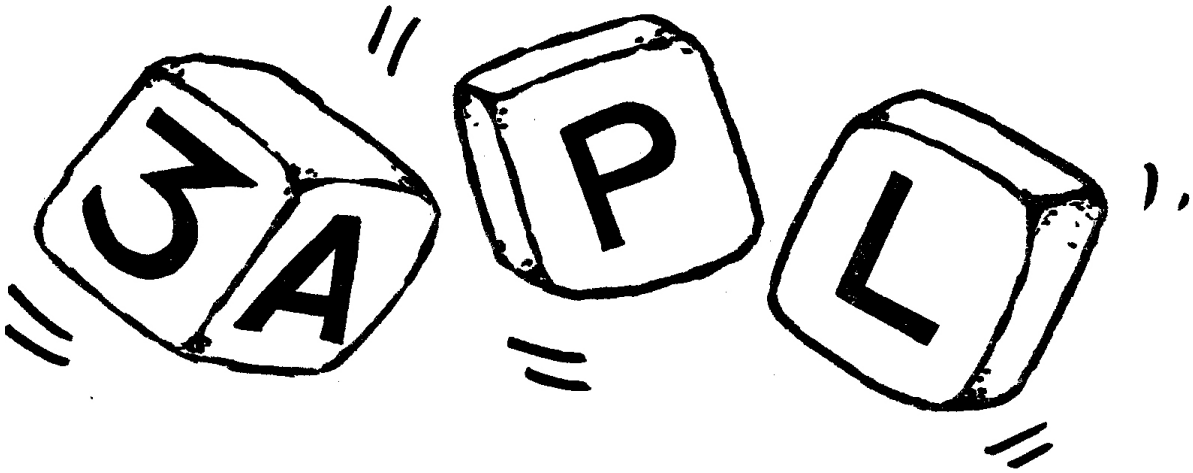
Master Thesis

Modelling Uncertainty in 3APL



Johan Kwisthout
studentnummer 837078259

Open Universiteit
Masteropleiding Technical Informatics
August 1, 2005

Institute

Universiteit Utrecht
Institute of Information and Computing Sciences
Intelligent Systems Group

Supervising committee

prof. dr. J.T. Jeuring (Open Universiteit)
dr. ir. S.E. de Vos (Open Universiteit)
dr. M.M. Dastani (Universiteit Utrecht)

Acknowledgements

I would like to thank my supervising committee and fellow students, both at the artificial intelligence-thesis group at the Open Universiteit, and my collegue-thesis students in 'room 142' in Utrecht, for their support, thoughtful critique and interest. I owe especially much to Schil de Vos and Mehdi Dastani.

Schil, having supervised tens of students at the Open Universiteit, knows probably better than anyone else out there what it takes to combine thesis research, a job and a family. As a mentor, Schil did his best to guide me through the bureaucracy an university sometimes can be.

Mehdi was more than 'just' a daily supervisor. He learned me the tricks of the trade of writing an academic paper, showed genuine interest in my work and greatly stimulated me to pursue an academic career. I hope we have ample opportunity to continue working together.

Some people greatly influenced my research interests, most probably without knowing. Although I hesitate to list such great and renowned researchers in this thesis, I feel I owe credit to Luc Steels, whose book 'Kennissystemen' first introduced me to Artificial Intelligence; Paul Churchland, Jerry Fodor and John Searle, whose disputes helped me to form my own opinion on Artificial Intelligence and how it is to be accomplished, and Arthur C. Clarke and Stanley Kubrick (and of course HAL) for inspiration and imagination.

Last, but certainly not least, I thank Judith for her love and caring. Hopefully I have some more spare time now to spend together.

Contents

Summary

In this thesis, I investigate how the agent programming language 3APL can be enhanced to model uncertainty. Typically, agent programming languages such as 3APL that are based on beliefs, goals and intentions use logical formulae to represent their beliefs and reason on them. These formulae are either true or false (i.e. they are believed or not), and this limits the use of such agent programming languages in practical applications. While a lot of research has been done on the topic of reasoning with uncertainty the possible use of these methods in the field of agent programming languages such as 3APL has not been given much attention.

I investigate several methods (with a focus on Bayesian networks and Dempster-Shafer theory), and show that Dempster-Shafer theory is a promising method to use in agent programming. Particulary appealing in this theory is the ability to model *ignorance*, as well as uncertainty. Nevertheless, the combinatorial explosion of its combination rule and the issue of inconsistency (which are addressed in the thesis) are serious disadvantages of this theory for its practical application to agent programming.

I investigate a possible mapping of Dempster-Shafer sets to belief formulae in 3APL. With restrictions on the mass functions and on the frame of discernment, Dempster-Shafer theory is a convenient way to model uncertainty in agent beliefs. Because, with certain restrictions, mass values can be computed based on the beliefs in the belief base, we do not need to keep a combined mass function of n beliefs in memory and update it with each belief update. Therefore there is no combinational explosion.

I propose a syntax and semantics for 3APL with uncertainty, and demonstrate a prototype Prolog implementation of the calculation of the certainty of a logical formula given a certain belief base.

Samenvatting

In deze scriptie onderzoek ik hoe de agent-programmeertaal 3APL uitgebreid kan worden om met onzekerheid te kunnen werken. Agent-programmeertalen, zoals 3APL, die gebaseerd zijn op *beliefs*, *goals* en *intentions* gebruiken over het algemeen logische formules om hun *beliefs* te representeren en er mee te redeneren. Deze formules zijn binair (ze zijn waar of niet), en dit vormt een beperking in het gebruik van dit soort programmeertalen voor practische applicaties. Hoewel er veel onderzoek is gedaan naar methoden voor het redeneren onder onzekerheid, is er weinig aandacht besteed aan het mogelijk toepasbaar zijn van deze methoden in agent-programmeertalen zoals 3APL.

