat{a}$ such that $p_7(\hat{a})$?"

In [AH89] we have the result that the set of valid formulae for first-order logic containing statistical terms is not recursively enumerable, implying that a complete proof procedure for this logic does not exist. Two different ways of constraining the language to achieve proof-theoretic completeness have been proposed:

- in [Bac88, Bac90a] the *probability measures* are relaxed to *non-σ-additive measures*, that is, the general probability axiom stating that "the probability of any *(infinitely) countable* set of pairwise disjoint events equals the sum of the probabilities of those events" is reduced to the case of *finite* sets of events. Moreover, the measures range on *real closed fields*[2] rather

[1] to be read as "the probability of selecting a vector of instances for the variables in $\vec{x}$ that make ψ true"

[2] a *real closed field* is a theory with equality containing the functions $+$ and $\times$ and the predicate $<$ and obeying the following axioms [Sho67]:

than on real numbers.

- in [AH89, Hal90] the domain of discourse is *bounded in size*, i.e. it contains a number of elements not greater than a fixed N.

Our base language to be extended to contain statistical expressions[3] obeys all these constraints: its domain is always finite and of fixed cardinality, so it is always bounded in size, σ-additivity coincides with finite additivity for finite domains and, as we intend to *compute* probabilities, field-valued measures are sufficient since, as pointed out in [Bac88], "computers are only capable of dealing with rational numbers (and only a finite set of them)".

On the other hand, the language introduced here extends the aforementioned results in two senses:

- we allow the occurrence of *fuzzy events*, i.e. statistical events that are characterised by fuzzy sets, and
- following a line suggested in [Hal90], we admit the existence of *non-measurable events* and the consequent need for *inner and outer approximations* for statistical measures.

$\forall x, y, z[(x+y)+z = x+(y+z)]$.	$\exists 0 \forall x[x+0 = x]$.
$\exists -1 \forall x[x+(-1 \times x) = 0]$.	$\forall x, y[x+y = y+x]$.
$\forall x, y, z[(x \times y) \times z = x \times (y \times z)]$.	$\exists 1 \forall x[x \times 1 = x]$.
$\forall x[x \neq 0 \rightarrow \exists y(x \times y = 1)]$.	$\forall x, y[x \times y = y \times x]$.
$\forall x, y, z[x \times (y+z) = (x \times y)+(x \times z)]$.	$0 \neq 1$.
$\forall x[\neg x < x]$.	$\forall x, y, z[x < y \rightarrow y < z \rightarrow x < z]$.
$\forall x, y[x < y \vee x = y \vee y < x]$.	$\forall x, y, z[x < y \rightarrow x+z < y+z]$.
$\forall x, y[0 < x \rightarrow 0 < y \rightarrow 0 < x \times y]$.	$\forall x[0 < x \rightarrow \exists y(y \times y = x)]$.

The set $\Re$ of real numbers is a particular real closed field in which the *continuity property* is valid [IK77]:

> for each nonempty subsets A and B of $\Re$, with $a < b$ for each pair $a \in A$ and $b \in B$, satisfying $\Re = A \cup B$ and $A \cap B = \{\}$, there exists a unique element $x \in \Re$ such that for every $a \in A, a \leq x$, and for every $b \in B, x \leq b$.

[3] recall that our base language is the language of normal clauses containing fuzzy predicates, under allowedness and strictness with respect to queries, and non-cyclical programs

4.1 Reasoning with Probabilities

Given a program P, the set of solutions with truth-values greater than 0 for a query ψ is always finite. This set also defines a fuzzy set of tuples of elements of D - the domain of P.

If our language is extended to accommodate the specification of probability measures of algebras of a partition of D through their bases, the set of solutions of ψ can be interpreted as a fuzzy event in the appropriate cartesian product of D, and upper and lower bounds can be evaluated for its probabilities using the measure of the corresponding cartesian product algebra.

The language is extended as follows:

- special unit formulae of the form $\mathcal{P}(S^c, \rho)$ are used to specify probability measures for D, i.e. a collection of expressions of the form $\mathcal{P}({S^c}_{ij}, \rho_{ij})$ is attached to P, where the ${S^c}_{ij}$ form the bases of algebras χ_{Di} of a partition of D and ρ_{ij} is the probability of ${S^c}_{ij}$;
- some definitions are implicitly assumed as part of our inference procedure: the definitions of the operations of *addition* $(+)$ and *multiplication* $(\times)$, of the *relations* $>$ and $=$, and of the properties of *non-negativity* $(\rho \geq 0 \leftarrow \mathcal{P}(S^c, \rho))$, *finite additivity* $(\mathcal{P}(\cup_1^n {S^c}_i, \rho) \leftarrow \mathcal{P}({S^c}_1, \rho_1), ..., \mathcal{P}({S^c}_n, \rho_n), \rho = \rho_1 + ... + \rho_n)$ and *total probability* $(\mathcal{P}(D, 1))$.
- special second-order expressions of the forms $\mathcal{P}_*(S, \psi, \rho_*)$ and $\mathcal{P}^*(S, \psi, \rho^*)$ are introduced, to be read as "the lower and the upper bounds for the probability of having a tuple of instances for the variables in S which satisfies ψ are ρ_* and ρ^*".
- special second-order expressions of the forms $\mathcal{P}_*(S, \psi_1|\psi_2, \rho_*)$ and $\mathcal{P}^*(S, \psi_1|\psi_2, \rho^*)$ are introduced, to be read as "the lower and the upper bounds for the probability of having a tuple of instances for the variables in S which satisfies ψ_1 given ψ_2 are ρ_* and ρ^*".

Intuitively, we want to be able to answer the query "what is the probability of having a tuple of terms S that makes ψ true?" or, taking into account the fuzzyness of ψ and the program P, " what is the expected truth-degree of ψ when we range the tuple S through the whole domain of discourse?". The answer (ρ) to a query like this can be obtained by generating exhaustively the alternative answers to ψ with the variables occurring in both S and ψ admitted free in ψ, and then calculating the expected truth-degree of ψ taking into account the declared probability of each tuple of ground terms that generated each truth-degree. Since probabilities of generic tuples of ground terms may have to be estimated by means of inner and outer measures (as not all tuples belong to cartesian products of the algebra χ_{Di}), the value (ρ) must, in the general case, be approximated from below and from above by (ρ_*) and (ρ^*), respectively. Similarly, for conditional probabilities, we want to be able to answer the query "what is the probability of having a tuple S that makes ψ_1 true whenever ψ_2 is true?", or "what is the expected truth-degree of ψ_1 when we range the tuple S through those elements in the domain that make ψ_2 true?".

These expressions are evaluated as follows:

- $\mathcal{P}_*(S, \psi, \rho_*)$ and $\mathcal{P}^*(S, \psi, \rho^*)$:

  1. generate K_ψ, the finite fuzzy set of tuples of instances of the free variables in ψ which associate a non-zero truth-degree τ to ψ.

     If ψ does not have free variables, K_ψ is the singleton set containing the tuple of terms occurring in ψ with its respective truth-degree.

     If ψ contains free variables, then K_ψ is generated by substituting exhaustively each free variable in ψ by elements of D and then selecting the substitutions which generate the desired truth-degrees.

  2. generate K_ψ^S - the projection of K_ψ over S: select from the tuple of free variables in ψ those which are also in S, and extract the corresponding tuples of instances from K_ψ.

  3. (a) If $K_\psi^S \neq \{\}$ then:

i. generate the cartesian product algebra and measure of the same arity as the tuples in K_ψ^S assuming the elements of D to be statistically independent.

ii. generate ρ_* and ρ^*:

$\rho_* = \sum_{\vec{S}_i \subseteq K_\psi^S} \mathcal{P}_m(\vec{S}_i) \times min\{\tau : \mu(\vec{K}) = \tau, \vec{K} \in \vec{S}_i \cap K_\psi^S\}, \vec{S}_i \in \chi'_D$

$\rho^* = \sum_{\vec{S}_i \cap K_\psi^S \neq \{\}} \mathcal{P}_m(\vec{S}_i) \times max\{\tau : \mu(\vec{K}) = \tau, \vec{K} \in \vec{S}_i \cap K_\psi^S\}, \vec{S}_i \in \chi'_D$

where m is the arity of the tuples in K_ψ^S.

(b) If $K_\psi^S = \{\}$ then

make $\rho_* = \rho^* = \tau$, where $\mathcal{T}(\psi, \tau)$.

- $\mathcal{P}_*(S, \psi_1|\psi_2, \rho_*)$ and $\mathcal{P}^*(S, \psi_1|\psi_2, \rho^*)$:

1. generate $K_{(\psi_1,\psi_2)}^S$ and $K_{(\neg\psi_1,\psi_2)}^S$.

2. generate $min_\wedge, min'_\wedge, min_\wedge^\neg, max_\wedge, max'_\wedge$ and $max_\wedge^\neg$:

   - $min_\wedge = \sum_{\vec{S}_i \subseteq K_{(\psi_1,\psi_2)}^S} \mathcal{P}_m(\vec{S}_i) \times min\{\tau : \mu(\vec{K}) = \tau, \vec{K} \in \vec{S}_i \cap K_{(\psi_1,\psi_2)}^S\}, \vec{S}_i \in \chi'_D$
   - $min'_\wedge = \sum_{\vec{S}_i \cap K_{(\psi_1,\psi_2)}^S \neq \{\}} \mathcal{P}_m(\vec{S}_i) \times min\{\tau : \mu(\vec{K}) = \tau, \vec{K} \in \vec{S}_i \cap K_{(\psi_1,\psi_2)}^S\}, \vec{S}_i \in \chi'_D$
   - $min_\wedge^\neg = \sum_{\vec{S}_i \subseteq K_{(\neg\psi_1,\psi_2)}^S} \mathcal{P}_m(\vec{S}_i) \times min\{\tau : \mu(\vec{K}) = \tau, \vec{K} \in \vec{S}_i \cap K_{(\neg\psi_1,\psi_2)}^S\}, \vec{S}_i \in \chi'_D$
   - $max_\wedge = \sum_{\vec{S}_i \cap K_{(\psi_1,\psi_2)}^S \neq \{\}} \mathcal{P}_m(\vec{S}_i) \times max\{\tau : \mu(\vec{K}) = \tau, \vec{K} \in \vec{S}_i \cap K_{(\psi_1,\psi_2)}^S\}, \vec{S}_i \in \chi'_D$
   - $max'_\wedge = \sum_{\vec{S}_i \subseteq K_{(\psi_1,\psi_2)}^S} \mathcal{P}_m(\vec{S}_i) \times max\{\tau : \mu(\vec{K}) = \tau, \vec{K} \in \vec{S}_i \cap K_{(\psi_1,\psi_2)}^S\}, \vec{S}_i \in \chi'_D$
   - $max_\wedge^\neg = \sum_{\vec{S}_i \cap K_{(\neg\psi_1,\psi_2)}^S \neq \{\}} \mathcal{P}_m(\vec{S}_i) \times max\{\tau : \mu(\vec{K}) = \tau, \vec{K} \in \vec{S}_i \cap K_{(\neg\psi_1,\psi_2)}^S\}, \vec{S}_i \in \chi'_D$

3. generate ρ_* and ρ^*:

- $\rho_* = \begin{cases} \dfrac{min_\wedge}{max'_\wedge + max_\wedge^\neg}, & max'_\wedge + max_\wedge^\neg \neq 0 \\ 0, & max'_\wedge + max_\wedge^\neg = 0 \end{cases}$
- $\rho^* = \begin{cases} min\{1, \dfrac{max_\wedge}{min'_\wedge + min_\wedge^\neg}\}, & min'_\wedge + min_\wedge^\neg \neq 0 \\ 0, & min'_\wedge + min_\wedge^\neg = 0 \end{cases}$

Since probabilities are completely defined by measures on the constants of the language, terms of the forms $\mathcal{P}_*(S, \psi, \rho_*)$, $\mathcal{P}^*(S, \psi, \rho^*)$, $\mathcal{P}_*(S, \psi_1|\psi_2, \rho_*)$ and $\mathcal{P}^*(S, \psi_1|\psi_2, \rho^*)$ never occur as heads of program clauses. Moreover, these terms only admit truth-degrees in $\{0, 1\}$.

Example 4.1 *Consider the program of example 3.1, with the unit clauses declaring truth-degrees replaced by the clauses below:*

$\mathcal{T}(p_2(a), 0.4).$
$\mathcal{T}(p_2(c), 0.7).$
$\mathcal{T}(p_2(e), 0.2).$
$\mathcal{T}(p_2(d), 0.5).$
$\mathcal{T}(p_3(a), 0.2).$
$\mathcal{T}(p_3(b), 0.1).$
$\mathcal{T}(p_3(e), 0.2).$
$\mathcal{T}(p_3(c), 0.8).$
$\mathcal{T}(p_3(f), 0.4).$

And with the following additional clauses representing statistical knowledge:

$\mathcal{P}(\{a, c\}, 0.5).$
$\mathcal{P}(\{e, d, b\}, 0.2).$
$\mathcal{P}(\{f\}, 0.3).$

Where each clause '$\mathcal{P}(S, \rho)$' is to be read as "the probability of randomly selecting an element from the domain which belongs to S is ρ".

And the queries:

$\mathcal{P}_*([x], p_6(x), \rho_*).$
$\mathcal{P}^*([x], p_6(x), \rho^*).$
$\mathcal{P}_*([x], p_6(a), \rho_*).$
$\mathcal{P}_*([x, y], (p_6(x), p_6(y)), \rho_*).$
$\mathcal{P}_*([x], p_6(x)|p_7(x), \rho_*).$

These queries are evaluated as follows:

- $\mathcal{P}_*([x], p_6(x), \rho_*)$.:

  Making $\psi = p_6(x)$, we have, by repeated application of the resolution procedure presented in figure 3.2, that $K_\psi = \{a/0.4, c/0.2, e/0.2, d/0.5\}$, where α/τ indicates that the degree to which α is a member of K_ψ is τ, i.e. if we have that $\mathcal{T}(p_6(\alpha), \tau)$ then α/τ will be a member of $K_{p6(x)}$.

  since x is the only free variable in ψ, we have that $K_\psi^S = K_\psi \neq \{\}$. Since $\mathcal{P}(\{a, c\}, 0.5)$ and $min\{0.4, 0.2, 0.2, 0.5\} = 0.2$, we have that:

  $\rho_* = 0.5 \times 0.2 = 0.1.$

- $\mathcal{P}^*([x], p_6(x), \rho^*).$:

  Like in the previous case, we have that $K_\psi^S = K_\psi = \{a/0.4, c/0.2, e/0.2, d/0.5\} \neq \{\}$. *The sets* $\{a, c\}$ *and* $\{e, d, b\}$ *have non-empty intersection with* K_ψ^S, *and* $\mathcal{P}(\{a, c\}, 0.5)$ *and* $\mathcal{P}(\{e, d, b\}, 0.2)$. *Furthermore,* $max\{\tau : \mu(K) = \tau, K \in \{a, c\} \cap K_\psi^S\} = 0.4$ *and* $max\{\tau : \mu(K) = \tau, K \in \{e, d, b\} \cap K_\psi^S\} = 0.5$. *Hence,* $\rho^* = 0.5 \times 0.4 + 0.2 \times 0.5 = 0.3$.

  These two results tell us that if we could evaluate the expected truth-value of $p_6(x)$, *it would belong to the interval* $[0.1, 0.3]$.

- $\mathcal{P}_*([x], p_6(a), \rho_*).$:

  In this case, $K_\psi^S = \{\}$, *thus* $\rho_* = \tau$, *where* $\mathcal{T}(\psi, \tau)$.

  $\mathcal{T}(p_6(a), 0.4) \Rightarrow \rho_* = 0.4$.

- $\mathcal{P}_*([x, y], (p_6(x), p_6(y)), \rho_*).$:

  To evaluate this query we need the probability measure $\mathcal{P}_{2*}$ *on the cartesian power* $\vec{\chi}_D$, *specified by:*

  - $\mathcal{P}_2(\{[a, a], [a, c], [c, a], [c, c]\}, 0.25)$.
  - $\mathcal{P}_2(\{[a, e], [a, d], [a, b], [c, e], [c, d], [c, b]\}, 0.10)$.
  - $\mathcal{P}_2(\{[e, a], [d, a], [b, a], [e, c], [d, c], [b, c]\}, 0.10)$.
  - $\mathcal{P}_2(\{[a, f], [c, f]\}, 0.15)$.
  - $\mathcal{P}_2(\{[f, a], [f, c]\}, 0.15)$.
  - $\mathcal{P}_2(\{[f, e], [f, d], [f, b]\}, 0.06)$.
  - $\mathcal{P}_2(\{[e, f], [d, f], [b, f]\}, 0.06)$.
  - $\mathcal{P}_2(\{[f, f]\}, 0.09)$.
  - $\mathcal{P}_2(\{[e, e], [e, d], [e, b], [d, e], [d, d], [d, b], [b, e], [b, d], [b, b]\}, 0.04)$.

making $\psi = (p_6(x), p_6(y))$, *we have that*

$$K^S_\psi = K_\psi = \{ \begin{array}{l} [a,a]/0.4, [a,c]/0.2, [a,e]/0.2, [a,d]/0.4, \\ [c,a]/0.2, [c,c]/0.2, [c,e]/0.2, [c,d]/0.2, \\ [e,a]/0.2, [e,c]/0.2, [e,e]/0.2, [e,d]/0.2, \\ [d,a]/0.4, [d,c]/0.2, [d,e]/0.2, [d,d]/0.5\}. \end{array}$$

Thus,

$\rho_* = 0.25 \times 0.2 = 0.05.$

- $\mathcal{P}_*([x], p_6(x)|p_7(x), \rho_*)$.:

  For $\psi = (p_6(x) \wedge p_7(x))$ *we have that:*

  $K^S_\psi = \{a/0.2, c/0.2, e/0.2\}$

  $min_\wedge = 0.2 \times 0.5 = 0.1$

  $max'_\wedge = 0.2 \times 0.5 = 0.1$

  For $\psi = (\neg p_6(x) \wedge p_7(x))$ *we have:*

  $K^S_\psi = \{a/0.2, c/0.8, e/0.2, b/0.1, f/0.4\}$

  $max^\neg_\wedge = 0.8 \times 0.5 + 0.2 \times 0.2 + 0.4 \times 0.3 = 0.56$

  Hence, we have:

  $$\rho_* = \frac{min_\wedge}{max'_\wedge + max^\neg_\wedge} = \frac{0.1}{0.1 + 0.56} = 0.152$$

If the program does not contain fuzzy predicates, i.e. if every clause in the program has a truth-degree which is either 0 or 1, then the language is an implementation of a significant portion of the logic Lp, introduced in [Bac88].

Example 4.2 *(a program not containing fuzzy predicates):*

Consider the program of example 4.1, with the unit clauses declaring truth-degrees replaced by the clauses below:

$\mathcal{T}(p_2(a), 1.0)$.
$\mathcal{T}(p_2(c), 1.0)$.
$\mathcal{T}(p_2(e), 1.0)$.
$\mathcal{T}(p_2(d), 1.0)$.
$\mathcal{T}(p_3(a), 1.0)$.
$\mathcal{T}(p_3(b), 1.0)$.
$\mathcal{T}(p_3(e), 1.0)$.
$\mathcal{T}(p_3(c), 1.0)$.
$\mathcal{T}(p_3(f), 1.0)$.

And the queries:

$\mathcal{P}^*([x], p_6(x), \rho^*)$.
$\mathcal{P}^*([x], p_6(x)|p_2(x), \rho^*)$.

These queries are evaluated as follows:

- $\mathcal{P}^*([x], p_6(x), \rho^*)$.:

  Making $\psi = p_6(x)$, *we have that* $K_\psi^S = K_\psi = \{d/1.0\}$.

  $\rho^* = 0.2 \times 1.0 = 0.2$.

- $\mathcal{P}^*([x], p_6(x)|p_2(x), \rho^*)$.:

  For $\psi = (p_6(x) \wedge p_2(x))$ *we have that:*

  $K_\psi^S = \{d/1.0\}$

  $max_\wedge = 0.2$

  $min'_\wedge = 0.2$

  For $\psi = (\neg p_6(x) \wedge p_2(x))$ *we have:*

  $K_\psi^S = \{a/1.0, c/1.0, e/1.0\}$

  $min_\wedge^\neg = 0.5$

  This gives us:

  $$\rho^* = \frac{max_\wedge}{min'_\wedge + min_\wedge^\neg} = \frac{0.2}{0.2 + 0.5} = 0.286$$

A *discrete probability measure* is a measure defined on the whole power set 2^D instead of only a particular algebra $\chi_D \subset 2^D$ [Bun85, FH89b, CdS91, CdSB90]. When a program does not contain fuzzy predicates and has a discrete probability measure, then exact probability evaluations are available for any query, and the interval defined by $\mathcal{P}_*$ and $\mathcal{P}^*$ always has length 0.

Example 4.3 *(a program not containing fuzzy information and with a discrete probability measure):*

Consider the program of example 4.2, with the unit clauses declaring statistical knowledge replaced by the clauses below:

$\mathcal{P}(\{a\}, 0.1).$
$\mathcal{P}(\{c\}, 0.1).$
$\mathcal{P}(\{e\}, 0.2).$
$\mathcal{P}(\{d\}, 0.2).$
$\mathcal{P}(\{b\}, 0.2).$
$\mathcal{P}(\{f\}, 0.2).$

And the queries:

$\mathcal{P}^*([x], p_6(x), \rho^*).$
$\mathcal{P}^*([x], p_6(x)|p_2(x), \rho^*).$

These queries are evaluated as follows:

- $\mathcal{P}^*([x], p_6(x), \rho^*).$*:*

  Making $\psi = p_6(x)$*, we have that* $K^S_\psi = K_\psi = \{d/1.0\}$*.*

  $\rho^* = \mathcal{P}(\{d\}) = 0.2.$

  Notice that, since every event is measurable, the upper and lower bounds for probabilities always coincide.

- $\mathcal{P}^*([x], p_6(x)|p_2(x), \rho^*).$*:*

  For $\psi = (p_6(x) \wedge p_2(x))$ *we have that:*

  $K^S_\psi = \{d/1.0\}$

$max_\wedge = min'_\wedge = 0.2$

For $\psi = (\neg p_6(x) \wedge p_2(x))$ *we have:*

$K^S_\psi = \{a/1.0, c/1.0, e/1.0\}$

$min^\neg_\wedge = 0.4$

This gives us:

$\rho^* = \frac{0.2}{0.2 + 0.4} = 0.333$

Observe that this result coincides with the classical definition of conditional probability:

$\mathcal{P}([x], (p_6(x), p_2(x)), 0.2).$

$\mathcal{P}([x], p_2(x), 0.6).$

$\mathcal{P}^*([x], p_6(x)|p_2(x), \rho^*), \rho^* = \frac{0.2}{0.6} = 0.333.$

Anecdotal Example 4.1 *Given the situation described at the anecdotal example 3.1, the Department of Physical Education wants to know whether it is necessary to maintain a team of paramedics available during the aerobic exercise sessions because of the practitioners from the Department of Artificial Intelligence. A team of paramedics is considered necessary if the probability of having someone who is susceptible to dehydration doing exercises is high. A survey revealed that from 100 times that someone from the Department of Artificial Intelligence went to do aerobics, 70 times this person was a student, 10 times s/he was a lecturer and 20 times s/he was another member of the staff. It is known that:*

- *Dave and Jane are lecturers,*
- *Carla, Flavio, Ian and Robert are students, and*
- *Andy and Joanne are members of the staff.*

What is the probability of having someone who is susceptible to dehydration?

We can calculate approximations from below and from above to this value with the queries

- $\mathcal{P}_*([x], (deptai(x), dehyd(x)), \rho_*)$ *and*
- $\mathcal{P}^*([x], (deptai(x), dehyd(x)), \rho^*)$.

We know that (see anecdotal example 3.1)

- $K_\psi^S = \{andy/0.2, flavio/0.4, ian/0.6, jane/0.1, joanne/0.1, robert/0.3\}$

We define a partition on the domain D of our program, such that we have the set

- $D_1 = \{andy, carla, dave, flavio, ian, jane, joanne, robert\}$

as a member of the partition. We also define the algebra which has as basis the set

- $\chi'_{D1} = \{\{dave, jane\}, \{carla, flavio, ian, robert\}, \{andy, joanne\}\}$

and the probability measure defined by:

- $\mathcal{P}(\{dave, jane\}, 0.1)$.
- $\mathcal{P}(\{carla, flavio, ian, robert\}, 0.7)$.
- $\mathcal{P}(\{andy, joanne\}, 0.2)$.

From these values, we have:

- $\rho_* = 0.2 \times min\{\mu(andy), \mu(joanne)\} = 0.2 \times min\{0.2, 0.1\} = 0.02$.
- $\rho^* = 0.1 \times max\{\mu(jane)\} + 0.7 \times max\{\mu(flavio), \mu(ian), \mu(robert)\} + 0.2 \times max\{\mu(andy), \mu(joanne)\} = 0.1 \times 0.1 + 0.7 \times max\{0.4, 0.6, 0.3\} + 0.2 \times max\{0.2, 0.1\} = 0.47$.