Ik beschrijf verschillende methoden (met de nadruk op Bayesiaanse netwerken en Dempster-Shafer theorie), en toon aan dat Dempster-Shafer theorie voor deze toepassing een veelbelovende methode is. Met name de mogelijkheid om *onwetendheid* te representeren, naast *onzekerheid*, maakt Dempster-Shafer theorie aantrekkelijk voor het gebruik in agent-programmeertalen. De combinatorische explosie van de combinatieregel om verschillende aanwijzingen te combineren, en het probleem met mogelijke inconsistentie tussen deze aanwijzingen, zijn echter serieuze nadelen van deze theorie voor het gebruik in de praktijk.

Ik heb Dempster-Shafer sets gebruikt om onzekere belief formules in 3APL te representateren. Met beperkingen aan de *mass functies* en aan het *frame of discernment* is Dempster-Shafer theorie een goed toepasbare methode om onzekere beliefs te modelleren. Omdat de mass berekend wordt op basis van de beliefs in de belief base is het niet nodig om een gecombineerde mass functie in het geheugen opgeslagen te hebben en met iedere belief update aan te passen. Het probleem met de combinatorische explosie van de combinatieregel is daarmee omzeild.

Ik stel een syntax en semantiek voor 3APL met onzekerheid voor, en demonstreer een prototype implementatie in Prolog voor de berekening van de waarschijnlijkheid van een logische formule op basis van een belief base.

Chapter 1

Introduction

For a lot of complex tasks, traditional design and programming paradigms, like imperative or object-oriented programming, fail to describe and analyze the task at hand in an intuitive way. An autonomous robot, the outcome of an electronic auction, or the behavior of an intelligent enemy in a role-playing game are examples of tasks that can be modelled more intuitively by using the *cognitive agent* metaphor. Although there is no definition to which all researchers subscribe, one of the most often used definitions is proposed by Wooldridge: a cognitive agent is equipped with high-level concepts such as beliefs, desires, intentions, obligations, and commitment, and has pro-active and re-active capabilities. They are situated in some environment, and are capable of autonomous actions in the environment in order to achieve their objectives [28]. Cognitive agents and their applications are hot research-topics in artificial intelligence, logics, and cognitive science, to name but a few.

An example of a cognitive agent might be a sort of *personal digital assistant* (PDA) that searches the Internet for relevant articles and news, communicates with other PDAs, notices the user when important information is available etcetera. Its characteristics satisfy Wooldridges definition: in order to be at use the agent must be active in a dynamic environment (the Internet), be both pro-active and re-active (has certain intentions), and perform autonomous actions, based on its perception of the environment (its beliefs), the wants and needs of the user (its desires).

1.1 Specifying and programming agents

Cognitive agents are often specified with epistemic or dynamical logics, the first specifying how agents can reason with their beliefs, the latter specifying temporal concepts. An often used approach that combines both is Rao and Georgeffs BDI$_{CTL}$model [21]. In this model, a Computation Tree Logic[1] is used to describe the world as a *time tree*, with a branching future and a single past. Each branch denotes a *choice* that the agent makes, a single point in the time tree is called a *situation*. With every situation, Rao associates a set of *belief-accessible worlds*, the worlds the agent believes to be possible at that given point in the time tree. Because of the agent's lack of knowledge there are multiple belief-accessible worlds, with a *probability distribution*[2] amongst the worlds, which representates the likeliness of each world. There are also goal-accessible and intention-accessible worlds associated with every point in the time tree. So, at every situation the agent has a set of beliefs, a set of goals (or desires) and a set of intentions. In figure 1.1 an example of such a time tree is given. At a certain point in the timetree, the agent has chosen to take the train. At this point, the train might be late or not; depending on this situation the time tree branches in separate ways, indicating the agent has different choices depending on the actual state of the world.

In order to develop agent systems, many programming languages have been proposed to implement individual agents, their environments, and interactions, based on the BDI$_{CTL}$ paradigm. Examples of such languages are 3APL, AgentSpeak, Jack, and Jadex [9, 15, 18, 4]. These languages provide programming constructs to enable the implementation of agents that can reason about their information and objectives and update them according to their interactions. 3APL (short for An Abstract Agent Programming Language) was developed by the Intelligent Systems Group at Utrecht University. It bridges the gap that existed between an agent's specification (using beliefs, desires and intentions) and the actual implementation. 3APL will be discussed in chapter 2.

Unfortunately, many of the proposed programming languages, including 3APL, assume that the information and objectives of agents are certain, which is obviously an unrealistic assumption for

[1] See the appendix on Modal Logics

[2] This concept will be formally defined in definition 10 in chapter 3

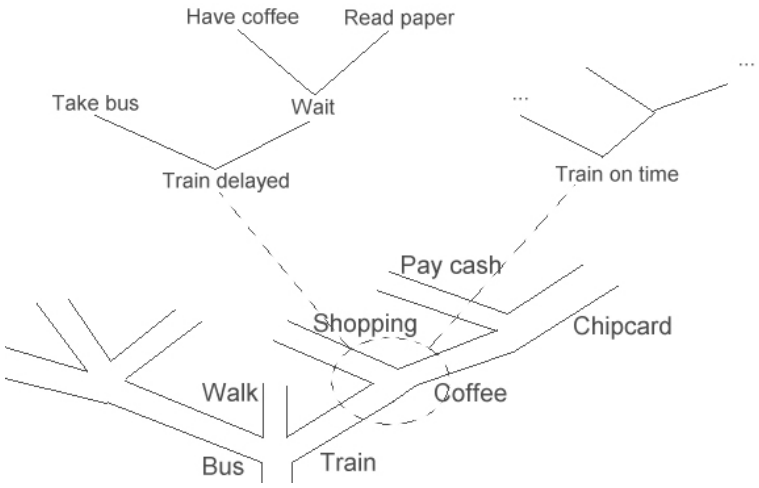


Figure 1.1: An example of a BDI$_{CTL}$ time tree

many real world applications. In such applications, either the environment of the agents involves uncertainty or the uncertainty is introduced to agents through imperfect sensory information. Past research dealing with the application of 3APL for robot control [26] showed that sensory input is not always accurate, and that external actions have unpredictable outcomes: the environment in which the agent operates is both inaccessible and nondeterministic. This seriously devalues the practical use of 3APL as a programming language for real world applications.