Hence, if we loosely rephrase "high probability" as "probability higher than 0.5", we conclude that there is no need of paramedics during aerobics sessions due to folk from the Department of Artificial Intelligence.

5

A Language Supporting Degrees of Belief

5.1 Adding Possible Worlds

The concept of *possible worlds* has been evoked frequently as a useful device to aid modelling uncertainty (see for example [Bac90c, Bac90b, Bun85, Bun90, CdSB90, CdSB91, FH89b, FHM90, Hal90, KJ90, Nil86, Rus87, Rus90a, Rus89]). The general idea is the assumption that there is a collection of *worlds* (or *states*, or *interpretations*), each of them assigning different truth-values to the formulae in our language. Intuitively, a possible world should be viewed as a conceivable hypothetical scenario upon which we can construct our reasoning.

Given a program P and a set of possible worlds $\Omega = \{\omega_1, ...\}$, a *rigid formula* is a formula which is always assigned the same truth-value in all possible worlds.

We assume in our language that, given a program P, each possible world ω_i assigns a different fuzzy truth-value to the set of unit clauses in P. We assume that the other clauses occurring in P, i.e. the logical dependency and statistical relations expressed in P, are rigid.

Ideally, we should keep track of every possible world independently, and repeatedly apply the machinery presented in the previous chapters for each of them each time we activated P with a query ψ. This procedure becomes computationally intractable as the size of Ω gets bigger[1]. Alternatively, we should be able to calculate singular truth-values like the minimum and the maximum values occurring in Ω for each clause: given a program P with unit clauses of the form $\mathcal{T}_*(C_i, \tau_{*i})$ and $\mathcal{T}^*(C_i, \tau_i^*)$ (representing minimum and maximum truth-degrees, respectively), we should be able to derive the values $\mathcal{T}_*(\psi, \tau_*)$ and $\mathcal{T}^*(\psi, \tau^*)$ for a query ψ.

It is not possible, however, to obtain these values for any query given only the singular values for the unit clauses, as the example below shows:

Example 5.1 *Consider the following program:*

$r(a) \leftarrow p(a), q(a).$
$s(a) \leftarrow p(a).$
$s(a) \leftarrow q(a).$

$\mathcal{T}_*(p(a), 0.2).$ $\mathcal{T}^*(p(a), 0.8).$
$\mathcal{T}_*(q(a), 0.3).$ $\mathcal{T}^*(q(a), 0.6).$

And assume that the truth-degrees have come from the possible worlds in Ω, according to Table 5.1 (values underlined):

$\Omega = \{\omega_1, \omega_2, \omega_3\}$

	ω_1	ω_2	ω_3
$\mathcal{T}(p(a))$	$\underline{0.2}$	0.5	$\underline{0.8}$
$\mathcal{T}(q(a))$	0.4	$\underline{0.6}$	$\underline{0.3}$

Table 5.1: Truth-degrees in Ω

Using the procedure introduced in figure 3.2 for each possible world separately, the results in Table 5.2 follow:

which indicate that (values underlined, table 5.2):

[1]notice that Ω is not even required to be finite. Obviously it would not be possible to keep track computationally of an infinite set of possible worlds

	ω_1	ω_2	ω_3
$\mathcal{T}(r(a))$	$\underline{0.2}$	$\underline{0.5}$	0.3
$\mathcal{T}(s(a))$	$\underline{0.4}$	0.6	$\underline{0.8}$

Table 5.2: Derived Truth-degrees in Ω

$\mathcal{T}_*(r(a), 0.2)$.
$\mathcal{T}^*(r(a), 0.5)$.
$\mathcal{T}_*(s(a), 0.4)$.
$\mathcal{T}^*(s(a), 0.8)$.

Observe that both $\mathcal{T}^*(r(a), 0.5)$ and $\mathcal{T}_*(s(a), 0.4)$ cannot be derived from the initial program unless we insert additional knowledge into the system.

Approximate solutions can be obtained for $\mathcal{T}_*$ and $\mathcal{T}^*$, i.e. we can obtain the values $\hat{\mathcal{T}}_*$ and $\hat{\mathcal{T}}^*$, such that $\hat{\mathcal{T}}_*(\psi) \leq \mathcal{T}_*(\psi)$ and $\hat{\mathcal{T}}^*(\psi) \geq \mathcal{T}^*(\psi)$ for any query ψ[2]

It is not difficult to verify that the following recursive rules satisfy these conditions:

$$\begin{array}{rcl}
r \leftarrow p & \Rightarrow & \hat{\mathcal{T}}_*(r) = \hat{\mathcal{T}}_*(p) \\
 & & \hat{\mathcal{T}}^*(r) = \hat{\mathcal{T}}^*(p) \\[1em]
r \leftarrow P, Q & \Rightarrow & \hat{\mathcal{T}}_*(r) = min\{\hat{\mathcal{T}}_*(P), \hat{\mathcal{T}}_*(Q)\} \\
 & & \hat{\mathcal{T}}^*(r) = min\{\hat{\mathcal{T}}^*(P), \hat{\mathcal{T}}^*(Q)\} \\[1em]
r \leftarrow P & \Rightarrow & \hat{\mathcal{T}}_*(r) = max\{\hat{\mathcal{T}}_*(P), \hat{\mathcal{T}}_*(Q)\} \\
r \leftarrow Q & & \hat{\mathcal{T}}^*(r) = max\{\hat{\mathcal{T}}^*(P), \hat{\mathcal{T}}^*(Q)\} \\[1em]
r \leftarrow \neg p & \Rightarrow & \hat{\mathcal{T}}_*(r) = 1 - \hat{\mathcal{T}}^*(p) \\
 & & \hat{\mathcal{T}}^*(r) = 1 - \hat{\mathcal{T}}_*(p)
\end{array}$$

where $p, q, r, \ldots$ denote atoms and $P, Q, R, \ldots$ denote conjunctions of literals.

When applied to the example above, these rules give:

$\hat{\mathcal{T}}_*(r(a), 0.2)$.

[2] there is a slight abuse of notation here: when we say that $\hat{\mathcal{T}}_*(\psi) \leq \mathcal{T}_*(\psi)$ for any query ψ, we mean that, if we have that $\hat{\mathcal{T}}_*(\psi, \hat{\tau}_*)$ and $\mathcal{T}_*(\psi, \tau_*)$, then $\hat{\tau}_* \leq \tau_*$. Symmetrically, when we say that $\hat{\mathcal{T}}^*(\psi) \geq \mathcal{T}^*(\psi)$ for any query ψ, we mean that, if we have that $\hat{\mathcal{T}}^*(\psi, \hat{\tau}^*)$ and $\mathcal{T}^*(\psi, \tau^*)$, then $\hat{\tau}^* \geq \tau^*$.

$\hat{T}^*(r(a), 0.6)$.

$\hat{T}_*(s(a), 0.3)$.

$\hat{T}^*(s(a), 0.8)$.

which obey the desired inequality conditions.

5.2 Reasoning with Possible Worlds

The *e-SLDNF* procedure and the completion *e-Comp*(P) presented in chapter 3 must be changed to accommodate the bounds for the truth-degrees across possible worlds. The *completion* of P is redefined as $*$-*Comp*(P) as presented in figure 5.1.

A model for a program P is any interpretation for which every expression occurring in $*$-*Comp*(P) has a value $\tau^* > 0$.

In order to redefine the inference procedure as $*$-*SLDNF* we need the following in our notation for the success and finite failure set:

- R_*, R^* :

  $\psi R_*(\psi, \tau_*)$ holds iff $*$-*SLDNF* succeeds, assigning τ_* to ψ as a lower bound for its truth-degree;

  $\psi R^*(\psi, \tau^*)$ holds iff $*$-*SLDNF* succeeds, assigning τ^* to ψ as an upper bound for its truth-degree.

- F_*, F^* :

  ψF_* holds iff $*$-*SLDNF* fails, assigning $\tau_* = 0$ to ψ;

  ψF^* holds iff $*$-*SLDNF* fails, assigning $\tau^* = 0$ to ψ.

$*$-*SLDNF* is defined inductively as presented in figure 5.2:

- Rules:

1. $$\frac{Def_p = \{\}}{\begin{array}{l}\forall \vec{x}[\hat{\mathcal{T}}_*(\neg p(\vec{x}), 1)] \\ \forall \vec{x}[\hat{\mathcal{T}}^*(\neg p(\vec{x}), 1)]\end{array}}$$

2. $$\frac{Def_p = \{p(\vec{t_i}) \leftarrow \psi_i : i = 1, ..., k\} \neq \{\}}{\begin{array}{l}\forall \vec{x}[\hat{\mathcal{T}}_*(p(\vec{x}), \tau_*) \leftrightarrow max\{\tau_{*i} : (\vec{x} = \vec{t_i}) \wedge [(\psi_i \neq \{\}, \hat{\mathcal{T}}_*(\psi_i, \tau_{*i}))\vee \\ (\psi_i = \{\}, \hat{\mathcal{T}}_*(p(\vec{t_i}), \tau_{*i}))]\} = \tau_*] \\ \\ \forall \vec{x}[\hat{\mathcal{T}}^*(p(\vec{x}), \tau^*) \leftrightarrow max\{\tau_i^* : (\vec{x} = \vec{t_i}) \wedge [(\psi_i \neq \{\}, \hat{\mathcal{T}}^*(\psi_i, \tau_i^*)) \\ \vee(\psi_i = \{\}, \hat{\mathcal{T}}^*(p(\vec{t_i}), \tau_i^*))]\} = \tau^*]\end{array}}$$

   where

   (a) Def_p is the set of clauses in P with p in the head;
   (b) $\vec{x}, \vec{t_i}$ are tuples of variables ($[x_1, ..., x_m]$) and terms ($[t_{1i}, ..., t_{mi}]$), respectively;
   (c) $\vec{x} = \vec{t_i}$ stands for $x_1 = t_{1i} \wedge ... \wedge x_m = t_{mi}$;
   (d) the scope of the existential quantifier are the variables occurring in the bodies of the clauses in Def_p;
   (e) ψ_i are (possibly empty) conjunctions of literals;
   (f) the connective $\leftrightarrow$ stands for equivalence.

- Axioms:

  – same as in 2.6.

Figure 5.1: *-$Comp(P)$

1. (a) $true\ R_*\ (yes, 1)$.

   (b) $true\ R^*\ (yes, 1)$.

2. (a) $$\frac{(q,\delta), max \left\{ \begin{array}{ll} \tau_{*i}: & [p_i \leftarrow \psi_i], \sigma_i = mgu(q,p_i), (\psi_i,\delta)\sigma_i\ R_*\ (\pi_i, \tau_{*i}) \vee \\ & \sigma_i = mgu(q,p_i), (\delta)\sigma_i\ R_*\ (\pi_i, \tau_i'), \\ & \hat{\mathcal{T}}_*((p_i)\sigma_i, \tau_i''), min\{\tau_i', \tau_i''\} = \tau_{*i} \end{array} \right\} = \tau_*}{(q,\delta)\ R_*\ (\sigma\pi, \tau_*)}$$

   (b) $$\frac{(q,\delta), max \left\{ \begin{array}{ll} \tau_i^*: & [p_i \leftarrow \psi_i], \sigma_i = mgu(q,p_i), (\psi_i,\delta)\sigma_i\ R^*\ (\pi_i, \tau_i^*) \vee \\ & \sigma_i = mgu(q,p_i), (\delta)\sigma_i\ R^*\ (\pi_i, \tau_i'), \\ & \hat{\mathcal{T}}^*((p_i)\sigma_i, \tau_i''), min\{\tau_i', \tau_i''\} = \tau_i^* \end{array} \right\} = \tau^*}{(q,\delta)\ R^*\ (\sigma\pi, \tau^*)}$$

   where $\sigma\pi$ are the substitutions that generate τ_* and τ^* and the ψ_i are non-empty conjunctions.

   (a) $$\frac{(\neg g,\delta), g\ R^*\ (yes, \tau'), \tau' < 1, \delta\ R_*\ (\sigma, \tau''), min\{(1-\tau'), \tau''\} = \tau_*}{(\neg g,\delta)\ R_*\ (\sigma, \tau_*)}$$

   (b) $$\frac{(\neg g,\delta), g\ R_*\ (yes, \tau'), \tau' < 1, \delta\ R^*\ (\sigma, \tau''), min\{(1-\tau'), \tau''\} = \tau^*}{(\neg g,\delta)\ R^*\ (\sigma, \tau^*)}$$

3. (a) $$\frac{(\neg g,\delta), gF^*, \delta\ R_*\ (\sigma, \tau_*)}{(\neg g,\delta)\ R_*\ (\sigma, \tau_*)}$$

   (b) $$\frac{(\neg g,\delta), gF_*, \delta\ R^*\ (\sigma, \tau^*)}{(\neg g,\delta)\ R^*\ (\sigma, \tau^*)}$$

4. (a) $$\frac{(q,\delta), \neg\exists[p \leftarrow \psi] : \exists mgu(q,p)}{(q,\delta)\ F_*}$$

   (b) $$\frac{(q,\delta), \neg\exists[p \leftarrow \psi] : \exists mgu(q,p)}{(q,\delta)\ F^*}$$

   (a) $$\frac{(q,\delta), \forall[p_i \leftarrow \psi_i] : \exists\sigma = mgu(q,p_i) \Rightarrow (\psi_i,\delta)\sigma\ F_*}{(q,\delta)\ F_*}$$

   (b) $$\frac{(q,\delta), \forall[p_i \leftarrow \psi_i] : \exists\sigma = mgu(q,p_i) \Rightarrow (\psi_i,\delta)\sigma\ F^*}{(q,\delta)\ F^*}$$

5. (a) $$\frac{(\neg g,\delta), g\ R^*\ (yes, 1)}{(\neg g,\delta)\ F_*}$$

   (b) $$\frac{(\neg g,\delta), g\ R_*\ (yes, 1)}{(\neg g,\delta)\ F^*}$$

Figure 5.2: **-SLDNF*

Example 5.2 *Consider the program introduced in example 3.1, with the unit clauses declaring truth-degrees as follows:*

$\hat{\mathcal{T}}_*(p_2(a), 0.2).$ $\hat{\mathcal{T}}^*(p_2(a), 0.4).$
$\hat{\mathcal{T}}_*(p_3(a), 0.15).$ $\hat{\mathcal{T}}^*(p_3(a), 0.2).$
$\hat{\mathcal{T}}_*(p_3(b), 0.05).$ $\hat{\mathcal{T}}^*(p_3(b), 0.1).$

And the queries:

$p_6(a).$
$p_7(b).$

The complete and/or-trees for these queries, computing bounds for the truth-degrees of $p_6(a)$ *and* $p_7(b)$*, are as presented in figures 5.3 and 5.4.*

This language subsumes the one presented in the previous chapter, having that language as the particular class of programs in which all truth-degrees are rigid.

Example 5.3 *(all truth-degrees are rigid)*

Consider the program of example 3.1, with the unit clauses declaring truth-degrees as follows:

$\hat{\mathcal{T}}_*(p_2(a), 0.4).$ $\hat{\mathcal{T}}^*(p_2(a), 0.4).$
$\hat{\mathcal{T}}_*(p_3(a), 0.2).$ $\hat{\mathcal{T}}^*(p_3(a), 0.2).$
$\hat{\mathcal{T}}_*(p_3(b), 0.1).$ $\hat{\mathcal{T}}^*(p_3(b), 0.1).$

And the queries:

$p_6(a).$
$p_7(b).$

The complete and/or-trees for these queries are as presented in figures 5.5 and 5.6.

5.3 Probabilities on the Domain with Possible Worlds

Probability evaluations must take into account the bounds for truth-degrees across possible worlds. The syntax of the language can be as before for declaring probabilities, but the evaluation procedure must be changed as follows:

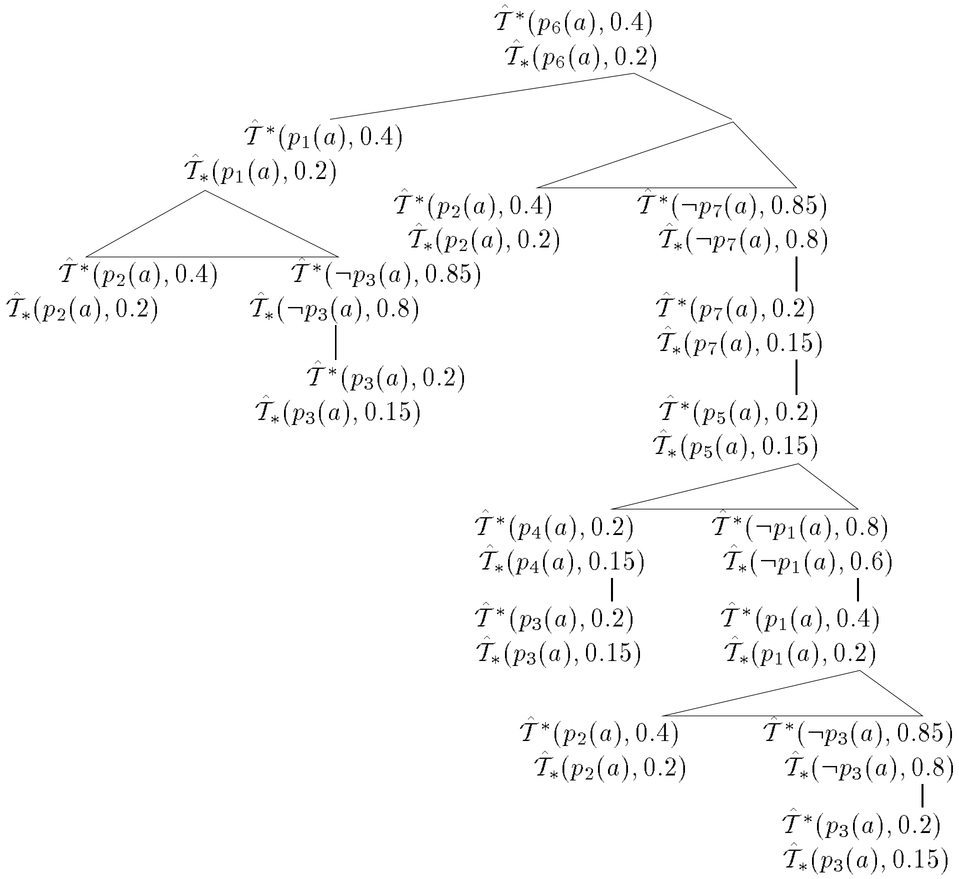

Figure 5.3: and/or-tree for $p_6(a)$

- $\mathcal{P}_*(S, \psi, \rho_*)$ and $\mathcal{P}^*(S, \psi, \rho^*)$:
  - $\mathcal{P}_*(S, \psi, \rho_*)$:
    1. generate $K_{\psi *}$, the finite fuzzy set of tuples of instances of the free variables in ψ which associate a non-zero lower bound for the truth-degree τ_* to ψ.

       If ψ does not have free variables, $K_{\psi *}$ is the singleton set containing the tuple of terms occurring in ψ with its respective bound for the truth-degree.

       If ψ contains free variables, then $K_{\psi *}$ is generated by substituting exhaustively each free variable in ψ by elements of D and then selecting the substitutions which generate the desired bounds for truth-degrees.
    2. generate $K^S_{\psi *}$ - the projection of $K_{\psi *}$ over S: select from the tuple of free variables in ψ those which are also in S, and extract the corresponding tuples of instances from $K_{\psi *}$.
    3. (a) case 1: $K^S_{\psi *} \neq \{\}$
          i. generate the cartesian product algebra and measure of the same arity as the tuples in $K^S_{\psi *}$ assuming the elements of D to be statistically independent.
          ii. generate ρ_*:

              $\rho_* = \sum_{\vec{S}_i \subseteq K^S_{\psi *}} \mathcal{P}_m(\vec{S}_i) \times min\{\tau_* : \mu_*(\vec{K}) = \tau_*, \vec{K} \in \vec{S}_i \cap K^S_{\psi *}\}, \vec{S}_i \in \vec{\chi}'_D$

       (b) case 2: $K^S_{\psi *} = \{\}$:

          make $\rho_* = \tau_*$, where $\hat{\mathcal{T}}_*(\psi, \tau_*)$.
  - $\mathcal{P}^*(S, \psi, \rho^*)$:
    1. generate K^*_{ψ}, the finite fuzzy set of tuples of instances of the free variables in ψ which associate a non-zero upper bound for the truth-degree τ^* to ψ.

       If ψ does not have free variables, K^*_{ψ} is the singleton set containing the tuple of terms occurring in ψ with its respective bound for the truth-degree.

If ψ contains free variables, then K^*_ψ is generated by substituting exhaustively each free variable in ψ by elements of D and then selecting the substitutions which generate the desired bounds for truth-degrees.

2. generate K^{*S}_ψ - the projection of K^*_ψ over S: select from the tuple of free variables in ψ those which are also in S, and extract the corresponding tuples of instances from K^*_ψ.

3. (a) case 1: $K^{*S}_\psi \neq \{\}$

   i. generate the cartesian product algebra and measure of the same arity as the tuples in K^{*S}_ψ assuming the elements of D to be statistically independent.

   ii. generate ρ^*:

   $\rho^* = \sum_{\vec{S}_i \cap K^{*S}_\psi \neq \{\}} \mathcal{P}_m(\vec{S}_i) \times max\{\tau^* : \mu^*(\vec{K}) = \tau^*, \vec{K} \in \vec{S}_i \cap K^{*S}_\psi\}, \vec{S}_i \in \vec{\chi}'_D$

   (b) case 2: $K^{*S}_\psi = \{\}$:

   make $\rho^* = \tau^*$, where $\hat{\mathcal{T}}^*(\psi, \tau^*)$.

- $\mathcal{P}_*(S, \psi_1|\psi_2, \rho_*)$ and $\mathcal{P}^*(S, \psi_1|\psi_2, \rho^*)$:

1. generate $K^{*S}_{(\psi_1,\psi_2)}$, $K^{*S}_{(\neg\psi_1,\psi_2)}$, $K^{S}_{(\psi_1,\psi_2)*}$ and $K^{S}_{(\neg\psi_1,\psi_2)*}$.

2. generate $min_\wedge, min'_\wedge, min^\neg_\wedge, max_\wedge, max'_\wedge$ and $max^\neg_\wedge$:

   – $min_\wedge = \sum_{\vec{S}_i \subseteq K^S_{(\psi_1,\psi_2)*}} \mathcal{P}_m(\vec{S}_i) \times min\{\tau_* : \mu_*(\vec{K}) = \tau_*, \vec{K} \in \vec{S}_i \cap K^S_{(\psi_1,\psi_2)*}\}, \vec{S}_i \in \vec{\chi}'_D$

   – $min'_\wedge = \sum_{\vec{S}_i \cap K^{*S}_{(\psi_1,\psi_2)} \neq \{\}} \mathcal{P}_m(\vec{S}_i) \times min\{\tau_* : \mu_*(\vec{K}) = \tau_*, \vec{K} \in \vec{S}_i \cap K^{*S}_{(\psi_1,\psi_2)}\}, \vec{S}_i \in \vec{\chi}'_D$

   – $min^\neg_\wedge = \sum_{\vec{S}_i \subseteq K^S_{(\neg\psi_1,\psi_2)*}} \mathcal{P}_m(\vec{S}_i) \times min\{\tau_* : \mu_*(\vec{K}) = \tau_*, \vec{K} \in \vec{S}_i \cap K^S_{(\neg\psi_1,\psi_2)*}\}, \vec{S}_i \in \vec{\chi}'_D$

   – $max_\wedge = \sum_{\vec{S}_i \cap K^{*S}_{(\psi_1,\psi_2)} \neq \{\}} \mathcal{P}_m(\vec{S}_i) \times max\{\tau^* : \mu^*(\vec{K}) = \tau^*, \vec{K} \in \vec{S}_i \cap K^{*S}_{(\psi_1,\psi_2)}\}, \vec{S}_i \in \vec{\chi}'_D$

   – $max'_\wedge = \sum_{\vec{S}_i \subseteq K^S_{(\psi_1,\psi_2)*}} \mathcal{P}_m(\vec{S}_i) \times max\{\tau^* : \mu^*(\vec{K}) = \tau^*, \vec{K} \in \vec{S}_i \cap K^S_{(\psi_1,\psi_2)*}\}, \vec{S}_i \in \vec{\chi}'_D$

   – $max^\neg_\wedge = \sum_{\vec{S}_i \cap K^{*S}_{(\neg\psi_1,\psi_2)} \neq \{\}} \mathcal{P}_m(\vec{S}_i) \times max\{\tau^* : \mu^*(\vec{K}) = \tau^*, \vec{K} \in \vec{S}_i \cap K^{*S}_{(\neg\psi_1,\psi_2)}\}, \vec{S}_i \in \vec{\chi}'_D$

3. generate ρ_* and ρ^*:

- $\rho_* = \begin{cases} \dfrac{min_\wedge}{max'_\wedge + max^\neg_\wedge}, & max'_\wedge + max^\neg_\wedge \neq 0 \\ 0, & max'_\wedge + max^\neg_\wedge = 0 \end{cases}$

- $\rho^* = \begin{cases} min\{1, \dfrac{max_\wedge}{min'_\wedge + min^\neg_\wedge}\}, & min'_\wedge + min^\neg_\wedge \neq 0 \\ 0, & min'_\wedge + min^\neg_\wedge = 0 \end{cases}$

Example 5.4 *Consider the program of example 4.1, with the unit clauses declaring truth-degrees as follows:*

$\hat{\mathcal{T}}_*(p_2(a), 0.2).$ $\hat{\mathcal{T}}^*(p_2(a), 0.4).$
$\hat{\mathcal{T}}_*(p_2(c), 0.3).$ $\hat{\mathcal{T}}^*(p_2(c), 0.7).$
$\hat{\mathcal{T}}_*(p_2(d), 0.1).$ $\hat{\mathcal{T}}^*(p_2(d), 0.2).$
$\hat{\mathcal{T}}_*(p_2(e), 0.2).$ $\hat{\mathcal{T}}^*(p_2(e), 0.5).$
$\hat{\mathcal{T}}^*(p_2(f), 0.1).$
$\hat{\mathcal{T}}_*(p_3(a), 0.1).$ $\hat{\mathcal{T}}^*(p_3(a), 0.2).$
$\hat{\mathcal{T}}_*(p_3(d), 0.1).$ $\hat{\mathcal{T}}^*(p_3(d), 0.2).$
$\hat{\mathcal{T}}_*(p_3(c), 0.4).$ $\hat{\mathcal{T}}^*(p_3(c), 0.8).$
$\hat{\mathcal{T}}_*(p_3(f), 0.2).$ $\hat{\mathcal{T}}^*(p_3(f), 0.4).$
$\hat{\mathcal{T}}^*(p_3(b), 0.1).$

And the queries:

$\mathcal{P}_*([x], p_6(x), \rho_*).$
$\mathcal{P}_*([x], p_6(x)|p_7(x), \rho_*).$

These queries are evaluated as follows:

- $\mathcal{P}_*([x], p_6(x), \rho_*).$*:*

  Making $\psi = p_6(x)$*, we have, by repeated application of the resolution procedure presented in figure 5.1, that*

  $K^S_{\psi*} = \{a/0.2, c/0.2, d/0.1, e/0.2\}.$

  Hence,

  $\rho_* = 0.5 \times min\{\mu(a)\} = 0.5 \times 0.2 = 0.1.$

- $\mathcal{P}_*([x], p_6(x)|p_7(x), \rho_*).$*:*

  For $\psi = (p_6(x) \wedge p_7(x))$ *we have that:*

$K^{S}_{\psi *} = \{a/0.1, c/0.2, d/0.1\}$

$K^{*S}_{\psi} = \{a/0.2, c/0.6, d/0.2, f/0.1\}$

$min_{\wedge} = 0.1 \times 0.5 = 0.05.$

$max'_{\wedge} = 0.6 \times 0.5 + 0.1 \times 0.3 = 0.33.$

For $\psi = (\neg p_6(x) \wedge p_7(x))$ *we have:*

$K^{*S}_{\psi} = \{a/0.2, c/0.8, d/0.2, b/0.1, f/0.4\}$

$max^{\neg}_{\wedge} = 0.8 \times 0.5 + 0.2 \times 0.2 + 0.4 \times 0.3 = 0.56.$

Hence:

$$\rho_* = \frac{min_{\wedge}}{max'_{\wedge} + max^{\neg}_{\wedge}} = \frac{0.05}{0.33 + 0.56} = 0.056$$

Example 5.5 *(all truth-degrees are rigid)*

Consider the program of example 5.4, with the following unit clauses declaring truth-degrees:

$\hat{\mathcal{T}}_*(p_2(a), 0.4).$ $\hat{\mathcal{T}}^*(p_2(a), 0.4).$
$\hat{\mathcal{T}}_*(p_2(c), 0.7).$ $\hat{\mathcal{T}}^*(p_2(c), 0.7).$
$\hat{\mathcal{T}}_*(p_2(d), 0.2).$ $\hat{\mathcal{T}}^*(p_2(d), 0.2).$
$\hat{\mathcal{T}}_*(p_2(e), 0.5).$ $\hat{\mathcal{T}}^*(p_2(e), 0.5).$
$\hat{\mathcal{T}}_*(p_2(f), 0.1).$ $\hat{\mathcal{T}}^*(p_2(f), 0.1).$
$\hat{\mathcal{T}}_*(p_3(a), 0.2).$ $\hat{\mathcal{T}}^*(p_3(a), 0.2).$
$\hat{\mathcal{T}}_*(p_3(b), 0.1).$ $\hat{\mathcal{T}}^*(p_3(b), 0.1).$
$\hat{\mathcal{T}}_*(p_3(d), 0.2).$ $\hat{\mathcal{T}}^*(p_3(d), 0.2).$
$\hat{\mathcal{T}}_*(p_3(c), 0.8).$ $\hat{\mathcal{T}}^*(p_3(c), 0.8).$
$\hat{\mathcal{T}}_*(p_3(f), 0.4).$ $\hat{\mathcal{T}}^*(p_3(f), 0.4).$

And the queries:

$\mathcal{P}_*([x], p_6(x), \rho_*).$
$\mathcal{P}_*([x], p_6(x)|p_7(x), \rho_*).$

These queries are evaluated as follows:

- $\mathcal{P}_*([x], p_6(x), \rho_*).$*:*

  Making $\psi = p_6(x)$*, we have that*

$K^S_{\psi *} = \{a/0.4, c/0.2, d/0.2, e/0.5, f/0.1\}$.

$\rho_* = 0.5 \times 0.2 + 0.3 \times 0.1 = 0.13$.

- $\mathcal{P}_*([x], p_6(x)|p_7(x), \rho_*)$.:

  For $\psi = (p_6(x) \wedge p_7(x))$ *we have that:*

  $K^S_{\psi *} = K^{*S}_{\psi} = \{a/0.2, c/0.2, d/0.2, f/0.1\}$

  $min_\wedge = 0.13$.

  $max'_\wedge = 0.13$.

  For $\psi = (\neg p_6(x) \wedge p_7(x))$ *we have:*

  $K^S_{\psi *} = K^{*S}_{\psi} = \{a/0.2, c/0.8, d/0.2, b/0.1, f/0.4\}$

  $max^{\neg}_\wedge = 0.56$.

  Hence:

  $$\rho_* = \frac{min_\wedge}{max'_\wedge + max^{\neg}_\wedge} = 0.188.$$

Example 5.6 *(a program containing only non-fuzzy rigid information and with a discrete probability measure, in which every query admits an exact probability evaluation)*

Consider the program of example 4.3, with the truth-degrees declared as follows:

$\hat{\mathcal{T}}_*(p_2(a), 1.0)$. $\hat{\mathcal{T}}^*(p_2(a), 1.0)$.
$\hat{\mathcal{T}}_*(p_2(c), 1.0)$. $\hat{\mathcal{T}}^*(p_2(c), 1.0)$.
$\hat{\mathcal{T}}_*(p_2(d), 1.0)$. $\hat{\mathcal{T}}^*(p_2(d), 1.0)$.
$\hat{\mathcal{T}}_*(p_2(e), 1.0)$. $\hat{\mathcal{T}}^*(p_2(e), 1.0)$.
$\hat{\mathcal{T}}_*(p_3(a), 1.0)$. $\hat{\mathcal{T}}^*(p_3(a), 1.0)$.
$\hat{\mathcal{T}}_*(p_3(b), 1.0)$. $\hat{\mathcal{T}}^*(p_3(b), 1.0)$.
$\hat{\mathcal{T}}_*(p_3(d), 1.0)$. $\hat{\mathcal{T}}^*(p_3(d), 1.0)$.
$\hat{\mathcal{T}}_*(p_3(c), 1.0)$. $\hat{\mathcal{T}}^*(p_3(c), 1.0)$.
$\hat{\mathcal{T}}_*(p_3(f), 1.0)$. $\hat{\mathcal{T}}^*(p_3(f), 1.0)$.

And the queries:

$\mathcal{P}_*([x], p_6(x), \rho_*)$.
$\mathcal{P}_*([x], p_6(x)|p_2(x), \rho_*)$.

These queries are evaluated as follows:

- $\mathcal{P}_*([x], p_6(x), \rho_*).$:

  Making $\psi = p_6(x)$, *we have, by repeated application of resolution, that* $K^S_{\psi*} = \{e/1.0\}$.

  $\rho_* = \mathcal{P}(\{e\}) = 0.2$.

- $\mathcal{P}_*([x], p_6(x)|p_2(x), \rho_*).$:

  For $\psi = (p_6(x) \wedge p_2(x))$ *we have that:*

  $K^S_{\psi*} = K^{*S}_{\psi} = \{e/1.0\}$

  $max_\wedge = min'_\wedge = 0.2$.

  For $\psi = (\neg p_6(x) \wedge p_2(x))$ *we have:*

  $K^S_{\psi*} = K^{*S}_{\psi} = \{a/1.0, c/1.0, d/1.0\}$

  $min^\neg_\wedge = 0.4$.

  This gives us:

  $\rho^* = \dfrac{0.2}{0.2 + 0.4} = 0.333$, *which, as expected, concides with the result obtained from the classical definition of conditional probability.*

Anecdotal Example 5.1 *From the situation portrayed at the anecdotal example 4.1, we have the Department of Physical Education investigating how fit individuals from the Department of Artificial Intelligence are for engaging into aerobics sessions. There is a suspicion that Flavio is below the average fitness at this Department, and in order to evaluate this the following queries are made:*

- *is Flavio fit?*

- *what is the probability of having someone from the Department of Artificial Intelligence who is fit? (alternatively, what is the expected degree of fitness of someone from the Department of Artificial Intelligence?)*

However, before evaluating these queries it is remarked that the qualifier "variable blood pressure" is quite unstable (i.e. it varies from day to day for each individual, due to several uncontrolled factors) and that assuming a single truth-value for each individual for this qualifier is an oversimplification. Alternatively, a collection of measures is made and the extreme values are recorded. The results are:

- *Carla's "blood pressure variability" is always quite high;*
- *Kathleen's "blood pressure variability" changes a lot from day to day;*
- *Flavio's "blood pressure variability" is usually quite low, although sometimes it goes very high;*
- *Andy's "blood pressure variability" is always very low, but not always insignificant.*

Given these informations, we update the unit clauses representing truth-degrees in our programs as follows:

$$\begin{array}{ll}
\mathcal{T}_*(deptai(andy), 1). & \mathcal{T}_*(deptai(carla), 1). \\
\mathcal{T}_*(deptai(dave), 1). & \mathcal{T}_*(deptai(flavio), 1). \\
\mathcal{T}_*(deptai(ian), 1). & \mathcal{T}_*(deptai(jane), 1). \\
\mathcal{T}_*(deptai(joanne), 1). & \mathcal{T}_*(deptai(robert), 1).
\end{array}$$

$$\begin{array}{ll}
\mathcal{T}_*(blond(ian), 0.9). & \mathcal{T}_*(blond(joanne), 0.9). \\
\mathcal{T}_*(blond(kathleen), 0.7). & \mathcal{T}_*(blond(william), 0.6). \\
\mathcal{T}_*(blond(jane), 0.7). &
\end{array}$$

$$\begin{array}{ll}
\mathcal{T}_*(parent(robert, kathleen), 1). & \mathcal{T}_*(parent(robert, william), 1). \\
\mathcal{T}_*(parent(carla, alice), 1). & \mathcal{T}_*(parent(carla, otavio), 1).
\end{array}$$

$\mathcal{T}_*(young(andy), 0.9).$ $\mathcal{T}_*(young(carla), 0.9).$
$\mathcal{T}_*(young(dave), 0.9).$ $\mathcal{T}_*(young(flavio), 0.9).$
$\mathcal{T}_*(young(ian), 0.9).$ $\mathcal{T}_*(young(jane), 0.9).$
$\mathcal{T}_*(young(joanne), 0.9).$ $\mathcal{T}_*(young(robert), 0.9).$
$\mathcal{T}_*(young(alice), 0.3).$ $\mathcal{T}_*(young(otavio), 0.3).$
$\mathcal{T}_*(young(kathleen), 0.4).$ $\mathcal{T}_*(young(william), 0.3).$

$\mathcal{T}_*(overw(flavio), 0.4).$ $\mathcal{T}_*(overw(otavio), 0.7).$
$\mathcal{T}_*(overw(ian), 0.6).$ $\mathcal{T}_*(overw(william), 0.5).$
$\mathcal{T}_*(overw(andy), 0.2).$ $\mathcal{T}_*(overw(robert), 0.3).$

$\mathcal{T}_*(varblpres(carla), 0.7).$ $\mathcal{T}_*(varblpres(kathleen), 0.2).$
$\mathcal{T}_*(varblpres(flavio), 0.3).$

$\mathcal{T}^*(deptai(andy), 1).$ $\mathcal{T}^*(deptai(carla), 1).$
$\mathcal{T}^*(deptai(dave), 1).$ $\mathcal{T}^*(deptai(flavio), 1).$
$\mathcal{T}^*(deptai(ian), 1).$ $\mathcal{T}^*(deptai(jane), 1).$
$\mathcal{T}^*(deptai(joanne), 1).$ $\mathcal{T}^*(deptai(robert), 1).$

$\mathcal{T}^*(blond(ian), 0.9).$ $\mathcal{T}^*(blond(joanne), 0.9).$
$\mathcal{T}^*(blond(kathleen), 0.7).$ $\mathcal{T}^*(blond(william), 0.6).$
$\mathcal{T}^*(blond(jane), 0.7).$

$\mathcal{T}^*(parent(robert, kathleen), 1).$ $\mathcal{T}^*(parent(robert, william), 1).$
$\mathcal{T}^*(parent(carla, alice), 1).$ $\mathcal{T}^*(parent(carla, otavio), 1).$

$\mathcal{T}^*(young(andy), 0.9).$ $\mathcal{T}^*(young(carla), 0.9).$
$\mathcal{T}^*(young(dave), 0.9).$ $\mathcal{T}^*(young(flavio), 0.9).$
$\mathcal{T}^*(young(ian), 0.9).$ $\mathcal{T}^*(young(jane), 0.9).$
$\mathcal{T}^*(young(joanne), 0.9).$ $\mathcal{T}^*(young(robert), 0.9).$
$\mathcal{T}^*(young(alice), 0.3).$ $\mathcal{T}^*(young(otavio), 0.3).$
$\mathcal{T}^*(young(kathleen), 0.4).$ $\mathcal{T}^*(young(william), 0.3).$

$\mathcal{T}^*(overw(flavio), 0.4).$ $\mathcal{T}^*(overw(otavio), 0.7).$
$\mathcal{T}^*(overw(ian), 0.6).$ $\mathcal{T}^*(overw(william), 0.5).$
$\mathcal{T}^*(overw(andy), 0.2).$ $\mathcal{T}^*(overw(robert), 0.3).$

$\mathcal{T}^*(varblpres(carla), 0.9).$ $\mathcal{T}^*(varblpres(kathleen), 0.8).$
$\mathcal{T}^*(varblpres(flavio), 0.8).$ $\mathcal{T}^*(varblpres(andy), 0.2).$

The repeated application of the procedure in figure 5.2 for the query '$(fit(x), deptai(x))$.' generates the set

$$K_\varphi^S = \{ \quad andy/[0.8, 0.8], carla/[0.1, 0.3], \\ dave/[0.9, 0.9], flavio/[0.1, 0.6], \\ ian/[0.4, 0.4], jane/[0.9, 0.9], \\ joanne/[0.9, 0.9], robert/[0.7, 0.7]\}$$

where $a/[\tau_*, \tau^*]$ iff $\mathcal{T}_*(a, \tau_*), \mathcal{T}^*(a, \tau^*)$.

From these values and the probabilities on the domain given in anecdotal example 4.1, we obtain that:

- $\mathcal{T}_*((fit(flavio), deptai(flavio)), 0.1)$.
- $\mathcal{T}^*((fit(flavio), deptai(flavio)), 0.6)$.
- $\mathcal{P}_*([x], (fit(x), deptai(x)), 0.32)$.
- $\mathcal{P}^*([x], (fit(x), deptai(x)), 0.76)$.

These results indicate that the truth-degree for how fit Flavio is is smaller than the expected lower bound for the truth-degree of how fit a generic individual is (since $0.1 < 0.32$), and also that the truth-degree for how fit Flavio is is not bigger than the expected upper bound for the truth-degree of how fit a generic individual can be (since $0.6 < 0.76$).

5.4 Adding Probabilities on Possible Worlds

Different worlds can have different likelihoods. Given a set of possible worlds Ω, we can define a *probability measure* $\mathcal{B}$ to represent these likelihoods. The *expected value* for the truth-degree of a clause ψ can be defined as $\mathcal{B}(\psi) = \int_\Omega \tau d\beta$, where $\mathcal{T}(\psi, \tau)$.

When the sets ω of possible worlds in which sentences have "non-zero" truth-degrees are measurable, this defines a straightforward extension of Nilsson's probabilistic logic [Nil86] to deal with fuzzy predicates. If we consider the non-measurable cases, then the language extends the so-called *Dempster-Shafer structures* [FH89b, CdSB90]. Dempster-Shafer structures for propositional languages are presented in [CdSB90] as tuples

$$[\Omega, \chi_\Omega, \mathcal{B}, At, \chi_{At}, i]$$

where:

- Ω is a (possibly infinite) set of possible worlds;
- χ_Ω is a σ-algebra of subsets of Ω;[3]
- $\mathcal{B}$ is a probability measure on χ_Ω[4];
- At is the set of atomic propositions of a finite propositional language[5];
- χ_{At} is the closure of At under $\wedge$ and $\neg$;
- i - identified as the *incidence mapping* - is a mapping from χ_{At} to 2^Ω as follows:

$$i : \chi_{At} \rightarrow 2^\Omega.$$

  - $i(\psi) = \{\omega \in \Omega : \psi \text{ is true in } \omega\}$.

Which gives as corollaries the following properties:

[3] a σ-algebra χ_Ω is a set of subsets of Ω such that:

- $\Omega \in \chi_\Omega$;
- $\omega \in \chi_\Omega \rightarrow \neg\omega \in \chi_\Omega$;
- $\omega_1, \omega_2, ... \in \chi_\Omega \rightarrow \bigcup_i \omega_i \in \chi_\Omega$ (where ω_i are countable).

[4] the symbol $\mathcal{B}$ is standing for both the expected truth-value of a clause across possible worlds and the probability measure that generates it. It should be clear from the context in which case each of the interpretations is the intended one

[5] a *finite propositional language* is a finite set of propositions $\Phi = \{p_1, ..., p_n\}$ and its closure under the application of the Boolean operators $\wedge$ and $\neg$. The set of *atomic propositions* of a finite propositional language is the set $At = \{\delta_1, ..., \delta_{2^n}\}$, where $\delta_i = p'_1 \wedge ... \wedge p'_n$ and p'_j is either p_j or $\neg p_j$

- $i(\{\}) = \{\}$.
- $i(\bigvee_{pj \in At} p_j) = \Omega$.
- $i(\neg\psi) = \Omega - i(\psi)$.
- $i(\psi_1 \wedge \psi_2) = i(\psi_1) \cap i(\psi_2)$.
- $i(\psi_1 \vee \psi_2) = i(\psi_1) \cup i(\psi_2)$.

The probability measure $\mathcal{B}$ can be extended to the power set 2^Ω by means of the inner and outer measures $\mathcal{B}_*$ and $\mathcal{B}^*$, where:

- $\mathcal{B}_*(\xi) = sup\{\mathcal{B}(\omega) : \omega \subseteq \xi, \omega \in \chi_\Omega\}$;
- $\mathcal{B}^*(\xi) = inf\{\mathcal{B}(\omega) : \omega \supseteq \xi, \omega \in \chi_\Omega\}$.

Inner and outer measures work as approximations for the measures of non-measurable sets of possible worlds and in [FH89b] we have the result that they correspond to Dempster-Shafer's belief and plausibility measures in a precise sense. They also contain Nilsson's probabilistic logic as the special case in which $\mathcal{B}$ is a discrete measure.

Dempster-Shafer structures are expressive enough to represent what two other important mechanisms to represent degrees of belief can represent, namely Possibilistic Logic and Incidence Calculus.

Possibilistic Logic [DP87, DP88, Som90, Rus89] assigns *degrees of necessity and possibility* to sentences, which express the extent to which these sentences are believed to be necessarily and possibly true, respectively. The semantics of possibilistic logic is defined based on a *fuzzy set of possible worlds*, i.e. a set of possible worlds together with a fuzzy membership function that describes the extent to which each possible world belongs to a referential set - the set containing the "real" world. Given a sentence ψ, the *necessity measure* $N(\psi)$ expresses the greatest lower bound for the degree of membership of the worlds that support ψ, and the *possibility measure* $\Pi(\psi)$ expresses the least upper bound for this degree of membership.

The *fundamental axiom of possibility measures* dictates that, given two sentences ψ_1 and ψ_2, $\Pi(\psi_1 \vee \psi_2) = max\{\Pi(\psi_1), \Pi(\psi_2)\}$. As a corollary, it is always true that $N(\psi_1 \wedge \psi_2) = min\{N(\psi_1), N(\psi_2)\}$. Necessity and possibility measures define intervals containing the degree of belief ascribed to a sentence being true, and are related by the following expressions:

- $\Pi(\psi) = 1 - N(\neg\psi)$;
- $N(\psi) \leq \Pi(\psi)$

Possibilistic Logic has been proved to be, "formally at least" [Sha87], a particular case of Dempster-Shafer structures in which the sets forming the algebra χ_Ω are *consonant*, i.e. they are a sequence $X_1, X_2, ...$ such that $\Omega \supseteq X_1 \supseteq X_2 \supseteq ....$

Incidence Calculus [Bun85, Bun86, Bun90] was formalised in [CdSB90, CdSB91] as the tuples

$$[\Omega, 2^\Omega, \mathcal{B}, At, \chi_{S \subseteq At}, i]$$

where:

- Ω is the set of possible worlds;
- 2^Ω is the power set of Ω;
- $\mathcal{B}$ is a probability measure on χ_Ω;
- At is the set of atomic propositions of a finite propositional language;
- $\chi_{S \subseteq At}$ is the closure under $\wedge$ and $\neg$ of a subset S of At;
- i is a mapping from $\chi_{S \subseteq At}$ to 2^Ω with the same properties as before.

There is a procedure called the *Legal Assignment Finder* which, given a proposition ψ from the language of At, identifies upper and lower bounds for the degree of belief in ψ, i.e. upper and lower limits within which $\mathcal{B}(i(\psi))$ must fall.

The interest of this formalism, when compared with e.g. Dempster-Shafer Theory of Evidence, is on the additional source of uncertainty it identifies: since *incidences* are allowed to be *partial mappings* with respect to χ_{At} (i.e. their domain is allowed to be a subset of χ_{At}), a rational agent may have to work with interval-based degrees of belief because of its inability to match propositions with possible worlds.

This characterisation of an additional source of uncertainty is a useful device to help constructing models for solving practical problems involving uncertain reasoning. However, in [CdSB90] it is proved that, at least from a formal point of view, every Incidence Calculus has a semantically equivalent Dempster-Shafer structure, i.e. for every tuple

$$IC = [\Omega, 2^{\Omega}, \mathcal{B}, At, \chi_{S \subseteq At}, i]$$

there is a tuple

$$DS = [\Omega', \chi'_{\Omega}, \mathcal{B}', At, \chi_{At}, i']$$

such that the the belief and plausibility measures in DS are the same as the upper and lower bounds in IC for every proposition from the language of At.

We extend the definition of Dempster-Shafer structures in two senses in order to make it expressive enough for our language:

- the first extension is the replacement of the *propositional language* At by the language of *normal clauses*. This is essentially a change in notation, since we are by definition dealing with finite languages (i.e. languages in which we can only generate a finite set of semantically distinct sentences), in which case we can always expand every sentence to its collection of semantically equivalent ground sentences and produce the "propositional version" of the language;

- the second extension is the addition of truth-degrees: each sentence must be allowed to assume a truth-degree from the interval $[0,1]$ in each possible world. In order to do so we redefine the incidence mapping as:

$$i : \chi_{At} \rightarrow 2^{\Omega\times[0,1]}.$$

  - $i(\psi) = \{[\omega,\tau] : \omega \in \Omega, \tau \in [0,1], \mathcal{T}(\psi,\tau), \tau > 0 \text{ in } \omega\}$.

  In which case we have the following properties:

  - $i(\{\}) = \{\}$.
  - $i(\bigvee_{pj\in At} p_j) = \{[\Omega, 1]\}$.
  - $i(\neg\psi) = \{[\omega,\tau] : \omega \in \Omega, [(\mathcal{T}(\psi,\tau'), \tau' < 1, \tau = 1-\tau')\ OR$
    $(\mathcal{T}(\psi,0), \tau = 1)]\}$.
  - $i(\psi_1 \wedge \psi_2) = \{[\omega,\tau] : \omega \in \Omega, \mathcal{T}(\psi_1,\tau_1), \mathcal{T}(\psi_2,\tau_2), min\{\tau_1,\tau_2\} = \tau\}$.
  - $i(\psi_1 \vee \psi_2) = \{[\omega,\tau] : \omega \in \Omega, [(\mathcal{T}(\psi_1,\tau), \mathcal{T}(\neg\psi_2,1))\ OR$
    $(\mathcal{T}(\neg\psi_1,1), \mathcal{T}(\psi_2,\tau))\ OR$
    $(\mathcal{T}(\psi_1,\tau_1), \mathcal{T}(\psi_2,\tau_2), max\{\tau_1,\tau_2\} = \tau)]\}$.