1.2 Research question

In this thesis, I investigate how 3APL can be enhanced to be able to deal with uncertainty. I discuss several methods for dealing with uncertainty, and argue why Dempster-Shafer theory is the best method to be used to implement uncertain beliefs. I show how the belief base can be queried and updated, and propose a syntax and semantics for 3APL with uncertainty. I suggest how these uncertain beliefs can refine the deliberation process, and I present a Prolog prototype for the implementation of uncertain beliefs.

1.3 Uncertainty in agent programming

Although Rao and Georgeff[21] explicitly state that a BDI$_{CTL}$ agent has multiple belief-accessible worlds with a probability distribution associated with them, little research has been done on the question how a transition from one belief-accessible world to another, as a result of an action that the agent performs, influences that probability distribution. Most research on uncertainty focusses on implementing the concept in modal logics on one side, or practical implementations like expert systems (which often use a more or less 'works-for-me'-method) on the other side. The first often lack production rules that can combine uncertain beliefs, the latter often lack a thorough theoretical background.

Milch and Koller [14] proposed a probabilistic epistemic logic and an algorithm for asserting and querying formulae in Bayesian networks. However, there agent model is different to Rao's BDI$_{CTL}$ model. Instead of beliefs, desires and intentions, their agents are based on beliefs and decisions. They also assume that agents have a common prior probability distribution over states of the world, and that the distribution at any state (at any point of the CTL timetree, in Rao's terms), is equal to the global distribution conditioned on the set of states the agent considers possible at this state. This contrasts with Rao's idea of a timetree with a branching future, where actions change the set of belief accessible worlds without constraints.

Parsons and Giorgini [19] consider quantifying an agent's beliefs using Dempster-Shafer theory. While based on the BDI model, their agents are built using multi-context systems (see [20] for a discussion on multi-context systems) and therefore not directly applicable in 3APL which uses basic actions and practical reasoning rules for belief and goal update.

1.4 Thesis outline

The outline of the thesis is structured as follows:

- Chapter 2 describes the agent programming language 3APL, both informally and formally, presenting the syntax, semantics and transition system of the 3APL language and the architecture of the 3APL platform;
- Chapter 3 provides an overview of reasoning with uncertainty and describes a number of methods that were investigated (in particular Dempster-Shafer theory and Bayesian networks), and discusses the arguments for using the Dempster-Shafer theory of evidence;
- Chapter 4 describes how concepts from Dempster-Shafer theory can be related to 3APL beliefs. I discuss the mapping of 3APL beliefs to Dempster-Shafer sets of hypotheses, the issue of inconsistency, the frame of discernment, calculation of mass functions, and updating and querying the belief base;
- In chapter 5, I propose altered formal definitions for the syntax and semantics of 3APL, enhanced with uncertainty in the belief base;
- Chapter 6 discusses a Prolog prototype implementation of the calculation of mass functions and updating and querying of the belief base.
- Chapter 7 concludes the research project and discusses further work. In particular, I discuss the consequences of uncertain beliefs for the agent's deliberation cycle.

I presume that the reader has a working knowledge of (modal) logic at the level of the Open Universiteit-course *Logica en Informatica*, and a working knowledge of Prolog. An introduction to modal logic is presented in the appendix. A short introduction in complexity theory is also provided in the appendix, mainly to explain the concepts *NP-complete*, *P-complete*, and *# P-complete* used in this thesis.

Chapter 2

3APL syntax and semantics

In this chapter, I will introduce 3APL as a platform from a programmer's view, and give formal definitions of an 3APL agent, as proposed by Dastani et al.[9]. 3APL was designed by the Intelligent Systems Group of the University of Utrecht, with the purpose of bridging the gap between agent specification and agent programming. Concepts described here, both informally and with their formal definitions [1], include beliefs, goals, basic actions, practical reasoning rules and the deliberation cycle.

2.1 Beliefs, goals, basic actions and practical reasoning rules

A 3APL program consists of a belief base with logical facts that are believed by the agent, a goal base with logical facts that are to be accomplished, a set of basic actions that change the belief base and a set of practical reasoning rules that change the goal base. 3APL follows the Prolog syntax, i.e. facts and goals start with a lower case letter, variables start with an upper case letter. Basic actions and practical reasoning rules start in upper, respectively lower case. Both have their optional parameters in parentheses.

The *belief base* is a set of logical formulae, which represent the beliefs the agent has regarding the environment in which it operates. The beliefs are implemented as Prolog clauses. Using some examples from Winograds blockworld (from the famous SHRDLU program), some beliefs might be:

```
on(a, b).
on(b, c).
clear(c).
```

Which represent the beliefs, that object b is placed on a, c is placed on b and nothing is placed on c. A *basic action* updates the belief base, and is specified with a precondition and a postcondition as follows:

```
{on(A, B), clear(C)} Move(C, B) {clear(A), on(C, B), NOT on(A, B), NOT clear(C)}
```

This basic action represents the moving of one object to another: if B is on top of A, and there is nothing on top of C, than we can move B to be on top of C. As a result of this, there is nothing on top of A, and B is on top of C. B is no longer on top of A and C is not clear anymore [2].

The *goal base* is a set of goals that the agent tries to accomplish. A goal can be a test on the belief base, an `IF..THEN..ELSE` clause, a `WHILE..DO` clause, an abstract goal (comparable to procedure call in imperative languages) or compositions of the former. Goals in 3APL are procedural[3] rather than declarative, meaning a goal describes a procedure or *plan* to follow, rather than a desired state. Some examples:

```
on(A, C)
clear(C)?
IF clear(C) THEN Move(C, B) ELSE Move(D, B)
WHILE NOT clear(C) DO Move(table, C)
on(A, C); on(B, A)
```

[1] More information on formal definitions can be found in the appendix on modal logics
[2] This example was inspired by Monty Python's Committee for Putting Things on Top of Other Things
[3] In the version of 3APL used in this thesis

A *practical reasoning rule* revises goals in the goal base. It is specified by a list of goals (optional if we want to add goals), a *guard*, and a list of basic actions and new goals (optional if we want to remove goals). In this way we can model recursion, revision, reactive behavior and removal.

`walk()` ← ¬`wall` | `Stepforward;walk()`

This rule reasons about the *walk()* goal: it instructs the agent to take a step forward until it bumps into a wall. This is an example of recursion.

`takeBus();takeTrain()` ← `strike` | `takeTaxi();takeTrain()`

This is an example of a revision rule. It states, that the goal sequence *takeBus();takeTrain()* is to be replaced by *takeTaxi();takeTrain()* if there is a strike going on.

← `hungry` | `eat()`

This is an example of a rule that models reactive behavior. If the agent acquires the belief that it is hungry at any point, it adds the goal *eat()* to the goal base.

`eat()` ← ¬`hungry` |

In this last example, we see how a goal can be removed. The goal *eat()* is to be removed (replaced by the empty goal) if the agent beliefs it is not hungry.

Using these concepts and some other language constructs for the sake of readability (like e.g. `BEGIN` and `END` keywords, the `AND` and `NOT` keywords, and comments), we can build a complete 3APL program, for example the following 'blockworld'-program (figure 2.1). The agent in this program can put blocks on top of another and on the floor, as specified by the basic actions. Furthermore, its goals are to change an initial configuration (as denoted in the initial belief base) to a end-configuration as denoted in the goal-base. Note that the goals in the goal base are procedural. There are two rules in this program, the first one denoting that the agent should do nothing when the desired state is accomplished, the other enhancing the abstract plan with some basic actions to accomplish the desired state.

2.2 Deliberation cycle

The *deliberation cycle* of an agent program determines which reasoning rules are to be selected to update the goals. In the version of 3APL that is used for this thesis, this deliberation cycle is a fixed, non-programmable process: it is hard-coded into the 3APL interpreter as is shown in figure 2.2[4]. The interpreter just picks a rule that is applicable. However, a programmable deliberation cycle has been investigated [8] in order to program the agent's behavior and determine for example which rule should be used if multiple rules are available. This gives the programmer not only control over *what* an agent can do - which is determined in the agent program - but also *how* the agent does it.

2.3 Syntax

In the former sections, I presented the informal notions of beliefs, goals and reasoning rules from a programmer's view. In this section, a relevant section of the underlying logic of 3APL is introduced. Since I concentrate on beliefs in this thesis, only definitions dealing with beliefs and reasoning rules are presented. The formal definitions are taken from [9]. First, I show how the *base language* (the language that can be used to build other constructs) is built from variables and functions, and how the proposition language is built from predicates and this base language. Note that the proposition language only contains atomic formulae (no logical connectives).

[4]This is the deliberation cycle of a version of 3APL enhanced with communication

```
PROGRAM "blockworld"

CAPABILITIES:

    {on(X,Y), NOT on(Y, _), clear(Z)} Move(Z, Y) {clear(X), on(Z, Y), NOT on(X, Y), NOT clear(Z)}
    {NOT on(Y, _), clear(Z)} Move(Z, Y) {on(Z, Y), NOT clear(Z)}
    {on(X, Y), NOT on(Y, _)} Clear(X) {NOT on(X, Y), clear(X), clear(Y)}


BELIEFBASE:

    on(a, b).
    on(b, c).
    clear(c).

GOALBASE:

    transport()

RULEBASE:

     transport() <- on(c, b) AND on(b, a) AND clear(a) | SKIP,
     transport() <- on(P, Q) AND on(Q, R) AND clear(R) |
        BEGIN
            Clear(Q);
            Move(R, Q);
            Move(Q, P);
            transport()
        END.
```