Given expressions of the form $\mathcal{B}_*(\psi,\beta_*)$ and $\mathcal{B}^*(\psi,\beta^*)$ for the unit clauses occurring in a program P - where β_* and β^* represent the *inner* and the *outer extensions* for the measure $\mathcal{B}$ - it is not possible to derive the degrees of belief for all queries on P, unless the statistical dependency among clauses is known [Bun85]. Nonetheless, *bounds* can be derived for these degrees of belief.

For programs without fuzzy predicates, these bounds can be defined by the following rules[6] (from [NS92]):

$$
\begin{array}{lll}
r \leftarrow p & \Rightarrow & \mathcal{B}_*(r) = \mathcal{B}_*(p) \\
 & & \mathcal{B}^*(r) = \mathcal{B}^*(p) \\[1em]
r \leftarrow P, Q & \Rightarrow & \mathcal{B}_*(r) \geq max\{0, [\mathcal{B}_*(P) + \mathcal{B}_*(Q) - 1]\} \\
 & & \mathcal{B}^*(r) \leq min\{\mathcal{B}^*(P), \mathcal{B}^*(Q)\} \\[1em]
r \leftarrow P & \Rightarrow & \mathcal{B}_*(r) \geq max\{\mathcal{B}_*(P), \mathcal{B}_*(Q)\} \\
r \leftarrow Q & & \mathcal{B}^*(r) \leq min\{1, [\mathcal{B}^*(P) + \mathcal{B}^*(Q)]\} \\[1em]
r \leftarrow \neg p & \Rightarrow & \mathcal{B}_*(r) = 1 - \mathcal{B}^*(p) \\
 & & \mathcal{B}^*(r) = 1 - \mathcal{B}_*(p)
\end{array}
$$

where $p, q, r, ...$ denote atoms and $P, Q, R, ...$ denote conjunctions of literals.

When a program contains fuzzy information, these rules can be further refined, since we have that $\hat{\mathcal{T}}_*(\psi) \leq \mathcal{B}(\psi) \leq \hat{\mathcal{T}}^*(\psi)$ for any clause ψ:

$$
\begin{array}{lll}
r \leftarrow p & \Rightarrow & \hat{\mathcal{B}}_*(r) = \hat{\mathcal{B}}_*(p) \\
 & & \hat{\mathcal{B}}^*(r) = \hat{\mathcal{B}}^*(p) \\[1em]
r \leftarrow P, Q & \Rightarrow & \mathcal{B}_*(r) \geq \hat{\mathcal{B}}_*(r) = max\{\hat{\mathcal{T}}_*(r), [\hat{\mathcal{B}}_*(P) + \hat{\mathcal{B}}_*(Q) - 1]\} \\
 & & \mathcal{B}^*(r) \leq \hat{\mathcal{B}}^*(r) = min\{\hat{\mathcal{T}}^*(r), \hat{\mathcal{B}}^*(P), \hat{\mathcal{B}}^*(Q)\} \\[1em]
r \leftarrow P & \Rightarrow & \mathcal{B}_*(r) \geq \hat{\mathcal{B}}_*(r) = max\{\hat{\mathcal{T}}_*(r), \hat{\mathcal{B}}_*(P), \hat{\mathcal{B}}_*(Q)\} \\
r \leftarrow Q & & \mathcal{B}^*(r) \leq \hat{\mathcal{B}}^*(r) = min\{\hat{\mathcal{T}}^*(r), [\hat{\mathcal{B}}^*(P) + \hat{\mathcal{B}}^*(Q)]\} \\[1em]
r \leftarrow \neg p & \Rightarrow & \hat{\mathcal{B}}_*(r) = 1 - \hat{\mathcal{B}}^*(p) \\
 & & \hat{\mathcal{B}}^*(r) = 1 - \hat{\mathcal{B}}_*(p)
\end{array}
$$

In what follows we introduce these concepts into our language.

5.5 Reasoning with Probabilities on Possible Worlds

The inference procedure defined in the previous chapters can be extended to deal with degrees of belief. We define, in addition to *$*$-SLDNF* and *$*$-Comp(P)* previously presented, the procedure and completion rules for evaluating degrees of belief in a program P presented in figures 5.7 and

[6] again, we make an abuse of notation here: when we say $\mathcal{B}_*(\psi)$ we are actually referring to a β_* such that $\mathcal{B}_*(\psi, \beta_*)$, and when we say $\mathcal{B}^*(\psi)$ we are actually referring to a β^* such that $\mathcal{B}^*(\psi, \beta^*)$

5.8, in which the following notation conventions are adopted:

- $\psi R_{\beta *}(\psi, \beta_*)$ holds iff **-SLDNF* succeeds, assigning β_* to ψ as a lower bound for its truth-degree;
- $\psi R^*_{\beta}(\psi, \beta^*)$ holds iff **-SLDNF* succeeds, assigning β^* to ψ as an upper bound for its truth-degree;
- $\psi F_{\beta *}$ holds iff **-SLDNF* fails, assigning $\beta_* = 0$ to ψ;
- ψF^*_{β} holds iff **-SLDNF* fails, assigning $\beta^* = 0$ to ψ.

Example 5.7 *Consider the program introduced in example 5.2, with the unit clauses declaring degrees of belief as follows:*

$\hat{\mathcal{B}}_*(p_2(a), 0.3)$. $\hat{\mathcal{B}}^*(p_2(a), 0.35)$.
$\hat{\mathcal{B}}_*(p_3(a), 0.17)$. $\hat{\mathcal{B}}^*(p_3(a), 0.19)$.
$\hat{\mathcal{B}}_*(p_3(b), 0.06)$. $\hat{\mathcal{B}}^*(p_3(b), 0.07)$.

And the queries:

$p_6(a)$.
$p_7(b)$.

The complete and/or-trees for these queries, showing all intermediate belief and truth values, are as follows (figures 5.9 and 5.10):

5.6 Dealing with Conditional Beliefs

We may want to constrain our queries to a specific set of possible worlds in which a statement is believed to be (to some extent) true. In other words, we may be interested in measuring *conditional beliefs* on queries.

If we had the values for the inner and outer measures $\mathcal{B}_*(\psi)$ and $\mathcal{B}^*(\psi)$, we could evaluate conditional beliefs by using the expressions given in [FH89a]:

- $\mathcal{B}_*(\psi_1|\psi_2) = \begin{cases} \dfrac{\mathcal{B}_*(\psi_1,\psi_2)}{\mathcal{B}_*(\psi_1,\psi_2)+\mathcal{B}^*(\neg\psi_1,\psi_2)}, & \mathcal{B}_*(\psi_1,\psi_2)+\mathcal{B}^*(\neg\psi_1,\psi_2) \neq 0 \\ 0, & \mathcal{B}_*(\psi_1,\psi_2)+\mathcal{B}^*(\neg\psi_1,\psi_2) = 0 \end{cases}$

- $\mathcal{B}^*(\psi_1|\psi_2) = \begin{cases} \dfrac{\mathcal{B}^*(\psi_1,\psi_2)}{\mathcal{B}^*(\psi_1,\psi_2)+\mathcal{B}_*(\neg\psi_1,\psi_2)}, & \mathcal{B}^*(\psi_1,\psi_2)+\mathcal{B}_*(\neg\psi_1,\psi_2) \neq 0 \\ 0, & \mathcal{B}^*(\psi_1,\psi_2)+\mathcal{B}_*(\neg\psi_1,\psi_2) = 0 \end{cases}$

However, we can only access the values $\hat{\mathcal{B}}_*(\psi) \leq \mathcal{B}_*(\psi)$ and $\hat{\mathcal{B}}^*(\psi) \geq \mathcal{B}^*(\psi)$ for a clause ψ. Since we have that [FH89a]:

$$[\mathcal{B}_*(\psi_1|\psi_2), \mathcal{B}^*(\psi_1|\psi_2)] \subseteq [\frac{\mathcal{B}_*(\psi_1,\psi_2)}{\mathcal{B}^*(\psi_2)}, \frac{\mathcal{B}^*(\psi_1,\psi_2)}{\mathcal{B}_*(\psi_2)}]$$

We immediately have that:

$$[\mathcal{B}_*(\psi_1|\psi_2), \mathcal{B}^*(\psi_1|\psi_2)] \subseteq [\frac{\hat{\mathcal{B}}_*(\psi_1,\psi_2)}{\hat{\mathcal{B}}^*(\psi_2)}, \frac{\hat{\mathcal{B}}^*(\psi_1,\psi_2)}{\hat{\mathcal{B}}_*(\psi_2)}]$$

Hence, we adopt these expressions as approximations for the lower and upper bounds for conditional degrees of belief:

- $\hat{\mathcal{B}}_*(\psi_1|\psi_2) = \dfrac{\hat{\mathcal{B}}_*(\psi_1,\psi_2)}{\hat{\mathcal{B}}^*(\psi_2)},$

- $\hat{\mathcal{B}}^*(\psi_1|\psi_2) = min\{1, \dfrac{\hat{\mathcal{B}}^*(\psi_1,\psi_2)}{\hat{\mathcal{B}}_*(\psi_2)}\}$

Example 5.8 *Consider the program presented in example 5.7 and the queries:*

$\hat{\mathcal{B}}_*(p_6(a)|p_7(b))$.
$\hat{\mathcal{B}}^*(p_6(a)|p_7(b))$.

We have that:

$\hat{\mathcal{B}}_*(p_6(a), p_7(b)) = 0.05$.
$\hat{\mathcal{B}}^*(p_6(a), p_7(b)) = 0.07$.

$\hat{\mathcal{B}}_*(p_7(b)) = 0.06$.
$\hat{\mathcal{B}}^*(p_7(b)) = 0.07$.

Thus, we obtain that:

$\hat{\mathcal{B}}_*(p_6(a)|p_7(b)) = 0.05/0.07 = 0.71$.
$\hat{\mathcal{B}}^*(p_6(a)|p_7(b)) = min\{1, 0.07/0.06\} = 1$.

Anecdotal Example 5.2 *From the anecdotal example 5.1, we observe that we have a better estimate for Flavio's "blood pressure variability" than just the extreme truth-degrees it can have. This estimate can be expressed as bounds for the degree of belief on the truth of the sentence "Flavio has a variable blood pressure". We can add to the program in the anecdotal example 5.1 a set of statements expressing degrees of belief on unit clauses, including the following:*

$\mathcal{B}_*(deptai(flavio), 1).$ $\mathcal{B}^*(deptai(flavio), 1).$
$\mathcal{B}_*(young(flavio), 0.9).$ $\mathcal{B}^*(young(flavio), 0.9).$
$\mathcal{B}_*(overw(flavio), 0.4).$ $\mathcal{B}^*(overw(flavio), 0.4).$
$\mathcal{B}_*(varblpres(flavio), 0.3).$ $\mathcal{B}^*(varblpres(flavio), 0.35).$

If we replace the statements expressing extreme truth-degrees on unit clauses regarding Flavio by the following:

$\mathcal{T}_*(deptai(flavio), 1).$ $\mathcal{T}^*(deptai(flavio), 1).$
$\mathcal{T}_*(young(flavio), 0.9).$ $\mathcal{T}^*(young(flavio), 0.9).$
$\mathcal{T}_*(overw(flavio), 0.4).$ $\mathcal{T}^*(overw(flavio), 0.4).$
$\mathcal{T}_*(varblpres(flavio), 0.3).$ $\mathcal{T}^*(varblpres(flavio), 0.9).$

We generate the following estimates for the bounds to the truth-degree and the degree of belief on "Flavio is fit for aerobics", based on $*$-SLDNF and β-SLDNF (see also figure 5.11):

$\mathcal{T}_*(fit(flavio), 0.1).$
$\mathcal{B}_*(fit(flavio), 0.15).$
$\mathcal{B}^*(fit(flavio), 0.6).$
$\mathcal{T}^*(fit(flavio), 0.6).$

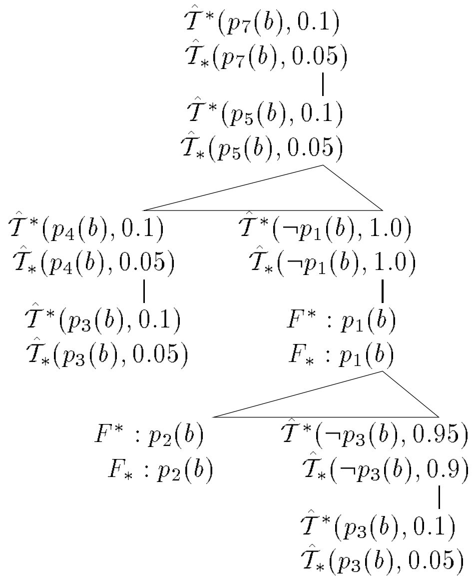

Figure 5.4: and/or-tree for $p_7(b)$

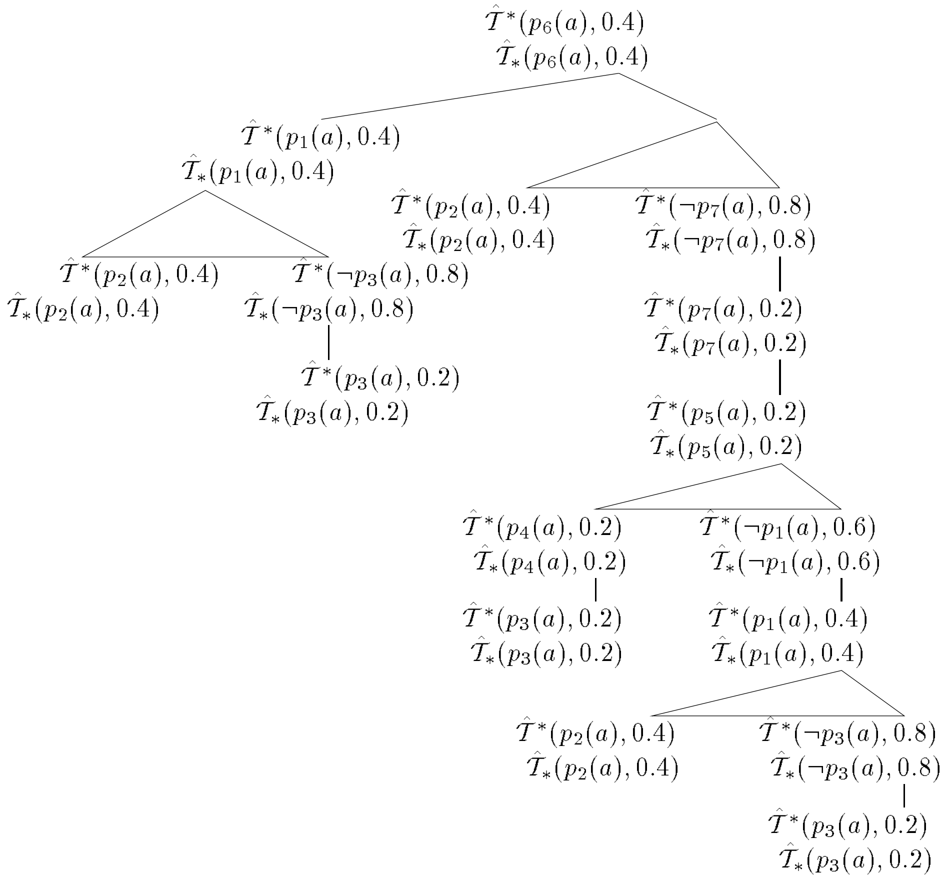

Figure 5.5: and/or-tree for $p_6(a)$ - rigid truth-degrees

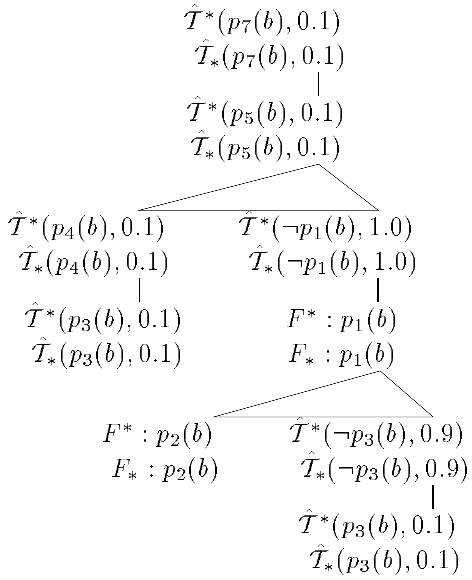

Figure 5.6: and/or-tree for $p_7(a)$ - rigid truth-degrees

- Rules:

  1. $$\frac{Def_p = \{\}}{\forall \vec{x}[\hat{\mathcal{B}}_*(\neg p(\vec{x}), 1)]}$$
     $$\forall \vec{x}[\hat{\mathcal{B}}^*(\neg p(\vec{x}), 1)]$$

  2. $$\frac{Def_p = \{p(\vec{t_i}) \leftarrow \psi_i : i = 1, ..., k\} \neq \{\}}{\forall \vec{x}[\hat{\mathcal{B}}_*(p(\vec{x}), \beta_*) \leftrightarrow max\{\beta_{*i} : (\vec{x} = \vec{t_i}) \wedge [(\psi_i \neq \{\}, \hat{\mathcal{B}}_*(\psi_i, \beta_{*i}))}$$
     $$\vee(\psi_i = \{\}, \hat{\mathcal{B}}_*(p(\vec{t_i}), \beta_{*i}))]\} = \hat{\beta}_*, \mathcal{T}_*(p(\vec{x}, \tau_*), max\{\hat{\beta}_*, \tau_*\} = \beta_*]$$

     $$\forall \vec{x}[\hat{\mathcal{B}}^*(p(\vec{x}), \beta^*) \leftrightarrow \sum\{\beta_i^* : (\vec{x} = \vec{t_i}) \wedge [(\psi_i \neq \{\}, \hat{\mathcal{B}}^*(\psi_i, \beta_i^*))$$
     $$\vee(\psi_i = \{\}, \hat{\mathcal{B}}^*(p(\vec{t_i}), \beta_i^*))]\} = \hat{\beta}^*, \mathcal{T}^*(p(\vec{x}, \tau^*), min\{\hat{\beta}^*, \tau^*\} = \beta^*]$$

- Axioms:

  – same as in figure 2.6.

Figure 5.7: β-$Comp(P)$

1. (a) $true\ R_{\beta*}\,(yes, 1)$.

   (b) $true\ R^*_\beta\,(yes, 1)$.

2. (a)

$$\frac{(q,\delta), max\left\{\begin{array}{l}\beta_{*i} : [p_i \leftarrow \psi_i], \sigma_i = mgu(q,p_i),\\ (\psi_i,\delta)\sigma_i\ R_{\beta*}\,(\pi_i,\beta_{*i}) \vee \sigma_i = mgu(q,p_i),\\ (\delta)\sigma_i\ R_{\beta*}\,(\pi_i,\beta'_i), \hat{\mathcal{B}}_*(p_i\sigma_i,\beta''_i), \beta_{*i} = \beta'_i + \beta''_i - 1\end{array}\right\} = \begin{array}{l}\beta'_*, \hat{\mathcal{T}}_*((q,\delta),\tau_*),\\ max\{\tau_*,\beta'_*\} = \beta_*\end{array}}{(q,\delta)\ R_{\beta*}\,(\sigma\pi,\beta_*)}.$$

   (b)

$$\frac{(q,\delta), \sum\left\{\begin{array}{l}\beta^*_i : [p_i \leftarrow \psi_i], \sigma_i = mgu(q,p_i),\\ (\psi_i,\delta)\sigma_i\ R^*_\beta\,(\pi_i,\beta^*_i) \vee \sigma_i = mgu(q,p_i),\\ (\delta)\sigma_i\ R^*_\beta\,(\pi_i,\beta'_i), \hat{\mathcal{B}}^*(p_i\sigma_i,\beta''_i), min\{\beta'_i,\beta''_i\} = \beta^*_i\end{array}\right\} = \begin{array}{l}\beta'^*, \hat{\mathcal{T}}^*((q,\delta),\tau^*),\\ min\{\tau^*,\beta'^*\} = \beta^*\end{array}}{(q,\delta)\ R^*_\beta\,(\sigma\pi,\beta^*)}.$$

   where $\sigma\pi$ are the substitutions that generate β_* and β^* and the ψ_i are non-empty conjunctions.

   (a)

$$\frac{(\neg g,\delta), g\ R^*_\beta\,(yes,\beta'), \beta' < 1, \delta\ R_{\beta*}\,(\sigma,\beta''), \hat{\mathcal{T}}_*((\neg g,\delta),\tau_*), max\{\tau_*, (1-\beta') + \beta'' - 1\} = \beta_*}{(\neg g,\delta)\ R_{\beta*}\,(\sigma,\beta_*)}.$$

   (b)

$$\frac{(\neg g,\delta), g\ R_{\beta*}\,(yes,\beta'), \beta' < 1, \delta\ R^*_\beta\,(\sigma,\beta''), \hat{\mathcal{T}}^*((\neg g,\delta),\tau^*), min\{\tau^*, (1-\beta'), \beta''\} = \beta^*}{(\neg g,\delta)\ R^*_\beta\,(\sigma,\beta^*)}.$$

3. (a) $\dfrac{(\neg g,\delta), gF^*_\beta, \delta\ R_{\beta*}\,(\sigma,\beta_*)}{(\neg g,\delta)\ R_{\beta*}\,(\sigma,\beta_*)}$.

   (b) $\dfrac{(\neg g,\delta), gF_{\beta*}, \delta\ R^*_\beta\,(\sigma,\beta^*)}{(\neg g,\delta)\ R^*_\beta\,(\sigma,\beta^*)}$.

4. (a) $\dfrac{(q,\delta), \nexists[p \leftarrow \psi] : \exists mgu(q,p)}{(q,\delta)\ F_{\beta*}}$.

   (b) $\dfrac{(q,\delta), \nexists[p \leftarrow \psi] : \exists mgu(q,p)}{(q,\delta)\ F^*_\beta}$.

   (a) $\dfrac{(q,\delta), \forall[p_i \leftarrow \psi_i] : \exists\sigma = mgu(q,p_i) \Rightarrow (\psi_i,\delta)\sigma\ F_{\beta*}}{(q,\delta)\ F_{\beta*}}$.

   (b) $\dfrac{(q,\delta), \forall[p_i \leftarrow \psi_i] : \exists\sigma = mgu(q,p_i) \Rightarrow (\psi_i,\delta)\sigma\ F^*_\beta}{(q,\delta)\ F^*_\beta}$.

5. (a) $\dfrac{(\neg g,\delta), g\ R^*_\beta\,(yes,1)}{(\neg g,\delta)\ F_{\beta*}}$.

   (b) $\dfrac{(\neg g,\delta), g\ R_{\beta*}\,(yes,1)}{(\neg g,\delta)\ F^*_\beta}$.

Figure 5.8: β-*SLDNF*

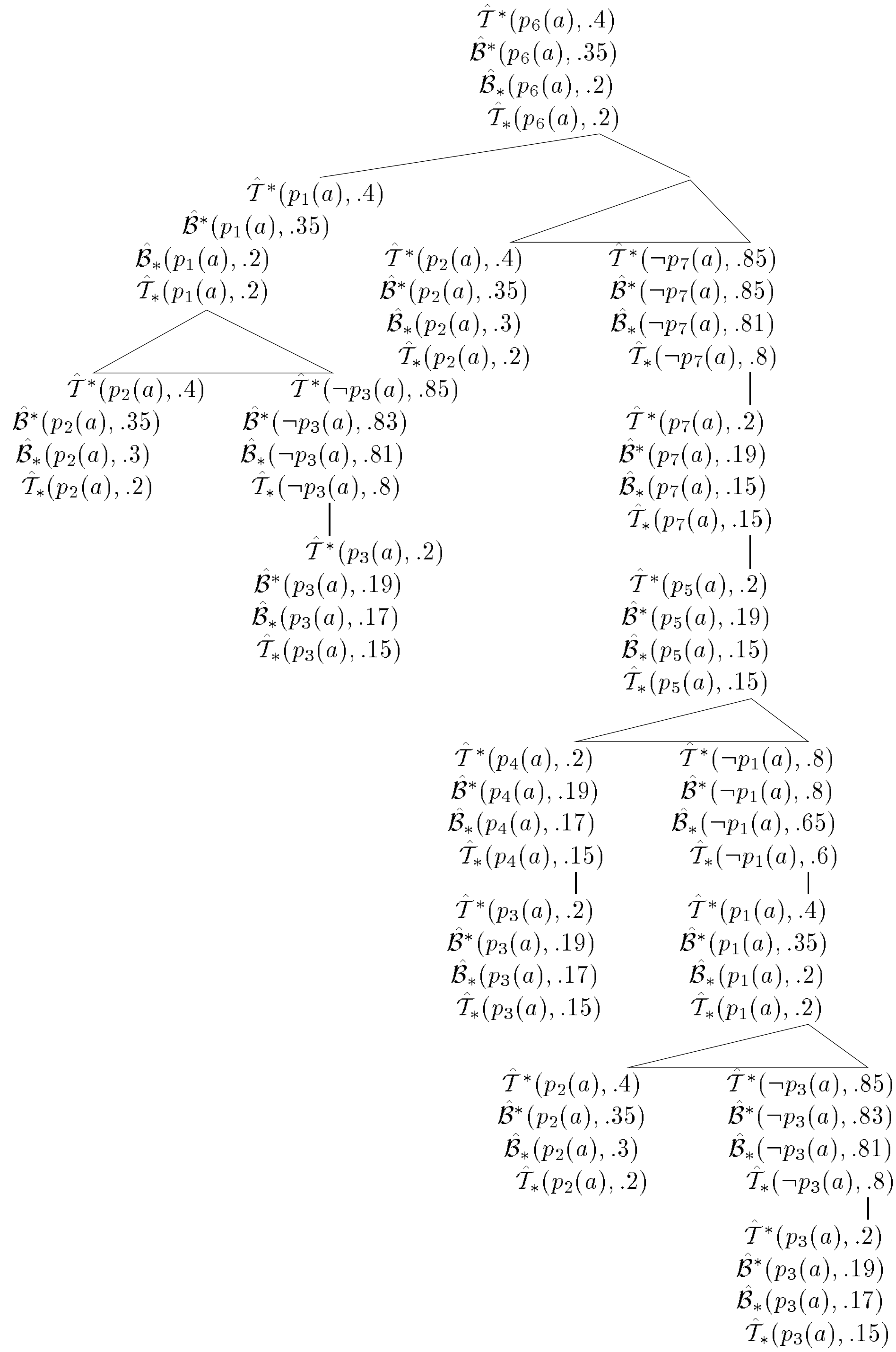

Figure 5.9: and/or-tree for $p_6(a)$

$$\begin{array}{c}
\hat{\mathcal{T}}^*(p_7(b), .1) \\
\hat{\mathcal{B}}^*(p_7(b), .07) \\
\hat{\mathcal{B}}_*(p_7(b), .06) \\
\hat{\mathcal{T}}_*(p_7(b), .05) \\
| \\
\hat{\mathcal{T}}^*(p_5(b), .1) \\
\hat{\mathcal{B}}^*(p_5(b), .07) \\
\hat{\mathcal{B}}_*(p_5(b), .06) \\
\hat{\mathcal{T}}_*(p_5(b), .05)
\end{array}$$

$$\begin{array}{cc}
\hat{\mathcal{T}}^*(p_4(b), .1) & \hat{\mathcal{T}}^*(\neg p_1(b), 1.) \\
\hat{\mathcal{B}}^*(p_4(b), .07) & \hat{\mathcal{B}}^*(\neg p_1(b), 1.) \\
\hat{\mathcal{B}}_*(p_4(b), .06) & \hat{\mathcal{B}}_*(\neg p_1(b), 1.) \\
\hat{\mathcal{T}}_*(p_4(b), .05) & \hat{\mathcal{T}}_*(\neg p_1(b), 1.) \\
| & | \\
\hat{\mathcal{T}}^*(p_3(b), .1) & F^* : p_1(b) \\
\hat{\mathcal{B}}^*(p_3(b), .07) & F^*_\beta : p_1(b) \\
\hat{\mathcal{B}}_*(p_3(b), .06) & F_{\beta *} : p_1(b) \\
\hat{\mathcal{T}}_*(p_3(b), .05) & F_* : p_1(b)
\end{array}$$

$$\begin{array}{cc}
F^* : p_2(b) & \hat{\mathcal{T}}^*(\neg p_3(b), .95) \\
F^*_\beta : p_2(b) & \hat{\mathcal{B}}^*(\neg p_3(b), .94) \\
F_{\beta *} : p_2(b) & \hat{\mathcal{B}}_*(\neg p_3(b), .93) \\
F_* : p_2(b) & \hat{\mathcal{T}}_*(\neg p_3(b), .9) \\
 & | \\
 & \hat{\mathcal{T}}^*(p_3(b), .1) \\
 & \hat{\mathcal{B}}^*(p_3(b), .07) \\
 & \hat{\mathcal{B}}_*(p_3(b), .06) \\
 & \hat{\mathcal{T}}_*(p_3(b), .05)
\end{array}$$

Figure 5.10: and/or-tree for $p_7(b)$

$$\hat{\mathcal{T}}_*(fit(flavio), 0.1)$$
$$\hat{\mathcal{B}}_*(fit(flavio), 0.15)$$
$$\hat{\mathcal{B}}^*(fit(flavio), 0.6)$$
$$\hat{\mathcal{T}}^*(fit(flavio), 0.6)$$

$$\hat{\mathcal{T}}_*(young(flavio), 0.9)$$
$$\hat{\mathcal{B}}_*(young(flavio), 0.9)$$
$$\hat{\mathcal{B}}^*(young(flavio), 0.9)$$
$$\hat{\mathcal{T}}^*(young(flavio), 0.9)$$

$$\hat{\mathcal{T}}_*(\neg spcare(flavio), 0.1)$$
$$\hat{\mathcal{B}}_*(\neg spcare(flavio), 0.1)$$
$$\hat{\mathcal{B}}^*(\neg spcare(flavio), 0.1)$$
$$\hat{\mathcal{T}}^*(\neg spcare(flavio), 0.1)$$

$$\hat{\mathcal{T}}_*(spcare(flavio), 0.4)$$
$$\hat{\mathcal{B}}_*(spcare(flavio), 0.4)$$
$$\hat{\mathcal{B}}^*(spcare(flavio), 0.75)$$
$$\hat{\mathcal{T}}^*(spcare(flavio), 0.9)$$

$$\hat{\mathcal{T}}_*(overw(flavio), 0.4)$$
$$\hat{\mathcal{B}}_*(overw(flavio), 0.4)$$
$$\hat{\mathcal{B}}^*(overw(flavio), 0.4)$$
$$\hat{\mathcal{T}}^*(overw(flavio), 0.4)$$

$$\hat{\mathcal{T}}_*(hyper(flavio), 0.3)$$
$$\hat{\mathcal{B}}_*(hyper(flavio), 0.3)$$
$$\hat{\mathcal{B}}^*(hyper(flavio), 0.35)$$
$$\hat{\mathcal{T}}^*(hyper(flavio), 0.9)$$

$$\hat{\mathcal{T}}_*(varblpres(flavio), 0.3)$$
$$\hat{\mathcal{B}}_*(varblpres(flavio), 0.3)$$
$$\hat{\mathcal{B}}^*(varblpres(flavio), 0.35)$$
$$\hat{\mathcal{T}}^*(varblpres(flavio), 0.9)$$

Figure 5.11: and/or-tree for $fit(flavio)$, including degrees of belief

6

Implementation Issues

The language presented in the previous three chapters can be implemented as a PROLOG meta-interpreter. One feature of the way the language is formalised is that it can be translated into PROLOG clauses almost literally, i.e. we can identify a term, predicate and clause in the PROLOG implementation with each term, predicate and clause in the language in an almost immediate way.

Unfortunately, this implementation is not very efficient, as it requires the exhaustive search of all solutions for any query to select the appropriate truth-value.

In this chapter we explore the applicability of two classical programming optimisation techniques to this meta-interpreter in order to improve its efficiency in processing time. The techniques we explore are $\alpha - \beta$ *pruning* and *caching*.

In the following sections we first present the "non-optimised" implementation, which is kept as close as possible to a transliteration of the language presented in chapters 3, 4 and 5. Then we introduce the $\alpha - \beta$ pruning optimisation strategy, caching, and finally we present some experimental results of the application of these techniques.

There are four PROLOG programs available, implementing:

- the "non-optimised" meta interpreter;
- the interpreter employing the $\alpha - \beta$ pruning search strategy;
- the interpreter employing the caching technique;
- the interpreter employing both $\alpha - \beta$ pruning and caching.

These programs can be accessed at the machines at the Department of Artificial Intelligence - University of Edinburgh, or received upon request directly from the author.

6.1 The "Non-Optimised" Meta-Interpreter

Recalling that a *program* P in our language is a finite non-empty set of expressions of the following forms:

- *normal clauses* of the form $p \leftarrow q_1, ..., q_n$, where p is an atom and $q_1, ..., q_n$ are literals, $n > 0$;
- *truth-degree unit clauses* of the form $\mathcal{T}_*(p, \tau_*)$ and $\mathcal{T}^*(p, \tau^*)$, representing *lower* and *upper bounds* for truth-degrees across possible worlds, where p is an atom and $\tau_*, \tau^* \in (0, 1]$;
- *belief-degree unit clauses* of the form $\mathcal{B}_*(p, \beta_*)$ and $\mathcal{B}^*(p, \beta^*)$, representing *lower* and *upper bounds* for degrees of belief across possible worlds, where p is an atom and $\beta_*, \beta^* \in (0, 1]$;
- *probability unit clauses* of the form $\mathcal{P}(S, \rho)$, representing *probabilities on the domain*, where S are sets of terms and $\rho \in [0, 1]$.

  Where:

  - τ_* is a lower bound for the truth-value of a clause, i.e. the truth-degree of that clause is not smaller than τ_* in all possible worlds;

- τ^* is an upper bound for the truth-value of a clause, i.e. the truth-degree of that clause is not greater than τ^* in all possible worlds;
- β_* is a lower bound for the degree of belief on a clause, i.e. the expected truth-degree of that clause is not smaller than β_*;
- β^* is an upper bound for the degree of belief on a clause, i.e. the expected truth-degree of that clause is not greater than β^*;
- ρ is the probability of a set of terms in the domain D of P, i.e. the probability measure of a member of the basis of an algebra of an element of a partition of D.

And that:

1. A program P is *valid* iff its values are semantically supported, i.e.:
   - for any clause ψ:
     - if τ_* is defined, then so is β_*, and $\tau_* \leq \beta_*$;
     - if β_* is defined, then so is β^*, and $\beta_* \leq \beta^*$;
     - if β^* is defined, then so is τ^*, and $\beta^* \leq \tau^*$;
   - the set of probability clauses in P is such that it *partitions* the domain D of P, i.e. $\bigcup S_i = D$, in which each S_i belongs to an expression $\mathcal{P}(S_i, \rho_i)$, and there are no S, S' such that $S \cap S' \neq \{\}$ and $\mathcal{P}(S, \rho), \mathcal{P}(S', \rho') \in P$. Furthermore, the values ρ follow the axioms of probability measures for each algebra of the partition of D.
2. A query Q is *valid* iff it is a non-empty finite conjunction of expressions of the following forms:
   - $\hat{\mathcal{T}}_*(\psi, \tau_*)$ or $\hat{\mathcal{T}}^*(\psi, \tau^*)$, where ψ is a *query clause*, and τ_* and τ^* are numerical term;
   - $\hat{\mathcal{B}}_*(\psi, \beta_*)$ or $\hat{\mathcal{B}}^*(\psi, \beta^*)$, where ψ is a *query clause*, and β_* and β^* are numerical term;

- $\hat{\mathcal{P}}_*(S, \psi, \rho_*)$ or $\hat{\mathcal{P}}^*(S, \psi, \rho^*)$, where S is a *vector of variables*, ψ is a *query clause*, and ρ_* and ρ^* are numerical terms;

The language presented in the previous chapters can be implemented as a PROLOG meta-interpreter as follows:

- *$*$-SLDNF*:

  Given a program P and a query $\hat{\mathcal{T}}_*(\psi, \tau_*)$:

  1. assume that $\psi = (\varphi, \delta)$, where φ is a literal and δ is a query clause (notice that $\varphi = (\varphi, true)$).
  2. solve $\hat{\mathcal{T}}_*(\varphi, \tau_\varphi)$, resulting either $\varphi\, R_*\, (\sigma_\varphi, \tau_\varphi)$ or $\varphi\, F_*$.

     if $\varphi\, F_*$ then return $\psi\, F_*$.
  3. if $\varphi\, R_*\, (\sigma_\varphi, \tau_\varphi)$ then solve $\hat{\mathcal{T}}_*(\delta\sigma_\varphi, \tau_\delta)$, resulting either $\delta\, R_*\, (\sigma_\varphi\sigma_\delta, \tau_\delta)$ or $\delta\sigma_\varphi\, F_*$.

     if $\delta\sigma_\varphi\, F_*$ then return $\psi\, F_*$.
  4. if $\delta\, R_*\, (\sigma_\varphi\sigma_\delta, \tau_\delta)$ then return $\psi\, R_*\, (\sigma_\varphi\sigma_\delta, min\{\tau_\varphi, \tau_\delta\})$.

  To solve $\hat{\mathcal{T}}_*(\varphi, \tau_\varphi)$:

  1. if $\varphi = true$ then return $true\, R_*\, (yes, 1)$.
  2. if $\varphi =$ positive literal q then find the collection C of all expressions E in P such that

     (a) $E = p_i \leftarrow \psi_i$ and $\exists \sigma_i = mgu(q, p_i)$ and $\hat{\mathcal{T}}_*(\psi_i\sigma_i, \tau_{*i})$, or

     (b) $E = \hat{\mathcal{T}}_*(p_i, \tau_{*i})$ and $\exists \sigma_i = mgu(q, p_i)$.

     if $C = \{\}$ then return $q\, F_*$. Otherwise

     select from C the index j such that $\tau_{*j} = max_C\{\tau_{*i}\}$ and return $q\, R_*\, (\sigma_j, \tau_{*j})$.
  3. if $\varphi =$ negative ground literal $\neg g$ then solve $\hat{\mathcal{T}}^*(g, \tau_g)$.

if $g\ F^*$ then return $\neg g\ R_*\,(yes, 1)$.

if $g\ R^*\,(\sigma_g, 1)$ then return $\neg g F_*$.

if $g\ R^*\,(\sigma_g, \tau_g), \tau_g < 1$ then return $\neg g\ R_*\,(\sigma_g, 1 - \tau_g)$.

Given a program P and a query $\hat{\mathcal{T}}^*(\psi, \tau^*)$:

1. assume that $\psi = (\varphi, \delta)$, where φ is a literal and δ is a query clause.
2. solve $\hat{\mathcal{T}}^*(\varphi, \tau_\varphi)$, resulting either $\varphi\ R^*\,(\sigma_\varphi, \tau_\varphi)$ or $\varphi\ F^*$.

   if $\varphi\ F^*$ then return $\psi\ F^*$.
3. if $\varphi\ R^*\,(\sigma_\varphi, \tau_\varphi)$ then solve $\hat{\mathcal{T}}^*(\delta\sigma_\varphi, \tau_\delta)$, resulting either $\delta\ R^*\,(\sigma_\varphi\sigma_\delta, \tau_\delta)$ or $\delta\sigma_\varphi\ F^*$.

   if $\delta\sigma_\varphi\ F^*$ then return $\psi\ F^*$.
4. if $\delta\ R^*\,(\sigma_\varphi\sigma_\delta, \tau_\delta)$ then return $\psi\ R^*\,(\sigma_\varphi\sigma_\delta, min\{\tau_\varphi, \tau_\delta\})$.

To solve $\hat{\mathcal{T}}^*(\varphi, \tau_\varphi)$:

1. if $\varphi = true$ then return $true\ R^*\,(yes, 1)$.
2. if $\varphi =$ positive literal q then find the collection C of all expressions E in P such that

   (a) $E = p_i \leftarrow \psi_i$ and $\exists\sigma_i = mgu(q, p_i)$ and $\hat{\mathcal{T}}^*(\psi_i\sigma_i, \tau_i^*)$, or

   (b) $E = \hat{\mathcal{T}}^*(p_i, \tau_i^*)$ and $\exists\sigma_i = mgu(q, p_i)$.

   if $C = \{\}$ then return $q\ F^*$. Otherwise

   select from C the index j such that $\tau_j^* = max_C\{\tau_i^*\}$ and return $q\ R^*\,(\sigma_j, \tau_j^*)$.
3. if $\varphi =$ negative ground literal $\neg g$ then solve $\hat{\mathcal{T}}_*(g, \tau_g)$.

   if $g\ F_*$ then return $\neg g\ R^*\,(yes, 1)$.

   if $g\ R_*\,(\sigma_g, 1)$ then return $\neg g F^*$.

   if $g\ R_*\,(\sigma_g, \tau_g), \tau_g < 1$ then return $\neg g\ R^*\,(\sigma_g, 1 - \tau_g)$.

- *β-SLDNF*:

Given a program P and a query $\hat{\mathcal{B}}_*(\psi, \beta_*)$:

1. assume that $\psi = (\varphi, \delta)$, where φ is a literal and δ is a query clause.
2. solve $\hat{\mathcal{B}}_*(\varphi, \beta_\varphi)$, resulting either $\varphi\, R_{\beta *}\,(\sigma_\varphi, \beta_\varphi)$ or $\varphi\, F_{\beta *}$.

   if $\varphi\, F_{\beta *}$ then return $\psi\, F_{\beta *}$.
3. if $\varphi\, R_{\beta *}\,(\sigma_\varphi, \beta_\varphi)$ then solve $\hat{\mathcal{B}}_*(\delta\sigma_\varphi, \beta_\delta)$, resulting either $\delta\, R_{\beta *}\,(\sigma_\varphi\sigma_\delta, \beta_\delta)$ or $\delta\sigma_\varphi\, F_{\beta *}$.

   if $\delta\sigma_\varphi\, F_{\beta *}$ then return $\psi\, F_{\beta *}$.
4. if $\delta\, R_{\beta *}\,(\sigma_\varphi\sigma_\delta, \beta_\delta)$ then solve $\hat{\mathcal{T}}_*(\psi, \tau_*)$ and return $\psi\, R_{\beta *}\,(\sigma_\varphi\sigma_\delta, max\{\tau_*, \beta_\varphi + \beta_\delta - 1\})$.

To solve $\hat{\mathcal{B}}_*(\varphi, \beta_\varphi)$:

1. if $\varphi = true$ then return $true\; R_{\beta *}\,(yes, 1)$.
2. if $\varphi =$ positive literal q then find the collection C of all expressions E in P such that

   (a) $E = p_i \leftarrow \psi_i$ and $\exists\sigma_i = mgu(q, p_i)$ and $\hat{\mathcal{B}}_*(\psi_i\sigma_i, \beta_{*i})$, or

   (b) $E = \hat{\mathcal{B}}_*(p_i, \beta_{*i})$ and $\exists\sigma_i = mgu(q, p_i)$.

   if $C = \{\}$ then return $q\, F_{\beta *}$. Otherwise

   select from C the index j such that $\beta_{*j} = max_C\{\beta_{*i}\}$ and return $q\, R_{\beta *}\,(\sigma_j, \beta_{*j})$.
3. if $\varphi =$ negative ground literal $\neg g$ then solve $\hat{\mathcal{B}}^*(g, \beta_g)$.

   if $g\, F^*_\beta$ then return $\neg g\; R_{\beta *}\,(yes, 1)$.

   if $g\, R^*_\beta\,(\sigma_g, 1)$ then return $\neg g F_{\beta *}$.

   if $g\, R^*_\beta\,(\sigma_g, \beta_g), \beta_g < 1$ then return $\neg g\; R_{\beta *}\,(\sigma_g, 1 - \beta_g)$.

Given a program P and a query $\hat{\mathcal{B}}^*(\psi, \beta^*)$:

1. assume that $\psi = (\varphi, \delta)$, where φ is a literal and δ is a query clause.
2. solve $\hat{\mathcal{B}}^*(\varphi, \beta_\varphi)$, resulting either $\varphi\, R^*_\beta\,(\sigma_\varphi, \beta_\varphi)$ or $\varphi\, F^*_\beta$.

   if $\varphi\, F^*_\beta$ then return $\psi\, F^*_\beta$.
3. if $\varphi\, R^*_\beta\,(\sigma_\varphi, \beta_\varphi)$ then solve $\hat{\mathcal{B}}^*(\delta\sigma_\varphi, \beta_\delta)$, resulting either $\delta\, R^*_\beta\,(\sigma_\varphi\sigma_\delta, \beta_\delta)$ or $\delta\sigma_\varphi\, F^*_\beta$.

   if $\delta\sigma_\varphi\, F^*_\beta$ then return $\psi\, F^*_\beta$.

4. if $\delta\ R^*_\beta\,(\sigma_\varphi\sigma_\delta, \beta_\delta)$ then solve $\hat{\mathcal{T}}^*(\psi, \tau^*)$ and return $\psi\ R^*_\beta\,(\sigma_\varphi\sigma_\delta, min\{\tau^*, \beta_\varphi, \beta_\delta\})$.

To solve $\hat{\mathcal{B}}^*(\varphi, \beta_\varphi)$:

1. if $\varphi = true$ then return $true\ R^*_\beta\,(yes, 1)$.
2. if $\varphi =$ positive literal q then find the collection C of all expressions E in P such that
   (a) $E = p_i \leftarrow \psi_i$ and $\exists \sigma_i = mgu(q, p_i)$ and $\hat{\mathcal{B}}^*(\psi_i\sigma_i, \beta^*_i)$, or
   (b) $E = \hat{\mathcal{B}}^*(p_i, \beta^*_i)$ and $\exists \sigma_i = mgu(q, p_i)$.

   if $C = \{\}$ then return $q\ F^*_\beta$. Otherwise

   calculate $\beta'^* = \sum_C\{\beta^*_i\}$ and return $q\ R^*_\beta\,(yes, \beta'^*)$.
3. if $\varphi =$ negative ground literal $\neg g$ then solve $\hat{\mathcal{B}}_*(g, \beta_g)$.

   if $g\ F_{\beta*}$ then solve $\hat{\mathcal{T}}^*(\neg g, \tau_g)$ and return $\neg g\ R^*_\beta\,(yes, \tau_g)$.

   if $g\ R_{\beta*}\,(\sigma_g, 1)$ then return $\neg g F^*_\beta$.

   if $g\ R_{\beta*}\,(\sigma_g, \beta_g), \beta_g < 1$ then solve $\hat{\mathcal{T}}^*(\neg g, \tau_g)$ and return $\neg g\ R^*_\beta\,(\sigma_g, min\{\tau_g, 1 - \beta_g\})$.

- Conditional Beliefs:

  Given a program P and a query $\hat{\mathcal{B}}_*(\psi_1|\psi_2, \beta_*)$:

  1. solve $\hat{\mathcal{B}}_*((\psi_1, \psi_2), \beta'_*)$ and $\hat{\mathcal{B}}^*(\psi_2, \beta^*)$.
  2. if $\beta^* = 0$ then return $\beta_* = 0$. Otherwise
  3. return $\beta_* = \frac{\beta'_*}{\beta^*}$.

  Given a program P and a query $\hat{\mathcal{B}}^*(\psi_1|\psi_2, \beta^*)$:

  1. solve $\hat{\mathcal{B}}^*((\psi_1, \psi_2), \beta'^*)$ and $\hat{\mathcal{B}}_*(\psi_2, \beta_*)$.
  2. if $\beta_* = 0$ then return $\beta^* = 0$. Otherwise
  3. return $\beta^* = \frac{\beta'^*}{\beta_*}$.

- Probabilities and Conditional Probabilities on the Domain:

  as presented in chapter 5.4.

6.2 Computational Optimisation I: $\alpha - \beta$ Pruning

Some values are generated and immediately discarded when computing the collections C of expressions in $\hat{\mathcal{T}}_*, \hat{\mathcal{T}}^*$ and $\hat{\mathcal{B}}_*$, as only the maximum value in C is used for further calculations. Similarly, when generating $\hat{\mathcal{T}}_*, \hat{\mathcal{T}}^*$ and $\hat{\mathcal{B}}^*$ for conjunctive queries, the minimum value obtained for a conjunct is used and the others are discarged.

In [Mel85] the $\alpha - \beta$ *pruning strategy*, originally proposed to increase the efficiency of search processes in game trees, is employed to guide query evaluation in expert systems. The fundamental idea is that some operations can be simplified in the evaluation procedure whenever a value is selected from a partially ordered set of alternatives and the choice is made based on this partial order. We do not have to generate the (truth or belief) values for all elements of C if we can generate *one* value and verify that no other value can be greater than it, and we do not have to generate the (truth of belief) values for each conjunct in a query if we can generate *one* value and verify that no other conjunct can produce a smaller value.

Assuming that:

- $\alpha\hat{N}(\delta, \hat{\alpha})$ denotes that $\hat{\alpha}$ is a *lower bound* for α in $\hat{N}(\delta, \alpha)$ - where $\hat{N}$ is one of $\hat{\mathcal{T}}_*, \hat{\mathcal{T}}^*, \hat{\mathcal{B}}_*, \hat{\mathcal{B}}^*$, and

- $\beta\hat{N}(\varphi, \hat{\beta})$ denotes that $\hat{\beta}$ is an *upper bound* for β in $\hat{N}(\varphi, \beta)$.

- **-SLDNF*:

  Given a program P and a query $\hat{\mathcal{T}}_*(\psi, \tau_*)$:

  1. assume that $\psi = (\varphi, \delta)$, where φ is a literal and δ is a query clause.