Figure 2.1: An example of a 3APL program

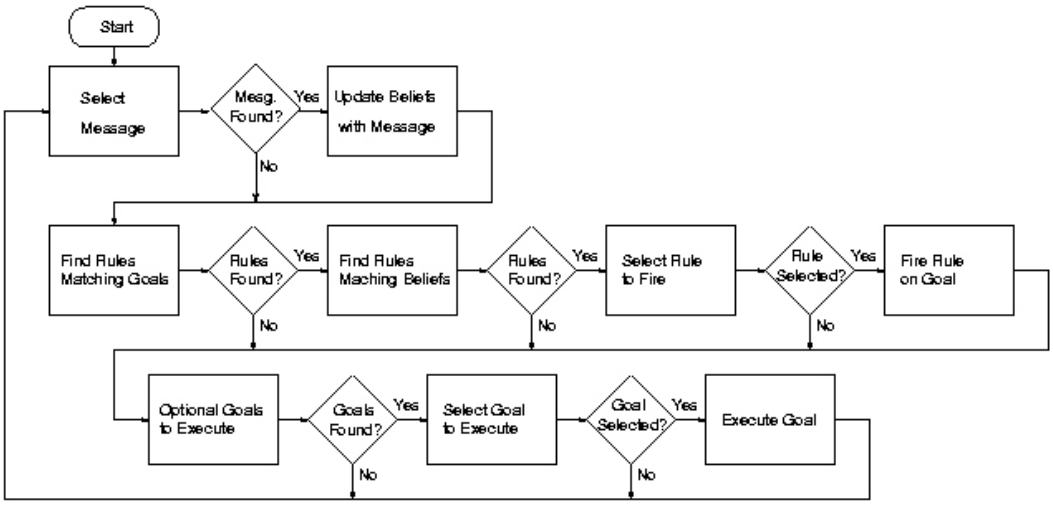


Figure 2.2: 3APL deliberation cycle

Definition 1 *(base language)*
Let VAR be the set of domain variables, FUNC the set of functions (where a function without an argument is a constant), and PRED be the set of predicates. Let $n \geq 0$. The terms T_L of the base language L are defined as follows:

- *If $x \in VAR$, then $x \in T_L$,*
- *If $f \in FUNC$ and $t_1, \ldots, t_n \in T_L$, then $f(t_1, \ldots, t_n) \in T_L$.*

The formulae of the proposition language L_{PRED} contains only atomic formulae, defined as follows:

- *If $p \in PRED$ and $t_1, \ldots, t_n \in T_L$, then $p(t_1, \ldots, t_n) \in L_{PRED}$.*

Furthermore, a 3APL agent has a belief base, defined in terms of a *belief base language*. The referred article[9] uses the notion of a *ground formula*, which is a formula that does not contain variables, and a *closed formula*, which is a formula in which all variables are bound by a quantifier.

Definition 2 *(belief base language)*
The formulae of the belief base language is defined as follows: Let $\psi \in L_{PRED}$ be a ground formula, and let $\phi, \phi_1, \ldots, \phi_n \in L_{PRED}$. The belief language L_B of a 3APL agent are formulae defined as follows:

- $\psi, \forall_{x_1,\ldots,x_n}(\phi_1 \wedge \ldots \wedge \phi_n \rightarrow \phi) \in L_B$

Where $\forall_{x_1,\ldots,x_n}(\varphi)$ denotes the universal closure of the formula φ for every variable $x_1, \ldots, x_n$ occurring in φ.

Using this definition, we can incorporate Prolog facts and clauses (which are to be unified, hence the ground formulae) in the belief base. Apart from this belief base language, a belief query language is defined. Belief queries are used in the reasoning rules (see definition 5) and in the postconditions of basic actions. The query language does not include negation in the base language, which is related to the use of Prolog as a reasoning engine. Since Prolog uses *negation as failure* in its reasoning process, it is impossible to explicitly model a belief as '¬rain'. In contrast, the query $\mathbf{B}$(rain) will return **false** if 'rain' is not among the facts. The query $\neg\mathbf{B}$(rain), on the other hand, will return **true**.

Definition 3 *(belief queries)*
Let L_{PRED} be the base language. Then, the belief query language L_{BQ} with typical formula β is defined as follows:

- *if $\phi_1, \ldots, \phi_n \in L_{PRED}$, then $\mathbf{B}(\phi_1 \wedge \ldots \wedge \phi_n), \neg\mathbf{B}(\phi_1 \wedge \ldots \wedge \phi_n) \in L_{BQ}$,*
- *$\top \in L_{BQ}$,*

Note the difference between the belief language L_B and the belief query language L_{BQ}. The belief language consists of Prolog facts and clauses (in their Horn-clause notation), such as `on(a, b)`, `solitairBlock(X) :- clear(X), not on(_,X)`. The belief query language consists of Prolog queries on the belief base.

The *plan language* is used in constructing the practical reasoning rules. In this plan language we can have basic actions, tests on the belief base, abstract plans (comparable with procedure calls in imperative programming languages), and composite plans. The empty plan is also defined as having no effect. Ultimately, a plan is thus a sequence of basic actions.

Definition 4 *(plans)*
Let $\beta \in L_{BQ}$. The plan language L_P consists of the following elements:

- *empty plan: $E \in L_P$*
- *basic action: $ACT \in L_P$*
- *test: $\beta? \in L_P$*
- *abstract plan: $AP \in L_P$*
- *composite plans: if $\pi_1, \pi_2 \in L_P$, then π_1 ; π_2,* `if` *β* `then` *π_1* `else` *π_2* `fi`*,* `while` *β* `do` *π_1* `od` *$\in L_P$*

Furthermore, if $E, \pi \in L_P$ *it holds that* $E;\pi = \pi;E = \pi$.

Using this plan language, we can define rules to reason about plans, which can be generated, extended or dropped. This can be described as follows:

Definition 5 *(rules)*
Let $\beta \in L_{BQ}$ *and* $\pi_h, \pi_b \in L_P$. *We define plan revision rules PR as follows:*

- $\pi_h \leftarrow \beta \mid \pi_b \in PR$,

Plan revision rules can be used to generate, extend or drop plans by using the following general forms, respectively:

- $E \leftarrow \beta \mid \pi_b$ *for plan generation,*
- $\pi_h \leftarrow \beta \mid \pi_h \wedge \pi_b$ *for plan extension,*
- $\pi_h \leftarrow \beta \mid E$ *for plan removal.*

For example, an example of a *plan revision rule* could be

Stepforward $\leftarrow$ **B**(obstacle) | Stepleft; Stepforward; Stepright.