2. solve $\hat{\mathcal{T}}_*(\varphi, \tau_\varphi)$, resulting either $\varphi\ R_*\,(\sigma_\varphi, \tau_\varphi)$ or $\varphi\ F_*$.

   if $\varphi\ F_*$ then return $\psi\ F_*$.

3. if $\varphi\ R_*\,(\sigma_\varphi, \tau_\varphi)$ then solve $\alpha\hat{\mathcal{T}}_*(\delta\sigma_\varphi, \hat{\tau}_\delta)$.

   if $\hat{\tau}_\delta \geq \tau_\varphi$ then return $\psi\ R_*\,(\sigma_\varphi, \tau_\varphi)$. Otherwise

4. solve $\hat{\mathcal{T}}_*(\delta\sigma_\varphi, \tau_\delta)$, resulting either $\delta\ R_*\,(\sigma_\varphi\sigma_\delta, \tau_\delta)$ or $\delta\sigma_\varphi\ F_*$.

   if $\delta\sigma_\varphi\ F_*$ then return $\psi\ F_*$.

5. if $\delta\ R_*\,(\sigma_\varphi\sigma_\delta, \tau_\delta)$ then return $\psi\ R_*\,(\sigma_\varphi\sigma_\delta, min\{\tau_\varphi, \tau_\delta\})$.

To solve $\hat{\mathcal{T}}_*(\varphi, \tau_\varphi)$:

1. if $\varphi = true$ then return $true\ R_*\,(yes, 1)$.

2. if $\varphi =$ positive literal q then find one expression E in P such that

   (a) $E = p_i \leftarrow \psi_i$ and $\exists\sigma_i = mgu(q, p_i)$ and $\hat{\mathcal{T}}_*(\psi_i\sigma_i, \tau_{*i})$, or

   (b) $E = \hat{\mathcal{T}}_*(p_i, \tau_{*i})$ and $\exists\sigma_i = mgu(q, p_i)$.

   if there is no such E then return $q\ F_*$. Otherwise solve $\beta\hat{\mathcal{T}}_*(\varphi, \hat{\tau}_\varphi)$.

   if $\hat{\tau}_\varphi \leq \tau_{*i}$ then return $\varphi\ R_*\,(\sigma_i, \tau_{*i})$. Otherwise find another expression E with the same conditions and iterate the process.

3. if $\varphi =$ negative ground literal $\neg g$ then solve $\hat{\mathcal{T}}^*(g, \tau_g)$.

   if $g\ F^*$ then return $\neg g\ R_*\,(yes, 1)$.

   if $g\ R^*\,(\sigma_g, 1)$ then return $\neg g F_*$.

   if $g\ R^*\,(\sigma_g, \tau_g), \tau_g < 1$ then return $\neg g\ R_*\,(\sigma_g, 1 - \tau_g)$.

To solve $\alpha\hat{\mathcal{T}}_*(\psi, \tau)$:

1. assume that $\psi = (\varphi, \delta)$, where φ is a literal and δ is a query clause.

2. solve $\alpha\hat{\mathcal{T}}_*(\varphi, \tau_\varphi)$, resulting either $\varphi\ R_*\,(\sigma_\varphi, \tau_\varphi)$ or $\varphi\ F_*$.

   if $\varphi\ F_*$ then return $\psi\ F_*$.

3. if $\varphi\ R_*\ (\sigma_\varphi, \tau_\varphi)$ then solve $\alpha\hat{\mathcal{T}}_*(\delta\sigma_\varphi, \hat{\tau}_\delta)$.

   if $\hat{\tau}_\delta \geq \tau_\varphi$ then return τ_φ. Otherwise

4. solve $\hat{\mathcal{T}}_*(\delta\sigma_\varphi, \tau_\delta)$, resulting either $\delta\ R_*\ (\sigma_\varphi\sigma_\delta, \tau_\delta)$ or $\delta\sigma_\varphi\ F_*$.

   if $\delta\sigma_\varphi\ F_*$ then return $\psi\ F_*$.

5. if $\delta\ R_*\ (\sigma_\varphi\sigma_\delta, \tau_\delta)$ then return $min\{\tau_\varphi, \tau_\delta\}$.

To solve $\alpha\hat{\mathcal{T}}_*(\varphi, \tau_\varphi)$:

1. if $\varphi = true$ then return $true\ R_*\ (yes, 1)$.

2. if $\varphi =$ positive literal q then find one expression E in P such that

   (a) $E = p_i \leftarrow \psi_i$ and $\exists\sigma_i = mgu(q, p_i)$ and $\hat{\mathcal{T}}_*(\psi_i\sigma_i, \tau_{*i})$, or

   (b) $E = \hat{\mathcal{T}}_*(p_i, \tau_{*i})$ and $\exists\sigma_i = mgu(q, p_i)$.

   if there is no such E then return $q\ F_*$. Otherwise return τ_{*i}.

3. if $\varphi =$ negative ground literal $\neg g$ then solve $\hat{\mathcal{T}}^*(g, \tau_g)$.

   if $g\ F^*$ then return $\tau_\varphi = 1$.

   if $g\ R^*\ (\sigma_g, 1)$ then return $\neg g F_*$.

   if $g\ R^*\ (\sigma_g, \tau_g), \tau_g < 1$ then return $\tau_\varphi = 1 - \tau_g$.

To solve $\beta\hat{\mathcal{T}}_*(\psi, \tau)$:

1. assume that $\psi = (\varphi, \delta)$, where φ is a literal and δ is a query clause.

2. solve $\hat{\mathcal{T}}_*(\varphi, \tau_\varphi)$, resulting either $\varphi\ R_*\ (\sigma_\varphi, \tau_\varphi)$ or $\varphi\ F_*$.

   if $\varphi\ F_*$ then return $\psi\ F_*$. Otherwise return τ_φ.

To solve $\beta\hat{\mathcal{T}}_*(\varphi, \tau_\varphi)$:

1. if $\varphi = true$ then return $true\ R_*\ (yes, 1)$.

2. if $\varphi =$ positive literal q then find one expression E in P such that

   (a) $E = p_i \leftarrow \psi_i$ and $\exists \sigma_i = mgu(q, p_i)$ and $\hat{\mathcal{T}}_*(\psi_i\sigma_i, \tau_{*i})$, or

   (b) $E = \hat{\mathcal{T}}_*(p_i, \tau_{*i})$ and $\exists \sigma_i = mgu(q, p_i)$.

   if there is no such E then return $q\ F_*$. Otherwise solve $\beta\hat{\mathcal{T}}_*(\varphi, \hat{\tau}_\varphi)$.

   if $\hat{\tau}_\varphi \leq \tau_{*i}$ then return $\varphi\ R_*\ (\sigma_i, \tau_{*i})$. Otherwise find another expression E with the same conditions and iterate the process.

3. if $\varphi =$ negative ground literal $\neg g$ then solve $\hat{\mathcal{T}}^*(g, \tau_g)$.

   if $g\ F^*$ then return $\neg g\ R_*\ (yes, 1)$.

   if $g\ R^*\ (\sigma_g, 1)$ then return $\neg g F_*$.

   if $g\ R^*\ (\sigma_g, \tau_g), \tau_g < 1$ then return $\neg g\ R_*\ (\sigma_g, 1 - \tau_g)$.

Given a program P and a query $\hat{\mathcal{T}}^*(\psi, \tau^*)$:

1. assume that $\psi = (\varphi, \delta)$, where φ is a literal and δ is a query clause.

2. solve $\hat{\mathcal{T}}^*(\varphi, \tau_\varphi)$, resulting either $\varphi\ R^*\ (\sigma_\varphi, \tau_\varphi)$ or $\varphi\ F^*$.

   if $\varphi\ F^*$ then return $\psi\ F^*$.

3. if $\varphi\ R^*\ (\sigma_\varphi, \tau_\varphi)$ then solve $\alpha\hat{\mathcal{T}}^*(\delta\sigma_\varphi, \hat{\tau}_\delta)$.

   if $\hat{\tau}_\delta \geq \tau_\varphi$ then return $\psi\ R^*\ (\sigma_\varphi, \tau_\varphi)$. Otherwise

4. solve $\hat{\mathcal{T}}^*(\delta\sigma_\varphi, \tau_\delta)$, resulting either $\delta\ R^*\ (\sigma_\varphi\sigma_\delta, \tau_\delta)$ or $\delta\sigma_\varphi\ F^*$.

   if $\delta\sigma_\varphi\ F^*$ then return $\psi\ F^*$.

5. if $\delta\ R^*\ (\sigma_\varphi\sigma_\delta, \tau_\delta)$ then return $\psi\ R^*\ (\sigma_\varphi\sigma_\delta, min\{\tau_\varphi, \tau_\delta\})$.

To solve $\hat{\mathcal{T}}^*(\varphi, \tau_\varphi)$:

1. if $\varphi = true$ then return $true\ R^*\ (yes, 1)$.

2. if $\varphi =$ positive literal q then find one expression E in P such that

(a) $E = p_i \leftarrow \psi_i$ and $\exists \sigma_i = mgu(q, p_i)$ and $\hat{\mathcal{T}}^*(\psi_i \sigma_i, \tau_i^*)$, or

(b) $E = \hat{\mathcal{T}}^*(p_i, \tau_i^*)$ and $\exists \sigma_i = mgu(q, p_i)$.

if there is no such E then return $q\ F^*$. Otherwise solve $\beta \hat{\mathcal{T}}^*(\varphi, \hat{\tau}_\varphi)$.

if $\hat{\tau}_\varphi \leq \tau_i^*$ then return $\varphi\ R^*\ (\sigma_i, \tau_i^*)$. Otherwise find another expression E with the same conditions and iterate the process.

3. if $\varphi =$ negative ground literal $\neg g$ then solve $\hat{\mathcal{T}}_*(g, \tau_g)$.

   if $g\ F_*$ then return $\neg g\ R^*\ (yes, 1)$.

   if $g\ R_*\ (\sigma_g, 1)$ then return $\neg g F^*$.

   if $g\ R_*\ (\sigma_g, \tau_g), \tau_g < 1$ then return $\neg g\ R^*\ (\sigma_g, 1 - \tau_g)$.

To solve $\alpha \hat{\mathcal{T}}^*(\psi, \tau)$:

1. assume that $\psi = (\varphi, \delta)$, where φ is a literal and δ is a query clause.

2. solve $\alpha \hat{\mathcal{T}}^*(\varphi, \tau_\varphi)$, resulting either $\varphi\ R^*\ (\sigma_\varphi, \tau_\varphi)$ or $\varphi\ F^*$.

   if $\varphi\ F^*$ then return $\psi\ F^*$.

3. if $\varphi\ R^*\ (\sigma_\varphi, \tau_\varphi)$ then solve $\alpha \hat{\mathcal{T}}^*(\delta \sigma_\varphi, \hat{\tau}_\delta)$.

   if $\hat{\tau}_\delta \geq \tau_\varphi$ then return τ_φ. Otherwise

4. solve $\hat{\mathcal{T}}^*(\delta \sigma_\varphi, \tau_\delta)$, resulting either $\delta\ R^*\ (\sigma_\varphi \sigma_\delta, \tau_\delta)$ or $\delta \sigma_\varphi\ F^*$.

   if $\delta \sigma_\varphi\ F^*$ then return $\psi\ F^*$.

5. if $\delta\ R^*\ (\sigma_\varphi \sigma_\delta, \tau_\delta)$ then return $min\{\tau_\varphi, \tau_\delta\}$.

To solve $\alpha \hat{\mathcal{T}}^*(\varphi, \tau_\varphi)$:

1. if $\varphi = true$ then return $true\ R^*\ (yes, 1)$.

2. if $\varphi =$ positive literal q then find one expression E in P such that

   (a) $E = p_i \leftarrow \psi_i$ and $\exists \sigma_i = mgu(q, p_i)$ and $\hat{\mathcal{T}}^*(\psi_i \sigma_i, \tau_i^*)$, or

(b) $E = \hat{\mathcal{T}}^*(p_i, \tau_i^*)$ and $\exists \sigma_i = mgu(q, p_i)$.

if there is no such E then return $q\ F^*$. Otherwise return τ_i^*.

3. if $\varphi =$ negative ground literal $\neg g$ then solve $\hat{\mathcal{T}}_*(g, \tau_g)$.

   if $g\ F_*$ then return $\tau_\varphi = 1$.

   if $g\ R_*\ (\sigma_g, 1)$ then return $\neg g F^*$.

   if $g\ R_*\ (\sigma_g, \tau_g), \tau_g < 1$ then return $\tau_\varphi = 1 - \tau_g$.

To solve $\beta\hat{\mathcal{T}}^*(\psi, \tau)$:

1. assume that $\psi = (\varphi, \delta)$, where φ is a literal and δ is a query clause.

2. solve $\hat{\mathcal{T}}^*(\varphi, \tau_\varphi)$, resulting either $\varphi\ R^*\ (\sigma_\varphi, \tau_\varphi)$ or $\varphi\ F^*$.

   if $\varphi\ F^*$ then return $\psi\ F^*$. Otherwise return τ_φ.

To solve $\beta\hat{\mathcal{T}}^*(\varphi, \tau_\varphi)$:

1. if $\varphi = true$ then return $true\ R^*\ (yes, 1)$.

2. if $\varphi =$ positive literal q then find one expression E in P such that

   (a) $E = p_i \leftarrow \psi_i$ and $\exists \sigma_i = mgu(q, p_i)$ and $\hat{\mathcal{T}}^*(\psi_i \sigma_i, \tau_i^*)$, or

   (b) $E = \hat{\mathcal{T}}^*(p_i, \tau_i^*)$ and $\exists \sigma_i = mgu(q, p_i)$.

   if there is no such E then return $q\ F^*$. Otherwise solve $\beta\hat{\mathcal{T}}^*(\varphi, \hat{\tau}_\varphi)$.

   if $\hat{\tau}_\varphi \leq \tau_i^*$ then return $\varphi\ R^*\ (\sigma_i, \tau_i^*)$. Otherwise find another expression E with the same conditions and iterate the process.

3. if $\varphi =$ negative ground literal $\neg g$ then solve $\hat{\mathcal{T}}_*(g, \tau_g)$.

   if $g\ F_*$ then return $\neg g\ R^*\ (yes, 1)$.

   if $g\ R_*\ (\sigma_g, 1)$ then return $\neg g F^*$.

   if $g\ R_*\ (\sigma_g, \tau_g), \tau_g < 1$ then return $\neg g\ R^*\ (\sigma_g, 1 - \tau_g)$.

- *β-SLDNF*:

  Given a program P and a query $\hat{\mathcal{B}}_*(\psi, \beta_*)$:

  1. assume that $\psi = (\varphi, \delta)$, where φ is a literal and δ is a query clause.
  2. solve $\hat{\mathcal{B}}_*(\varphi, \beta_\varphi)$, resulting either $\varphi\ R_{\beta*}\,(\sigma_\varphi, \beta_\varphi)$ or $\varphi\ F_{\beta*}$.

     if $\varphi\ F_{\beta*}$ then return $\psi\ F_{\beta*}$.
  3. if $\varphi\ R_{\beta*}\,(\sigma_\varphi, \beta_\varphi)$ then solve $\hat{\mathcal{B}}_*(\delta\sigma_\varphi, \beta_\delta)$, resulting either $\delta\ R_{\beta*}\,(\sigma_\varphi\sigma_\delta, \beta_\delta)$ or $\delta\sigma_\varphi\ F_{\beta*}$.

     if $\delta\sigma_\varphi\ F_{\beta*}$ then return $\psi\ F_{\beta*}$.
  4. if $\delta\ R_{\beta*}\,(\sigma_\varphi\sigma_\delta, \beta_\delta)$ then solve $\hat{\mathcal{T}}_*(\psi, \tau_*)$ and return $\psi\ R_{\beta*}\,(\sigma_\varphi\sigma_\delta, max\{\tau_*, \beta_\varphi + \beta_\delta - 1\})$.

  To solve $\hat{\mathcal{B}}_*(\varphi, \beta_\varphi)$:

  1. if $\varphi = true$ then return $true\ R_{\beta*}\,(yes, 1)$.
  2. if $\varphi =$ positive literal q then find the collection C of all expressions E in P such that

     (a) $E = p_i \leftarrow \psi_i$ and $\exists\sigma_i = mgu(q, p_i)$ and $\hat{\mathcal{B}}_*(\psi_i\sigma_i, \beta_{*i})$, or

     (b) $E = \hat{\mathcal{B}}_*(p_i, \beta_{*i})$ and $\exists\sigma_i = mgu(q, p_i)$.

     if $C = \{\}$ then return $q\ F_{\beta*}$. Otherwise

     select from C the index j such that $\beta_{*j} = max_C\{\beta_{*i}\}$ and return $q\ R_{\beta*}\,(\sigma_j, \beta_{*j})$.
  3. if $\varphi =$ negative ground literal $\neg g$ then solve $\hat{\mathcal{B}}^*(g, \beta_g)$.

     if $g\ F^*_\beta$ then return $\neg g\ R_{\beta*}\,(yes, 1)$.

     if $g\ R^*_\beta\,(\sigma_g, 1)$ then return $\neg g F_{\beta*}$.

     if $g\ R^*_\beta\,(\sigma_g, \beta_g), \beta_g < 1$ then return $\neg g\ R_{\beta*}\,(\sigma_g, 1 - \beta_g)$.

  Given a program P and a query $\hat{\mathcal{B}}^*(\psi, \beta^*)$:

  1. assume that $\psi = (\varphi, \delta)$, where φ is a literal and δ is a query clause.

2. solve $\hat{\mathcal{B}}^*(\varphi, \beta_\varphi)$, resulting either $\varphi\ R^*_\beta\,(\sigma_\varphi, \beta_\varphi)$ or $\varphi\ F^*_\beta$.

   if $\varphi\ F^*_\beta$ then return $\psi\ F^*_\beta$.

3. if $\varphi\ R^*_\beta\,(\sigma_\varphi, \beta_\varphi)$ then solve $\alpha\hat{\mathcal{B}}^*(\delta\sigma_\varphi, \hat{\beta}_\delta)$.

   if $\hat{\beta}_\varphi \geq \beta_\varphi$ then solve $\hat{\mathcal{T}}^*(\psi, \tau^*)$ and return $\psi\ R^*_\beta\,(\sigma_\varphi, min\{\tau^*, \beta_\varphi\})$. Otherwise

4. solve $\hat{\mathcal{B}}^*(\delta\sigma_\varphi, \beta_\delta)$, resulting either $\delta\ R^*_\beta\,(\sigma_\varphi\sigma_\delta, \beta_\delta)$ or $\delta\sigma_\varphi\ F^*_\beta$.

   if $\delta\sigma_\varphi\ F^*_\beta$ then return $\psi\ F^*_\beta$.

5. if $\delta\ R^*_\beta\,(\sigma_\varphi\sigma_\delta, \beta_\delta)$ then solve $\hat{\mathcal{T}}^*(\psi, \tau^*)$ and return $\psi\ R^*_\beta\,(\sigma_\varphi\sigma_\delta, min\{\tau^*, \beta_\varphi, \beta_\delta\})$.

To solve $\hat{\mathcal{B}}^*(\varphi, \beta_\varphi)$:

1. if $\varphi = true$ then return $true\ R^*_\beta\,(yes, 1)$.

2. if $\varphi =$ positive literal q then find the collection C of all expressions E in P such that

   (a) $E = p_i \leftarrow \psi_i$ and $\exists\sigma_i = mgu(q, p_i)$ and $\hat{\mathcal{B}}^*(\psi_i\sigma_i, \beta^*_i)$, or

   (b) $E = \hat{\mathcal{B}}^*(p_i, \beta^*_i)$ and $\exists\sigma_i = mgu(q, p_i)$.

   if $C = \{\}$ then return $q\ F^*_\beta$. Otherwise

   calculate $\beta'^* = \sum_C\{\beta^*_i\}$ and return $q\ R^*_\beta\,(yes, \beta'^*)$.

3. if $\varphi =$ negative ground literal $\neg g$ then solve $\hat{\mathcal{B}}_*(g, \beta_g)$.

   if $g\ F_{\beta*}$ then solve $\hat{\mathcal{T}}^*(\neg g, \tau_g)$ and return $\neg g\ R^*_\beta\,(yes, \tau_g)$.

   if $g\ R_{\beta*}\,(\sigma_g, 1)$ then return $\neg g F^*_\beta$.

   if $g\ R_{\beta*}\,(\sigma_g, \beta_g), \beta_g < 1$ then solve $\hat{\mathcal{T}}^*(\neg g, \tau_g)$ and return $\neg g\ R^*_\beta\,(\sigma_g, min\{\tau_g, 1 - \beta_g\})$.

To solve $\alpha\hat{\mathcal{B}}^*(\psi, \beta)$:

1. assume that $\psi = (\varphi, \delta)$, where φ is a literal and δ is a query clause.

2. solve $\hat{\mathcal{B}}^*(\varphi, \beta_\varphi)$, resulting either $\varphi\ R^*_\beta\,(\sigma_\varphi, \beta_\varphi)$ or $\varphi\ F^*_\beta$.

   if $\varphi\ F^*_\beta$ then return $\psi\ F^*_\beta$.

3. if $\varphi\ R^*_\beta\,(\sigma_\varphi, \beta_\varphi)$ then solve $\alpha\hat{\mathcal{B}}^*(\delta\sigma_\varphi, \hat{\beta}_\delta)$.

   if $\hat{\beta}_\varphi \geq \beta_\varphi$ then solve $\hat{\mathcal{T}}^*(\psi, \tau^*)$ and return $\psi\ R^*_\beta\,(\sigma_\varphi, min\{\tau^*, \beta_\varphi\})$. Otherwise

4. solve $\hat{\mathcal{B}}^*(\delta\sigma_\varphi, \beta_\delta)$, resulting either $\delta\ R^*_\beta\,(\sigma_\varphi\sigma_\delta, \beta_\delta)$ or $\delta\sigma_\varphi\ F^*_\beta$.

   if $\delta\sigma_\varphi\ F^*_\beta$ then return $\psi\ F^*_\beta$.

5. if $\delta\ R^*_\beta\,(\sigma_\varphi\sigma_\delta, \beta_\delta)$ then solve $\hat{\mathcal{T}}^*(\psi, \tau^*)$ and return $\psi\ R^*_\beta\,(\sigma_\varphi\sigma_\delta, min\{\tau^*, \beta_\varphi, \beta_\delta\})$.

To solve $\alpha\hat{\mathcal{B}}^*(\varphi, \beta_\varphi)$:

1. if $\varphi = true$ then return $true\ R^*_\beta\,(yes, 1)$.

2. if $\varphi =$ positive literal q then find one expression E in P such that

   (a) $E = p_i \leftarrow \psi_i$ and $\exists\sigma_i = mgu(q, p_i)$ and $\hat{\mathcal{B}}^*(\psi_i\sigma_i, \beta^*_i)$, or

   (b) $E = \hat{\mathcal{B}}^*(p_i, \beta^*_i)$ and $\exists\sigma_i = mgu(q, p_i)$.

   if there is no such E then return $q\ F^*_\beta$. Otherwise return β^*_i.

3. if $\varphi =$ negative ground literal $\neg g$ then solve $\hat{\mathcal{B}}_*(g, \beta_g)$.

   if $g\ F_{\beta*}$ then return $\tau_\varphi = 1$.

   if $g\ R_{\beta*}\,(\sigma_g, 1)$ then return $\neg g F^*_\beta$.

   if $g\ R_{\beta*}\,(\sigma_g, \beta_g), \beta_g < 1$ then return $\beta_\varphi = 1 - \beta_g$.

- Conditional Beliefs:

  Given a program P and a query $\hat{\mathcal{B}}_*(\psi_1|\psi_2, \beta_*)$:

  1. solve $\hat{\mathcal{B}}_*((\psi_1, \psi_2), \beta'_*)$ and $\hat{\mathcal{B}}^*(\psi_2, \beta^*)$.
  2. if $\beta^* = 0$ then return $\beta_* = 0$. Otherwise
  3. return $\beta_* = \dfrac{\beta'_*}{\beta^*}$.

  Given a program P and a query $\hat{\mathcal{B}}^*(\psi_1|\psi_2, \beta^*)$:

  1. solve $\hat{\mathcal{B}}^*((\psi_1, \psi_2), \beta'^*)$ and $\hat{\mathcal{B}}_*(\psi_2, \beta_*)$.

2. if $\beta_* = 0$ then return $\beta^* = 0$. Otherwise

3. return $\beta^* = \dfrac{\beta'^*}{\beta_*}$.

- Probabilities and Conditional Probabilities on the Domain:

  as presented in chapter 5.4.

The $\alpha - \beta$ pruning strategy does not reduce the worst-case time complexity of our language, since it can be totally ineffective depending on the order in which clauses are selected for unification. It should reduce the average time complexity, however, as the following example illustrates:

Example 6.1 *Consider the following program:*

$p_1(x) \leftarrow p_2(x), \neg p_3(x).$
$p_4(x) \leftarrow p_3(x).$
$p_5(x) \leftarrow p_4(x), \neg p_1(x).$
$p_6(x) \leftarrow p_1(x).$
$p_7(x) \leftarrow p_5(x).$
$p_6(x) \leftarrow p_2(x), \neg p_7(x).$

$\mathcal{T}_*(p_2(a), 0.4).$
$\mathcal{T}_*(p_3(a), 0.2).$
$\mathcal{T}_*(p_3(b), 0.1).$

$\mathcal{T}^*(p_2(a), 0.4).$
$\mathcal{T}^*(p_3(a), 0.2).$
$\mathcal{T}^*(p_3(b), 0.1).$

And the query:

$\mathcal{T}_*(p_6(a), \tau_*).$

Assuming that the and/or-tree is generated depth-first, left-to-right and according to the order in which the program is written, the extract from the complete tree which is generated in order to evaluate the query is the one underlined in fig. 6.1.

However, if we change the order of the literals in the program, we can loose the advantage of not having to generate the complete deduction tree for the query. If the initial program is:

$p_1(x) \leftarrow p_2(x), \neg p_3(x).$
$p_4(x) \leftarrow p_3(x).$
$p_5(x) \leftarrow p_4(x), \neg p_1(x).$
$p_6(x) \leftarrow p_1(x).$
$p_7(x) \leftarrow p_5(x).$
$\boldsymbol{p_6(x) \leftarrow \neg p_7(x), p_2(x).}$

$\mathcal{T}_*(p_2(a), 0.4).$
$\mathcal{T}_*(p_3(a), 0.2).$
$\mathcal{T}_*(p_3(b), 0.1).$

$\mathcal{T}^*(p_2(a), 0.4).$
$\mathcal{T}^*(p_3(a), 0.2).$
$\mathcal{T}^*(p_3(b), 0.1).$

Then the tree to be generated is the one in fig. 6.2.

6.3 Computational Optimisation II: Caching

When the evaluation of a clause is required in different points of a program, a sub-tree is generated repeatedly to resolve a query. If the generation of this sub-tree is computationally costly, it can be worthwhile to store the result of its evaluation the first time it is generated in order to avoid its recalculation.

Whenever a (sub)query containing only ground terms and starting with $\mathcal{P}_*$, $\mathcal{P}^*$, $\mathcal{T}_*$, $\mathcal{T}^*$, $\mathcal{B}_*$ or $\mathcal{B}^*$ is evaluated, we can store its value and possibly avoid redundant computations.

As in the previous case, the improvements on efficiency achieved by applying this optimisation depend on the characteristics of specific programs. Storing and recalling values also imply computational costs, and the trade-off is advantageous only when these costs are smaller than the costs of generating the sub-trees which evaluation is being stored[1].

Example 6.2 *Consider the program presented in example 6.1 and the query:*

[1] recent results presented in [CG92] suggest that the caching strategy can be applied in a selective way, optimising the advantages of using it

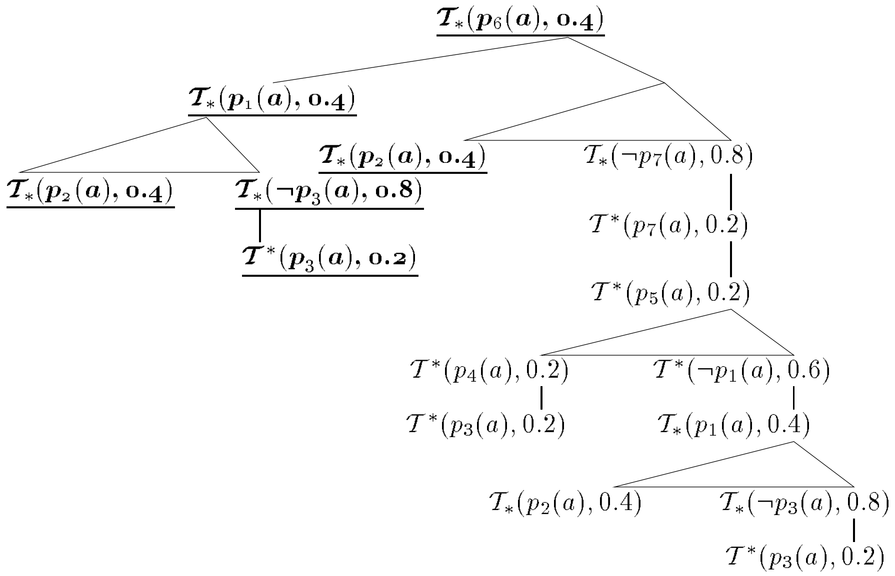

Figure 6.1: and/or-tree for $p_6(a)$ generated by $\alpha - \beta$ procedure

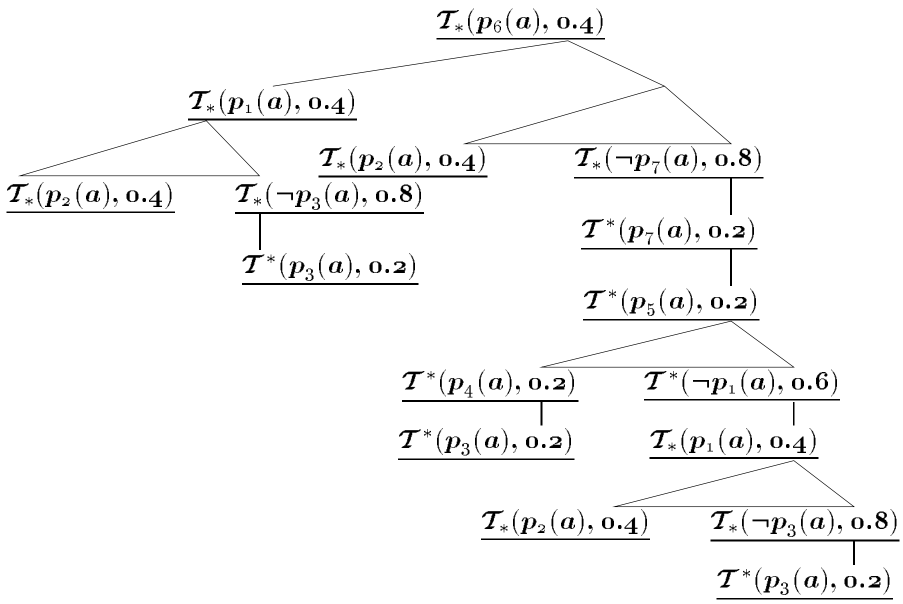

Figure 6.2: and/or-tree for $p_6(a)$ generated by $\alpha - \beta$ procedure - ineffective case

$\mathcal{T}_*(p_6(a), \tau_*)$.

Assuming that the and/or-tree is generated depth-first, left-to-right, the extract from the complete tree which is generated in order to evaluate the query is underlined in fig. 6.3.

6.4 Combining Optimisations I and II

The optimisations above can be combined in a single procedure. An interesting observation about this combination is that the optimisations compete with each other, hence the combined increased efficiency is not a linear function of the improvements obtained from each strategy. This is due to the fact that caching exploits the use of previously stored evaluations, while $\alpha - \beta$ pruning avoids the generation of these very evaluations to be stored.

The simple programs below illustrate the comparative gains in efficiency given by $\alpha - \beta$ pruning, by caching and by the combination of both, with respect to the original "non-optimised" version. A single program is given to which a variable number of unit clauses is added, and the time efficiency of each version of the language for solving specific queries is plotted as a function of the number of unit clauses.

Example 6.3 *Consider the following programs:*

- P_1:
  $p_1(x) \leftarrow p_2(x), \neg p_3(x).$
  $p_4(x) \leftarrow p_3(x).$
  $p_5(x) \leftarrow p_4(x), \neg p_1(x).$
  $p_6(x) \leftarrow p_1(x).$
  $p_7(x) \leftarrow p_5(x).$
  $p_6(x) \leftarrow p_2(x), \neg p_7(x).$

  $\mathcal{T}_*(p_3(a1), 0.1).$
  $\mathcal{T}^*(p_3(a1), 0.4).$
  $\mathcal{B}_*(p_3(a1), 0.2).$
  $\mathcal{B}^*(p_3(a1), 0.3).$

  $\mathcal{P}([a1], 1.0).$

- P_2:

$p_1(x) \leftarrow p_2(x), \neg p_3(x).$
$p_4(x) \leftarrow p_3(x).$
$p_5(x) \leftarrow p_4(x), \neg p_1(x).$
$p_6(x) \leftarrow p_1(x).$
$p_7(x) \leftarrow p_5(x).$
$p_6(x) \leftarrow p_2(x), \neg p_7(x).$

$\mathcal{T}_*(p_3(a1), 0.1).$ $\mathcal{T}_*(p_3(a2), 0.1).$
$\mathcal{T}^*(p_3(a1), 0.4).$ $\mathcal{T}^*(p_3(a2), 0.4).$
$\mathcal{B}_*(p_3(a1), 0.2).$ $\mathcal{B}_*(p_3(a2), 0.2).$
$\mathcal{B}^*(p_3(a1), 0.3).$ $\mathcal{B}^*(p_3(a2), 0.3).$

$\mathcal{P}([a1], 0.5).$ $\mathcal{P}([a2], 0.5).$

- P_3:

$p_1(x) \leftarrow p_2(x), \neg p_3(x).$
$p_4(x) \leftarrow p_3(x).$
$p_5(x) \leftarrow p_4(x), \neg p_1(x).$
$p_6(x) \leftarrow p_1(x).$
$p_7(x) \leftarrow p_5(x).$
$p_6(x) \leftarrow p_2(x), \neg p_7(x).$

$\mathcal{T}_*(p_3(a1), 0.1).$ $\mathcal{T}_*(p_3(a2), 0.1).$ $\mathcal{T}_*(p_3(a3), 0.1).$
$\mathcal{T}^*(p_3(a1), 0.4).$ $\mathcal{T}^*(p_3(a2), 0.4).$ $\mathcal{T}^*(p_3(a3), 0.4).$
$\mathcal{B}_*(p_3(a1), 0.2).$ $\mathcal{B}_*(p_3(a2), 0.2).$ $\mathcal{B}_*(p_3(a3), 0.2).$
$\mathcal{B}^*(p_3(a1), 0.3).$ $\mathcal{B}^*(p_3(a2), 0.3).$ $\mathcal{B}^*(p_3(a3), 0.3).$

$\mathcal{P}([a1], 0.3).$ $\mathcal{P}([a2], 0.3).$ $\mathcal{P}([a3], 0.4).$

- P_4:

$p_1(x) \leftarrow p_2(x), \neg p_3(x).$
$p_4(x) \leftarrow p_3(x).$
$p_5(x) \leftarrow p_4(x), \neg p_1(x).$
$p_6(x) \leftarrow p_1(x).$
$p_7(x) \leftarrow p_5(x).$
$p_6(x) \leftarrow p_2(x), \neg p_7(x).$

$\mathcal{T}_*(p_3(a1), 0.1).$ $\mathcal{T}_*(p_3(a2), 0.1).$ $\mathcal{T}_*(p_3(a3), 0.1).$
$\mathcal{T}^*(p_3(a1), 0.4).$ $\mathcal{T}^*(p_3(a2), 0.4).$ $\mathcal{T}^*(p_3(a3), 0.4).$
$\mathcal{B}_*(p_3(a1), 0.2).$ $\mathcal{B}_*(p_3(a2), 0.2).$ $\mathcal{B}_*(p_3(a3), 0.2).$
$\mathcal{B}^*(p_3(a1), 0.3).$ $\mathcal{B}^*(p_3(a2), 0.3).$ $\mathcal{B}^*(p_3(a3), 0.3).$

$\mathcal{T}_*(p_3(a4), 0.1).$
$\mathcal{T}^*(p_3(a4), 0.4).$
$\mathcal{B}_*(p_3(a4), 0.2).$
$\mathcal{B}^*(p_3(a4), 0.3).$

$\mathcal{P}([a1], 0.3)$. $\mathcal{P}([a2], 0.3)$. $\mathcal{P}([a3], 0.2)$.
$\mathcal{P}([a4], 0.2)$.

- P_5:

  $p_1(x) \leftarrow p_2(x), \neg p_3(x)$.
  $p_4(x) \leftarrow p_3(x)$.
  $p_5(x) \leftarrow p_4(x), \neg p_1(x)$.
  $p_6(x) \leftarrow p_1(x)$.
  $p_7(x) \leftarrow p_5(x)$.
  $p_6(x) \leftarrow p_2(x), \neg p_7(x)$.

  $\mathcal{T}_*(p_3(a1), 0.1)$. $\mathcal{T}_*(p_3(a2), 0.1)$. $\mathcal{T}_*(p_3(a3), 0.1)$.
  $\mathcal{T}^*(p_3(a1), 0.4)$. $\mathcal{T}^*(p_3(a2), 0.4)$. $\mathcal{T}^*(p_3(a3), 0.4)$.
  $\mathcal{B}_*(p_3(a1), 0.2)$. $\mathcal{B}_*(p_3(a2), 0.2)$. $\mathcal{B}_*(p_3(a3), 0.2)$.
  $\mathcal{B}^*(p_3(a1), 0.3)$. $\mathcal{B}^*(p_3(a2), 0.3)$. $\mathcal{B}^*(p_3(a3), 0.3)$.

  $\mathcal{T}_*(p_3(a4), 0.1)$. $\mathcal{T}_*(p_3(a5), 0.1)$.
  $\mathcal{T}^*(p_3(a4), 0.4)$. $\mathcal{T}^*(p_3(a5), 0.4)$.
  $\mathcal{B}_*(p_3(a4), 0.2)$. $\mathcal{B}_*(p_3(a5), 0.2)$.
  $\mathcal{B}^*(p_3(a4), 0.3)$. $\mathcal{B}^*(p_3(a5), 0.3)$.

  $\mathcal{P}([a1], 0.2)$. $\mathcal{P}([a2], 0.2)$. $\mathcal{P}([a3], 0.2)$.
  $\mathcal{P}([a4], 0.2)$. $\mathcal{P}([a5], 0.2)$.

and the queries:

$$
\begin{array}{lll}
Q_1 : (& \mathcal{T}_*(p_7(a1), \tau_*), & \mathcal{T}^*(p_7(a1), \tau^*), \\
& \mathcal{B}_*(p_7(a1), \beta_*), & \mathcal{B}^*(p_7(a1), \beta^*), \\
& \mathcal{B}_*(p_7(a1)|p_4(a1), \beta^c_*), & \mathcal{B}^*(p_7(a1)|p_4(a1), \beta^{c*})).
\end{array}
$$

$$
\begin{array}{lll}
Q_2 : (& \mathcal{T}_*(p_7(x1), \tau^x_*), & \mathcal{T}^*(p_7(x2), \tau^{x*}), \\
& \mathcal{B}_*(p_7(x3), \beta^x_*), & \mathcal{B}^*(p_7(x4), \beta^{x*}), \\
& \mathcal{B}_*(p_7(x5)|p_4(x5), \beta^{xc}_*), & \mathcal{B}^*(p_7(x6)|p_4(x6), \beta^{xc*})).
\end{array}
$$

$$
\begin{array}{lll}
Q_3 : (& \mathcal{T}_*(p_7(a1), \tau_*), & \mathcal{T}^*(p_7(a1), \tau^*), \\
& \mathcal{B}_*(p_7(a1), \beta_*), & \mathcal{B}^*(p_7(a1), \beta^*), \\
& \mathcal{B}_*(p_7(a1)|p_4(a1), \beta^c_*), & \mathcal{B}^*(p_7(a1)|p_4(a1), \beta^{c*}), \\
& \mathcal{P}_*([x7], p_7(x7), \rho_*), & \mathcal{P}^*([x8], p_7(x8), \rho^*), \\
& \mathcal{P}_*([x9], p_7(x9)|p_4(x9), \rho^c_*), & \mathcal{P}^*([x10], p_7(x10)|p_4(x10), \rho^{c*})).
\end{array}
$$

$$
\begin{array}{lll}
Q_4 : (& \mathcal{T}_*(p_7(x1), \tau^x_*), & \mathcal{T}^*(p_7(x2), \tau^{x*}), \\
& \mathcal{B}_*(p_7(x3), \beta^x_*), & \mathcal{B}^*(p_7(x4), \beta^{x*}), \\
& \mathcal{B}_*(p_7(x5)|p_4(x5), \beta^{xc}_*), & \mathcal{B}^*(p_7(x6)|p_4(x6), \beta^{xc*}), \\
& \mathcal{P}_*([x7], p_7(x7), \rho_*), & \mathcal{P}^*([x8], p_7(x8), \rho^*), \\
& \mathcal{P}_*([x9], p_7(x9)|p_4(x9), \rho^c_*), & \mathcal{P}^*([x10], p_7(x10)|p_4(x10), \rho^{c*})).
\end{array}
$$

$$
\begin{array}{lll}
Q_5 : (& \mathcal{T}_*(p_7(a1), \tau_*), & \mathcal{T}^*(p_7(a1), \tau^*), \\
& \mathcal{B}_*(p_7(a1), \beta_*), & \mathcal{B}^*(p_7(a1), \beta^*), \\
& \mathcal{B}_*(p_7(a1)|p_4(a1), \beta_*^c), & \mathcal{B}^*(p_7(a1)|p_4(a1), \beta^{c*}), \\
& \mathcal{T}_*(p_7(x1), \tau_*^x), & \mathcal{T}^*(p_7(x2), \tau^{x*}), \\
& \mathcal{B}_*(p_7(x3), \beta_*^x), & \mathcal{B}^*(p_7(x4), \beta^{x*}), \\
& \mathcal{B}_*(p_7(x5)|p_4(x5), \beta_*^{xc}), & \mathcal{B}^*(p_7(x6)|p_4(x6), \beta^{xc*}), \\
& \mathcal{P}_*([x7], p_7(x7), \rho_*), & \mathcal{P}^*([x8], p_7(x8), \rho^*), \\
& \mathcal{P}_*([x9], p_7(x9)|p_4(x9), \rho_*^c), & \mathcal{P}^*([x10], p_7(x10)|p_4(x10), \rho^{c*})).
\end{array}
$$

We have the execution times as presented in tables 6.1, 6.2, 6.3, 6.4 and 6.5 and figures 6.4, 6.5, 6.6, 6.7 and 6.8. The experiments were run using an Edinburgh-PROLOG implementation running on SPARC workstations. V_1 *corresponds to the non-optimised language,* V_2 *corresponds to the version including caching,* V_3 *corresponds to* $\alpha - \beta$ *pruning and* V_4 *corresponds to the version combining both strategies. The execution times are presented in seconds.*

The number of unit clauses in the program is irrelevant for the solution of query Q_1*, which is a ground query. Moreover, in this case both optimisations are effective and their combination is advantageous over each of them separately. caching is not effective for solution of queries* Q_2 *and* Q_4*, which do not have ground conjuncts. The employment of* $\alpha - \beta$ *pruning is effective, but the combination of* $\alpha - \beta$ *pruning and caching decreases the efficiency obtained when using purely the* $\alpha - \beta$ *pruning strategy. For general queries (i.e. containing ground and non-ground conjuncts) like query* Q_3*, the employment of the combination of both optimisation strategies can be still advantageous, as both strategies are effective. For general queries like query* Q_5*, however, the employment of the combination of both strategies may be less effective than the employment of one of them individually.*

Generally speaking, the use of the optimised versions of the language is useful to improve the time efficiency of execution of programs, as every combination of the explored strategies caused an improvement over the non-optimised version of the language.

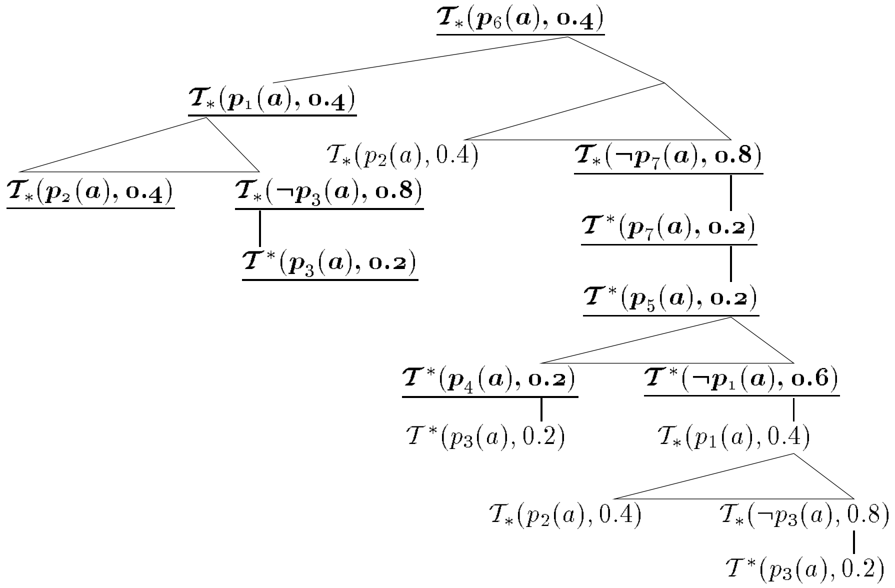

Figure 6.3: and/or-tree for $p_6(a)$ generated by caching procedure

	V_1	V_2	V_3	V_4
P_1	*10.9*	*3.3*	*2.4*	*1.6*
P_2	*11.3*	*3.3*	*2.5*	*1.6*
P_3	*11.1*	*3.4*	*2.4*	*1.6*
P_4	*11.1*	*3.3*	*2.6*	*1.6*
P_5	*11.3*	*3.4*	*2.6*	*1.7*

Table 6.1: Experimental Results - Execution Times for Resolving Query Q_1

	V_1	V_2	V_3	V_4
P_1	*11.1*	*11.5*	*2.5*	*4.1*
P_2	*17.8*	*17.8*	*4.0*	*6.5*
P_3	*24.8*	*24.1*	*5.6*	*9.2*
P_4	*31.5*	*30.4*	*7.2*	*12.0*
P_5	*38.8*	*37.1*	*9.0*	*14.9*

Table 6.2: Experimental Results - Execution Times for Resolving Query Q_2

	V_1	V_2	V_3	V_4
P_1	*17.0*	*8.7*	*3.1*	*2.5*
P_2	*21.6*	*11.3*	*3.8*	*3.1*
P_3	*26.2*	*14.3*	*4.6*	*3.7*
P_4	*30.8*	*17.1*	*5.3*	*4.4*
P_5	*35.7*	*20.2*	*6.1*	*5.2*

Table 6.3: Experimental Results - Execution Times for Resolving Query Q_3

	V_1	V_2	V_3	V_4
P_1	*17.6*	*16.8*	*3.3*	*4.7*
P_2	*28.5*	*25.6*	*5.4*	*7.9*
P_3	*40.0*	*34.7*	*7.6*	*11.2*
P_4	*51.6*	*44.0*	*10.0*	*14.7*
P_5	*63.6*	*53.6*	*12.6*	*18.3*

Table 6.4: Experimental Results - Execution Times for Resolving Query Q_4

	V_1	V_2	V_3	V_4
P_1	*27.9*	*22.0*	*5.5*	*6.3*
P_2	*39.5*	*31.4*	*7.9*	*9.7*
P_3	*50.7*	*40.3*	*10.1*	*12.8*
P_4	*62.6*	*49.8*	*12.6*	*16.3*
P_5	*76.1*	*60.5*	*15.2*	*19.9*