Having defined these elements of a 3APL program, we can now define a 3APL configuration (consisting of beliefs, plans, rules, and a substitution component) and a 3APL agent, which is effectively a description of its initial plans, beliefs, and reasoning rules.

Definition 6 *(configuration)*
A configuration of a 3APL agent is a tuple $< \sigma, \Pi, \theta >$, *in which* $\sigma \subseteq L_B$ *is the belief base of the agent,* $\Pi \subseteq L_P$ *is the plan base of the agent, and* θ *represents the substitution that binds domain variables to domain terms. The belief base is assumed to be grounded.*

Definition 7 *(3APL agent)*
A 3APL agent is a tuple $< \sigma_0, \Pi_0, PR >$, *where* σ_0 *is the initial belief base,* Π_0 *is the initial plan base, and PR is a set of rule plans.*

2.4 Semantics

In this section I present the semantics of the belief formulae in a 3APL configuration. Informally, the 3APL semantics state, that the agent in a certain 3APL configuration beliefs a formula if the belief base is a model of that belief.

Definition 8 *(semantics of belief formulae)*
Let $< \sigma, \Pi, \theta >$ *be an agent configuration, let* $\phi \in L_B$ *and* $\boldsymbol{B}\phi \in L_{BQ}$. *Then*

- $< \sigma, \Pi, \theta > \models \boldsymbol{B}\phi \iff \sigma \models \phi$

In 3APL, *transition rules* bind the variables in test goals and practical reasoning rules, and can be seen as derivation rules that specify the transition from one state to another (for example by executing a plan). An extensive discussion of this transition system is out of the scope of this thesis; I will give one example of a transition rule that states, that the execution of a set of plans is accomplished by executing each plan separately. The interested reader can find the definitions of all relevant transition rules in [9].

Definition 9 *(transition rule for plan execution)*
Let $\Pi = \{\pi_0, \ldots, \pi_i, \ldots, \pi_n\} \subseteq L_P$, $\Pi' = \{\pi_0, \ldots, \pi_i', \ldots, \pi_n\} \subseteq L_P$, *let* σ *and* σ' *be the agent's beliefs, and let* θ *and* θ' *be ground substitutions. Then the transition rule is formulated as:*

$$\frac{< \{\pi_i\}, \sigma, \theta > \rightarrow < \{\pi_i'\}, \sigma', \theta' >}{< \{\pi_0, \ldots, \pi_i, \ldots, \pi_n\}, \sigma, \theta > \rightarrow < \{\pi_0, \ldots, \pi_i', \ldots, \pi_n\}, \sigma', \theta' >}$$

This rule is to be read as follows. The transition of a tuple $< \Pi, \sigma, \theta >$ to a tuple $< \Pi', \sigma', \theta' >$ can be deducted from the transition of a tuple $< \{\pi_i\}, \sigma, \theta >$ to a tuple $< \{\pi_i'\}, \sigma', \theta' >$

2.5 Research question revisited

3APL does not have a means to reason with uncertain information. In this thesis, I investigate how the belief base of an agent can be modified to hold uncertain beliefs, how these beliefs can be updated and how belief queries can be changed to reason with uncertain beliefs. Apart from syntax, I also investigate a possible semantics for belief base consisting uncertain belief formulae. In the next chapter, I describe various methods for reasoning with uncertainty and argue why Dempster-Shafer theory can be used to model uncertainty in 3APL.

Chapter 3

Reasoning with uncertainty

In real life, omniscient knowledge is rare. People are usually forced to reason with imperfect knowledge, because of a number of reasons, both practically and theoretically. Even if it were possible to know all facts, it would be impossible to store all these facts in memory, judge the relevance of these facts, and reason with them in a bounded time slot. Fortunately, it is not necessary to know all facts to come to a conclusion which satisfies *in most cases*. One could argue it is not desirable either: imagine a surgeon who has to operate on a patient in an emergency situation. If she would take all possibly relevant facts into account, the patient would most probably be deceased before a conclusion could be drawn. Most of the time, however, a restricted set of relevant facts is enough to make a decision.

While human beings normally cope well with these bounded resources, it is very hard to formalize such reasoning in traditional logic. The task of determining which facts are relevant, or, more generally, how to determine which facts stay the same in a changing world, has been labelled as the Frame Problem and has tortured the AI-community since the early seventies (see [16] for an overview), especially those who tried to formalize common-sense reasoning in mathematical logic.

One way to tackle the Frame Problem, is to enhance traditional logic with probabilistic reasoning. Practical AI-systems, such as the MYCIN expert system [5], for example, use a variant of probabilistic reasoning, to allow a certain ignorance to be incorporated into the reasoning engine. Often, practical implementations are also forced to use some sort of imperfect knowledge, because the data and rules they deal with are inconsistent, imprecise or vague. To clarify the concept of imperfect knowledge, one can make a taxonomy of different sources for this imperfection, e.g. ambiguity, vagueness, and uncertainty. A typical example, taken from [25], is presented in figure 3.1. In this thesis only *uncertainty* will be investigated.

The concept *uncertainty* is closely related to probability theory. In this theory, we differentiate between the notion of *chance* and *probability*: a chance represents a objective, statistical likeliness of an event (such as throwing a six with a dice), probability the likeliness of an event, given certain subjective knowledge (for example, the probability of six, given that we know the number is even). This knowledge is called subjective, because different agents can have different knowledge, which influences this likeliness. Probabilistic reasoning deals with the question, how *evidence* influences our *belief* in a certain hypothesis H. We define the probability of H, denoted $P(H)$, as a real number between 0 and 1, with $P(H) = 0$ meaning H is definitely false, and $P(H) = 1$ meaning H is definitely true. A value between 0 and 1 is a measure for the probability of H.

A *probability distribution* assigns values to a set of possible worlds, and is defined as follows:

Definition 10 *(Probability distribution)*

Let Ω be a set of possible worlds. A probability distribution Γ assigns a value μ to every world ϖ in Ω, such that $\mu(\varpi) \geq 0$, and $\sum_{\varpi \in \Omega} \mu(\varpi) = 1$. Furthermore, the probability of a hypothesis H, denoted as $P(H)$, is defined as $\sum_{\varpi \models H} \mu(\varpi)$.

For example, if Ω represents the outcome of the throwing of a fair dice, than the probability distribution $\Gamma = \mu(\varpi) = \frac{1}{6}$ for every world in Ω. The probability of the hypothesis 'outcome is even', $P(\text{even})$, is $\frac{1}{2}$, since H follows from worlds ϖ_2, ϖ_4, and ϖ_6.

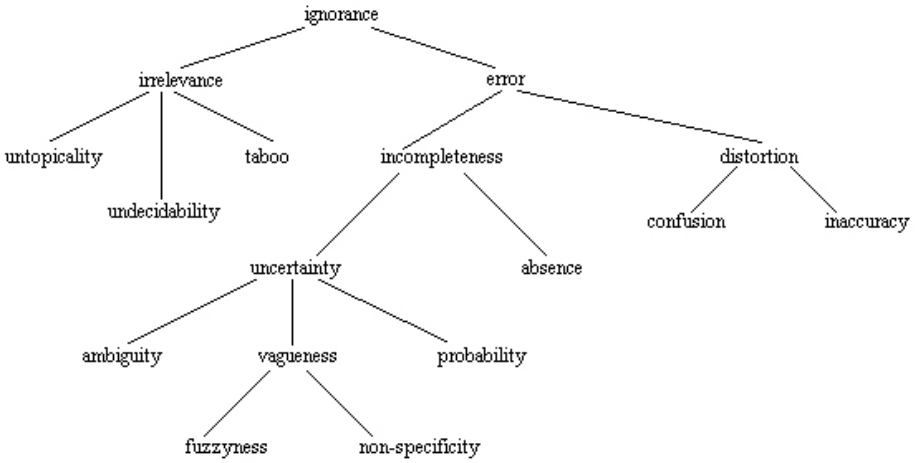


Figure 3.1: Smithson's taxonomy of ignorance

Certain evidence can transform a probability distribution Γ into Γ'. If E is a certain piece of evidence, then Γ' assigns values μ' to every world ϖ in Ω. If we define Ω_E as $\{\varpi \mid \varpi \in \Omega \wedge \varpi \models E\}$, then μ' is related to μ as follows:

Definition 11 *(Change in probability distribution)*

$$\mu'(\varpi) = \begin{cases} \frac{\mu(\varpi)}{\sum_{\varpi \in \Omega_E} \mu(\varpi)} & \textit{for } \varpi \models E \\ 0 & \textit{for } \varpi \not\models E \end{cases}$$

For example, using the same Ω and probability distribution Γ, then evidence $E =$'dice is even' transforms Γ into Γ', where

$$\Gamma' = \begin{cases} \mu(\varpi) = \frac{1}{3} & \text{for } \varpi \in \{2,4,6\} \\ \mu(\varpi) = 0 & \text{for } \varpi \in \{1,3,5\} \end{cases}$$

The probability of H, given evidence E, is noted as $P(H \mid E)$, is called the *a posteriori* or *conditional* probability of H with evidence E, and with $\Omega_{E \wedge H} = \{\varpi \mid \varpi \in \Omega \wedge \varpi \models E \wedge H\}$ this is defined as:

Definition 12 *(probability $P(H \mid E)$)*

$$P(H \mid E) = \frac{1}{P(E)} \sum_{\varpi \in \Omega_{H \wedge E}} \mu(\varpi) = \frac{P(H \wedge E)}{P(E)}$$

In the BDI$_{CTL}$model, as discussed in the introduction, an agent has a set of *belief-accessible worlds* at every point in the time tree: worlds that the agent beliefs to be possible. A probability distribution is associated with such a set, and in this thesis I will investigate several methods to transform the distribution at a certain point, to another, based on the outcome of an action of the agent.

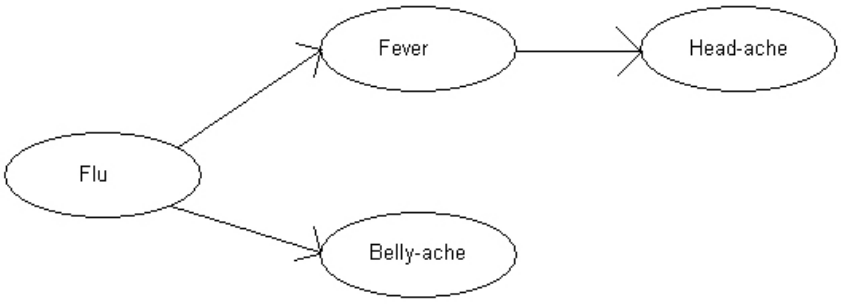


Figure 3.2: An example of a Bayesian Network

3.1 Bayesian networks

A Bayesian or causal network [12] is a set of connected nodes, in which the connections represent causal links between the nodes. A *multivariate distribution* is associated with the network in the form of a posteriori probabilities for each node. An example of a Bayesian network is presented in figure 3.2. A Bayesian network can formally be described as follows.