Table 6.5: Experimental Results - Execution Times for Resolving Query Q_5

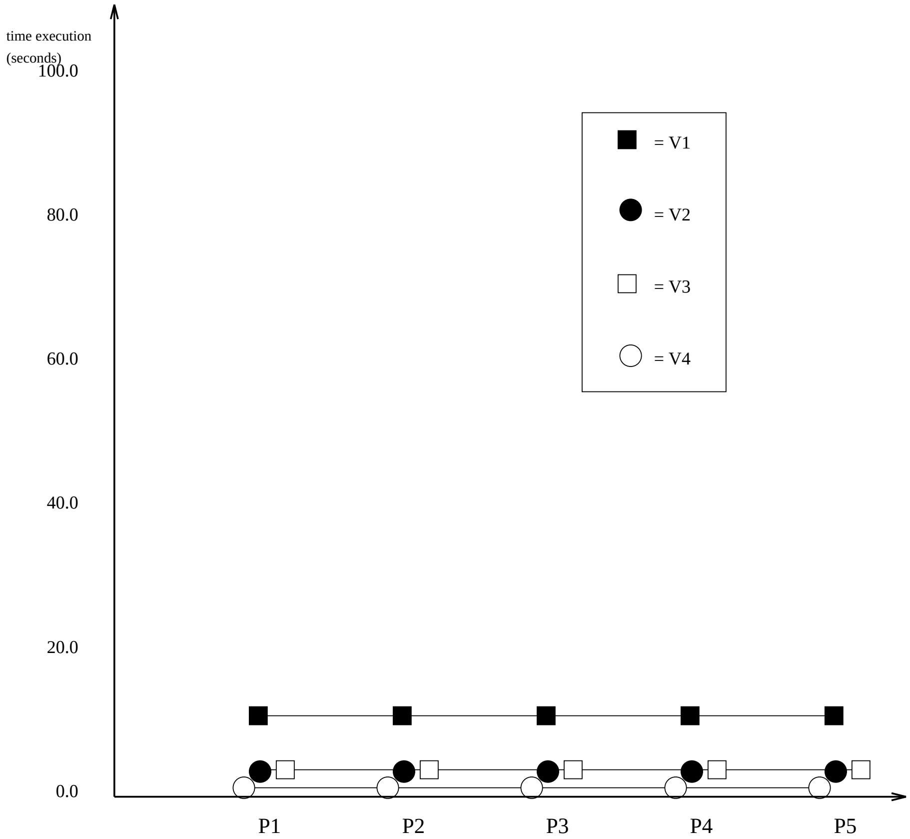

Figure 6.4: Experimental Results - Execution Times for Resolving Query Q_1

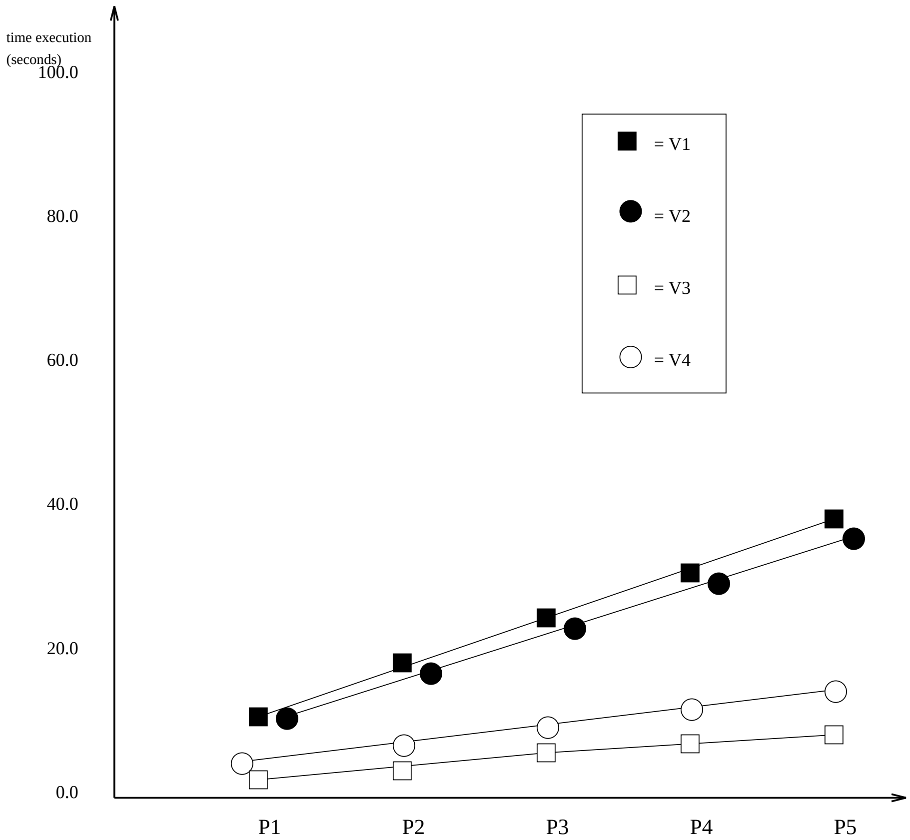

Figure 6.5: Experimental Results - Execution Times for Resolving Query Q_2

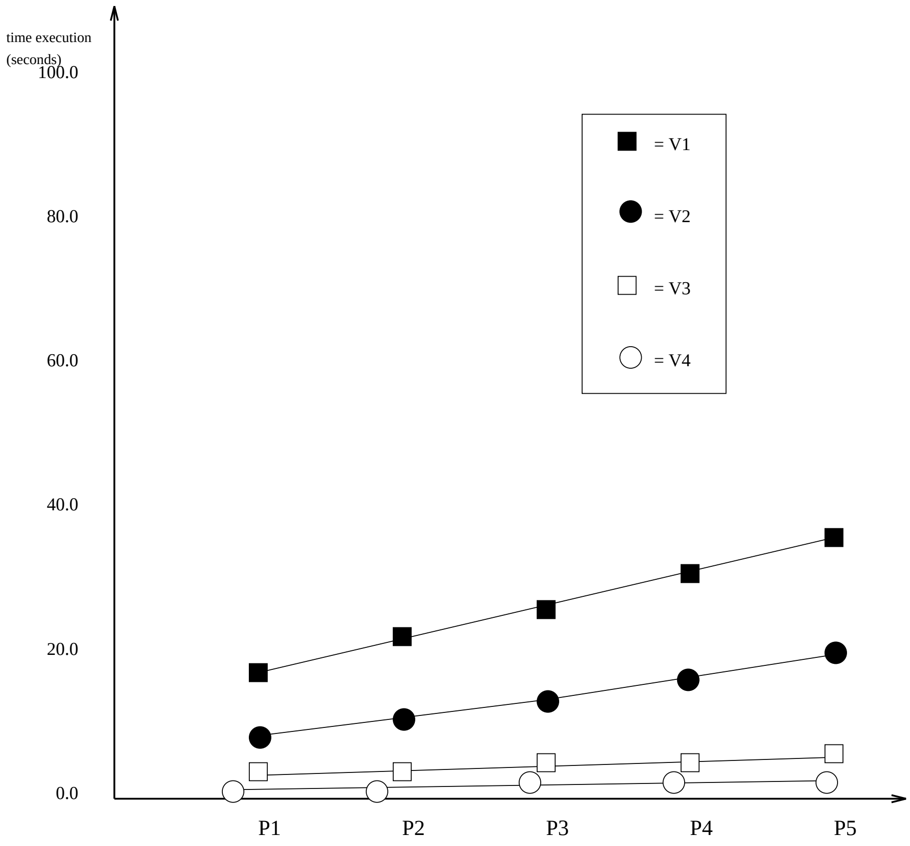

Figure 6.6: Experimental Results - Execution Times for Resolving Query Q_3

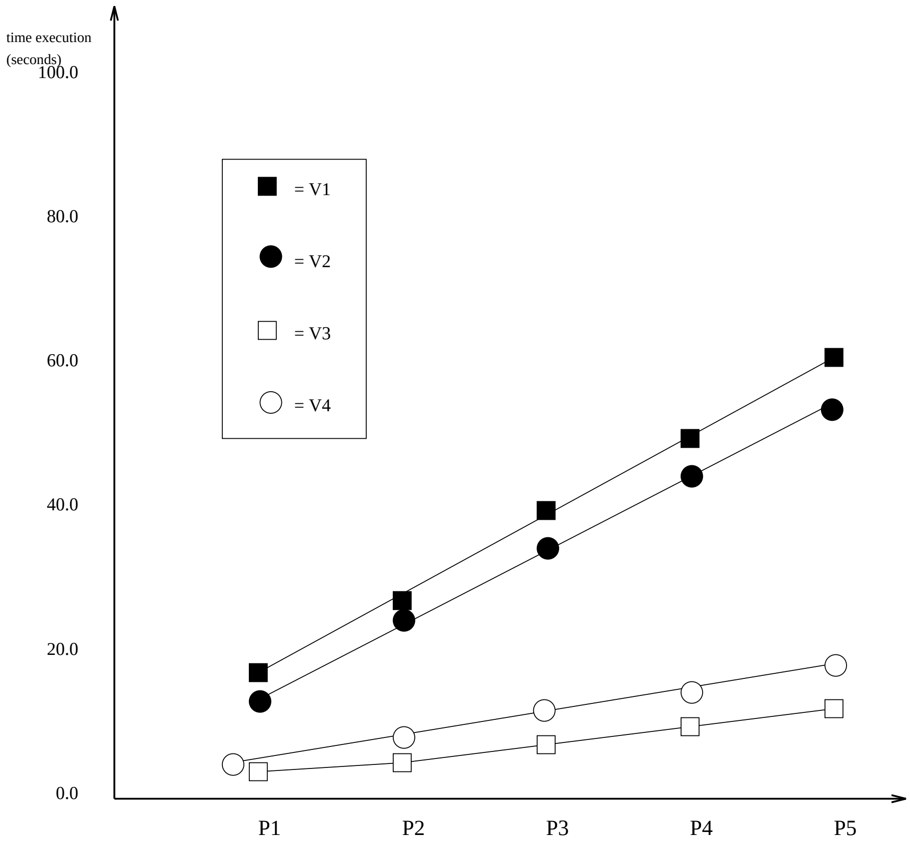

Figure 6.7: Experimental Results - Execution Times for Resolving Query Q_4

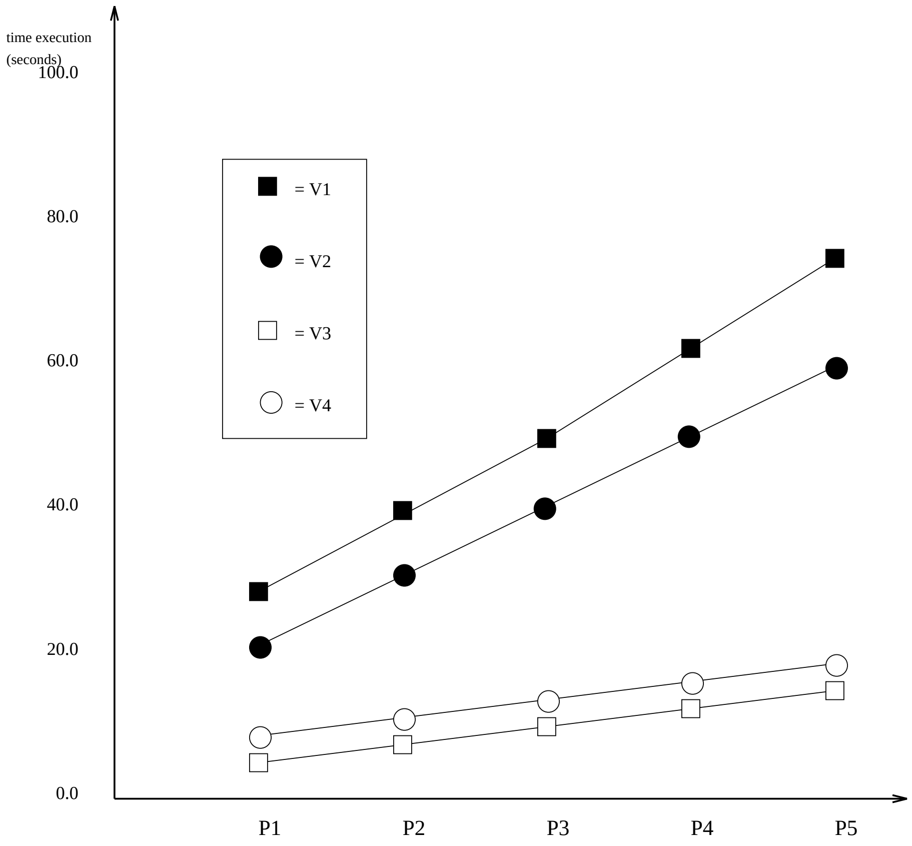

Figure 6.8: Experimental Results - Execution Times for Resolving Query Q_5

7

Summary and Discussion

There has been a lot of debate on which formalism to measure uncertainty is the most general, and many researchers have recently defended the view that there is not a most general formalism but that different formalisms are better to measure different facets of uncertainty.

In the present work we adopted the latter view. We also avoided the simplification that *a single* facet of uncertainty should be selected at the end, therefore accepting that multiple measures could have to be considered within a single representation language.

Assuming this point of view, we explored the feasibility of performing automated reasoning about a domain containing more than one facet of uncertainty by (i) selecting three of these facets and their corresponding measures; (ii) incorporating them to a resolution-based, first-order, clausal theorem prover; and (iii) implementing this theorem prover as a PROLOG meta-interpreter.

The selected measures were:

- fuzzy truth-values, characterising degrees of truth for vague predicates,
- probabilities on the domain, characterising statistical relations among terms of the language, and
- probabilities on possible worlds, characterising degrees of belief on sentences.

An important aspect of any knowledge representation scheme for automated reasoning is having a clearly and rigorously specified semantics for its expressions and operations, so we were careful about providing a model theory for our language and guaranteeing the soundness of its inference procedures.

The main expected contribution of this work was the evidential proof that multiple measures of uncertainty are useful in knowledge representation and inference, and that they can (and should) be treated conjointly within a single representation language. Nevertheless, we believe that the language which was constructed and implemented to constitute this proof presents interest in itself as the prototype of a language to implement knowledge-based systems about domains pervaded with uncertainty. With this in mind, we explored some possibilities to improve its computational efficiency in time, with the positive results presented in chapter 6.

One aspect of our language is the variable coarseness of the results it produces. The language is a proper extension to several simpler theorem provers (e.g. Lee's language [Lee72], Halpern's logic [Hal90], the logic *Lp* [Bac88, Bac90a], Nilsson's logic [Nil86]), and when "projected" to one of these languages it produces results at least as precise as those (i.e. if the result is an interval, it is going to be at least as tight as the one produced by the simpler language). In those more complex cases in which the extensions are needed, however, the uncertainty intervals generated by our language grow rapidly in width. It remains as a topic for further research whether we can specify particular classes of problems with special structural properties such that more precise results (i.e. tighter intervals) can be obtained.

Another limitation of the language is the presupposition of a single source of information for a program (i.e. a single agent to assign belief and truth-degrees to expressions), and the consideration of only those problems which can be treated monotonically. It remains as an open question whether a richer language, capable of treating non-monotonicities and multiple agents (which can be independent, partially dependent or totally dependent), can be constructed in such a way that it has a clear declarative semantics and is computationally tractable.

The main constraints in the expressive power of the language come from the structural restrictions that had to be imposed on the construction of programs in order to preserve the clarity of specification of its declarative semantics, namely the conditions that programs should be *(i) normal, (ii) non-cyclical, (iii) strict with respect to queries, (iv) allowed,* and *(vi) with the declaration of truth and belief values restricted to unit clauses.* It is well-known from the literature that some of these constraints can be lifted when not all of the three measures studied here are used within a single domain, but it is yet another open question whether the constraints could be relaxed with all measures being present, and under what conditions that would be the case.

Yet, we believe that the results presented here are a positive empirical confirmation that multiple representations of uncertainty can be incorporated into a single language in a systematic way.

It should be stressed that no previous implementations involving these three uncertainty measures could be found in the literature. As we mentioned before, in [Hal90, Bac88, Bac90c] we have theoretical results proving the computability of specific versions of their languages involving probabilities on the domain and on possible worlds, but implementation and computational efficiency issues are not treated there.

Implementation issues are considered in the works of Dubois and Prade involving measures on possible worlds and fuzzy predicates (cf. [DP87, DP88]), but those authors use possibilistic rather than probabilistic measures on possible worlds, which are more amenable to implementation although not as expressive as probabilistic measures of the form of Dempster-Shafer structures (cf. chapter 5.1 for a discussion on how Dempster-Shafer structures subsume possibilistic measures).

Finally, although the concept of probability of a fuzzy event was established in [Zad68], its explicit use in knowledge representation languages to describe the expected truth-degree of a query containing fuzzy predicates and free variables could not be found in previous references.

Future work includes pursuing completeness results for the procedures presented here, and the construction of more refined implementations of the programming language proposed for multiple representations of uncertainties.

Bibliography

[AH89] M. Abadi and J. Y. Halpern. Decidability and Expressiveness for First-Order Logics of Probability. Technical Report RJ-7220, IBM Research Report, 1989.

[Apt87] K. F. Apt. Introduction to Logic Programming. Technical Report CS-R8741, Centre for Mathematics and Computer Science, 1987.

[Bac88] F. Bacchus. *Representing and Reasoning with Probabilistic Knowledge.* PhD thesis, University of Alberta, 1988.

[Bac90a] F. Bacchus. Lp, a Logic for Representing and Reasoning with Statistical Knowledge. *Computational Intelligence*, 6:209–231, 1990.

[Bac90b] F. Bacchus. On Probability Distributions over Possible Worlds. In *Uncertainty in Artificial Intelligence 4*, 1990.

[Bac90c] F. Bacchus. *Representing and Reasoning with Probabilistic Knowledge.* MIT Press, 1990.

[Bun85] A. Bundy. Incidence Calculus: a Mechanism for Probabilistic Reasoning. *Journal of Automated Reasoning*, 1(3):263–284, 1985.

[Bun86] A. Bundy. Correctness Criteria of Some Algorithms for Uncertain Reasoning Using Incidence Calculus. *Journal of Automated Reasoning*, 2(1):109–126, 1986.

[Bun90] A. Bundy. Incidence Calculus. Technical Report 497, University of Edinburgh, Department of Artificial Intelligence, 1990.

[CdS91] F. S. Correa da Silva. Automated Reasoning About an Uncertain Domain. Technical report, University of Edinburgh, Department of Artificial Intelligence (a preliminary version of this work was presented at the European Conference on Symbolic and Quantitative Aspects of Uncertainty, under the same title, authored by Correa da Silva, F. S., Robertson, D., and Chung, P.), 1991.

[CdSB90] F. S. Correa da Silva and A. Bundy. On Some Equivalence Relations Between Incidence Calculus and Dempster-Shafer Theory of Evidence. In 6^{th} *Conference on Uncertainty in Artificial Intelligence*, 1990.

[CdSB91] F. S. Correa da Silva and A. Bundy. A Rational Reconstruction of Incidence Calculus. Technical Report 517, University of Edinburgh, Department of Artificial Intelligence, 1991.

[CG92] M. D. Chaudri and R. Greiner. A Formal Analysis of Solution Caching. In *Proceedings of the 9^{th} Canadian Conference on Artificial Intelligence*, 1992.

[Che88] P. Cheeseman. An Inquiry into Computer Understanding. *Computational Intelligence*, 4:58–66, 1988.

[CL89] L. Cavedon and J. W. Lloyd. A Completeness Theorem for SLDNF Resolution. *Journal of Logic Programming*, 7:177–191, 1989.

[Cla88] D. A. Clark. Responses to 'An AI View of the Treatment of Uncertainty' by Alessandro Saffiotti. *The Knowledge Engineering Review*, 3:59–86, 1988.

[Cla90] D. A. Clark. Numerical and Symbolic Approaches to Uncertainty Management in AI. *Artificial Intelligence Review*, 4:109–146, 1990.

[DP87] D. Dubois and H. Prade. Necessity Measures and the Resolution Principle. Technical Report 267, LSI - Universite Paul Sabatier, 1987.

[DP88] D. Dubois and H. Prade. An Introduction to Possibilistic and Fuzzy Logics. In P. Smets, A. Mamdani, D. Dubois, and H. Prade, editors, *Non-standard Logics for Automated Reasoning.* Academic Press, 1988.

[DP89] D. Dubois and H. Prade. Fuzzy Sets, Probability and Measurement. *European Journal of Operational Research*, 40:135–154, 1989.

[Dud89] R. M. Dudley. *Real Analysis and Probability.* Wadsworth & Brooks/Cole, 1989.

[FH89a] R. Fagin and J. Y. Halpern. A New Approach to Updating Beliefs. Technical Report RJ-7222, IBM Research Report, 1989.

[FH89b] R. Fagin and J. Y. Halpern. Uncertainty, Belief, and Probability. Technical Report RJ-6191, IBM Research Report, 1989.

[FHM90] R. Fagin, J. Y. Halpern, and N. Megiddo. A Logic for Reasoning about Probabilities. *Information and Computation*, 87:78–128, 1990.

[Fit85] M. Fitting. A Kripke-Kleene Semantics for Logic Programs. *Journal of Logic Programming*, 4:295–312, 1985.

[Fit88] M. Fitting. Logic Programming on a Topological Bilattice. *Fundamenta Informaticae*, XI:209–218, 1988.

[Fit90] M. Fitting. Bilattices in Logic Programming. In *Proceedings of the 20^{th} International Symposium on Multiple-valued Logic*, 1990.

[GS82] G. R. Grimmett and D. R. Stirzaker. *Probability and Random Processes.* Oxford Sci. Publ, 1982.

[Hal90] J. Y. Halpern. An Analysis of First-Order Logics of Probability. *Artificial Intelligence*, 46:311–350, 1990.

[Hin86] C. J. Hinde. Fuzzy Prolog. *International Journal of Man-Machine Studies*, 24:569–595, 1986.

[Hog90] C. J. Hogger. *Essentials of Logic Programming.* Oxford Univ. Press, 1990.

[IK77] S. Iyanaga and Y. (eds) Kawada. *Encyclopedic Dictionary of Mathematics.* MIT Press, 1977.

[IK85] M. Ishizuka and K. Kanai. Prolog-ELF Incorporating Fuzzy Logic. In *IJCAI'85 - Proceedings of the 9^{th} International Joint Conference on Artificial Intelligence*, 1985.

[KJ87] H. E. Kyburg Jr. Bayesian and non-Bayesian Evidential Updating. *Artificial Intelligence*, 31:271–293, 1987.

[KJ90] H. E. Kyburg Jr. Uncertainty Logics. Technical Report TR 337, University of Rochester, Dept. of Computer Science, 1990.

[Kle82] E. P. Klement. Construction of Fuzzy σ-algebras Using Triangular Norms. *Journal of Mathematical Analysis and Applications*, 85:543–565, 1982.

[KS91] M. Kifer and V. S. Subrahmanian. Theory of Generalized Annotated Logic Programs and its Applications. *Journal of Logic Programming (forthcoming)*, 1991.

[Kun87] K. Kunen. Negation in Logic Programming. *Journal of Logic Programming*, 4:289–308, 1987.

[Kun89] K. Kunen. Signed Data Dependencies in Logic Programs. *Journal of Logic Programming*, 7:231–245, 1989.

[Kun90] K. Kunen. Some Remarks on the Completed Database. *Fundamenta Informaticae*, XIII:35–49, 1990.

[Lee72] R. C. T. Lee. Fuzzy Logic and the Resolution Principle. *Journal of the ACM*, 19:109–119, 1972.

[Mel85] C. S. Mellish. Generalised Alpha-Beta Pruning as a Guide to Expert Systems Question Selection. In *Expert Systems '85 - Proceedings of the 5^{th} Technical Conference of the British Computer Society, Specialist Group on Expert Systems*, 1985.

[Men87] E. Mendelson. *Introduction to Mathematical Logic (3rd. ed).* Wadsworth & Brooks/Cole, 1987.

[Nil86] N. J. Nilsson. Probabilistic Logic. *Artificial Intelligence*, 28:71–87, 1986.

[NS92] R. Ng and V. S. Subrahmanian. Probabilistic Logic Programming. *Information and Computation (forthcoming)*, 1992.

[Orc89] I. P. Orci. Programming in Possibilistic Logic. *International Journal of Expert Systems*, 2:79–96, 1989.

[Pea88] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, 1988.

[Pia88] K. Piasecki. Fuzzy p-Measures and their Application in Decision Making. In J. Kacprzyk and M. Fedrizzi, editors, *Combining Fuzzy Imprecision with Probabilistic Uncertainty in Decision Making.* Springer Verlag, 1988.

[Rus87] E. H. Ruspini. The Logical Foundations of Evidential Reasoning. Technical Report 408, SRI International, 1987.

[Rus89] E. H. Ruspini. On the Semantics of Fuzzy Logic. Technical Report 475, SRI International, 1989.

[Rus90a] E. H. Ruspini. Approximate Reasoning: Past, Present, Future. Technical Report 492, SRI International, 1990.

[Rus90b] E. H. Ruspini. Possibility as Similarity: The Semantics of Fuzzy Logic. In 6^{th} *Conference on Uncertainty in Artificial Intelligence*, 1990.

[Saf87] A. Saffiotti. An AI View of the Treatment of Uncertainty. *The Knowledge Engineering Review*, 2(2):75–97, 1987.

[Sha76] G. Shafer. *A Mathematical Theory of Evidence.* Princeton Univ. Press, 1976.

[Sha83] E. Y. Shapiro. Logic Programming with Uncertainties - a Tool for Implementing Rule-based Systems. In *IJCAI'83 - Proceedings of the 8^{th} International Joint Conference on Artificial Intelligence*, 1983.

[Sha87] G. Shafer. Belief Functions and Possibility Measures. In J. C. Bezdek, editor, *Analysis of Fuzzy Information.* CRC Press, 1987.

[Sho67] J. R. Shoenfield. *Mathematical Logic.* Addison-Wesley, 1967.

[Sme82] P. Smets. Probability of a Fuzzy Event: An Axiomatic Approach. *Fuzzy Sets and Systems*, 7:153–164, 1982.

[Som90] Léa Sombé. Reasoning under incomplete information in artificial intelligence: a comparison of formalisms using a single example. *International Journal of Intelligent Systems*, 5, 1990.

[Tur88] I. B. Turksen. Stochastic Fuzzy Sets: a Survey. In J. Kacprzyk and M. Fedrizzi, editors, *Combining Fuzzy Imprecision with Probabilistic Uncertainty in Decision Making.* Springer Verlag, 1988.

[Tur89] D. Turi. Logic Programs with Negation: Classes, Models, Interpreters. Technical Report CS-R8943, Centre for Mathematics and Computer Science, 1989.

[vE86] M. H. van Emden. Quantitative Deduction and its Fixpoint Theory. *Journal of Logic Programming*, 1:37–53, 1986.

[Zad68] L. Zadeh. Probability Measures of Fuzzy Events. *Journal of Mathematical Analysis and Applications*, 23:421–427, 1968.

[Zad88] L. Zadeh. Fuzzy Logic. Technical Report CSLI-88-116, Center for the Study of Language and Information, 1988.