Definition 13 *(Bayesian Network). A Bayesian network is a tuple* $\beta = (G, P)$*, where*

- $G = (V(G), A(G))$ *is an a-cyclic directed graph, where*
  - $V(G) = \{X_1, X_2, \ldots, X_n\}$ *is a set of nodes, and*
  - $A(G) \subseteq V(G) \cdot V(G)$ *is a set of arrows, and*
- $P : \wp(V(G)) \rightarrow [0..1]$ *is a multivariate distribution, such that* $P(X_1, \ldots, X_n) = \prod_{i=1\ldots n} P(X_i \mid \pi_G(X_i))$*,*
  where $\pi_G(X_i)$ *is the set of immediate predecessors of node* X_i*.*

Typically, a Bayesian network can be divided in a number of relatively independent components. If there is no traversal possible between components, the probabilities of the nodes are assumed to be independent. Certain nodes can be instantiated (an unconditional probability value is associated with them) and the certainty values of the connecting nodes are calculated on the basis of the evidence and the conditional probabilities. Depending on the type of connection between nodes (serial, diverging or converging) evidence propagation can be blocked between certain nodes if evidence becomes available. For example, in figure 3.2 evidence of whether the patient has a fever does not influence the probability of a belly-ache, if we already know that the patient has a flu.

Evidence propagation goes as follows. If we define the conditional probabilities of the network in figure 3.2 as in table 3.1, and we instantiate $P(\text{Headache}) = 0.9$ and $P(\text{Belly ache}) = 0.7$, then we can calculate $P(\text{Fever}) = 0.15 \cdot 0.9 + 0.05 \cdot 0.1 = 0.14$, and $P(\text{Flu}) = 0.4 \cdot (0.7 \cdot 0.14) + 0.05 \cdot (0.7 \cdot 0.86) + 0.2 \cdot (0.3 \cdot 0.14) + 0.01 \cdot (0.3 \cdot 0.86) = 0.34$ using these conditional probabilities and calculation rules for independent probabilities.[1]

Probability	Value	Probability	Value
$P(\text{Flu} \mid \text{Belly ache} \wedge \text{Fever})$	0.4	$P(\text{Flu} \mid \neg\text{Belly ache} \wedge \text{Fever})$	0.2
$P(\text{Flu} \mid \text{Belly ache} \wedge \neg\text{Fever})$	0.05	$P(\text{Flu} \mid \neg\text{Belly ache} \wedge \neg\text{Fever})$	0.01
$P(\text{Fever} \mid \text{Headache})$	0.15	$P(\text{Fever} \mid \neg\text{Headache})$	0.05

Table 3.1: Example of inference in a Bayesian network

This inference needs rather a lot of calculations with every new piece of evidence. In fact, inference in a Bayesian network is an NP-complete problem[7]. An alternative exists in the form of a *message passing*-algorithm. This algorithm sees a node in the network as an object with local calculations, and messages are sent between these nodes if it is necessary to update the network. This algorithm reduces the amount of information that needs to be sent over the network.

[1] $P(A \wedge B) = (P(A) \cdot P(B))$ *iff.* $P(A \mid B) = P(A)$ & $P(B \mid A) = P(B)$ for $P(A), P(B) > 0$.

Table 3.2 shows an informal mapping of the various components of a Bayesian network to terms of the BDI$_{CTL}$ model. Note, that we can only map the set of all nodes in the network to the set of all belief-accessible worlds at a given point in the time-tree, because in a Bayesian network structure all nodes are instantiated. Typically, a set of nodes and their connections are given beforehand, as well as the conditional probabilities, and evidence is represented by instantiation of the nodes in the network.

Bayesian network	BDI$_{CTL}$
Nodes in the network	Set of belief-accessible worlds at any point in the time-tree
Network state	Probability distribution at a given point in the time-tree
Inference	Transition between probability distributions

Table 3.2: Bayesian network vs. BDI$_{CTL}$

3.2 Dempster-Shafer theory

The theory of Dempster and Shafer [23] can be seen as a generalization of the probability theory presented earlier in this chapter. In this theory, a *frame of discernment* Ω is defined as the set of all hypotheses in a certain domain. On the power set 2^{Ω}, a *mass function* $m(X)$ is defined for every $X \subseteq \Omega$, with $m(X) \geq 0$ and $\sum_{X \subseteq \Omega} m(X) = 1$. If there is no information available with respect to Ω, $m(\Omega) = 1$, and $m(X) = 0$ for every subset of Ω.

For example, in a murder case Ω is a list of suspects, {Peter, John, Paul, Mary, Cindy}. If the investigator has no further information, the mass function associated with Ω will assign a value of 1 to Ω and 0 to all subsets of Ω. If there is evidence found regarding certain subsets of Ω, for example a (slightly unreliable) witness claims the killer was probably a male, we assign a mass value to this particular subset of Ω (say 0.6) and - since we have no further information - the remaining mass to Ω. The mass function in this case would be:

$$m_1(X) = \begin{cases} 0.6 & \text{if } X = \{\text{Peter, John, Paul}\} \\ 0.4 & \text{if } X = \Omega \\ 0 & \text{otherwise} \end{cases}$$

Note that no value whatsoever is assigned to *subsets* of {Peter, John, Paul}. If we receive further evidence, for example that the killer was most likely left-handed, and both John and Mary are left-handed, then we might have another mass function like:

$$m_2(X) = \begin{cases} 0.9 & \text{if } X = \{\text{John, Mary}\} \\ 0.1 & \text{if } X = \Omega \\ 0 & \text{otherwise} \end{cases}$$

Dempster's Rule of Combination is a method to combine both pieces of evidence into one combined mass function. This function for the combination of $m_1 \oplus m_2$ is defined as:

Definition 14 *(Dempsters Rule of Combination [23]). Let $X, Y, Z \subseteq \Omega$. Then*

$$m_1 \oplus m_2(X) = \frac{\sum_{Y \cap Z = X} m_1(Y) \cdot m_2(Z)}{\sum_{Y \cap Z \neq \emptyset} m_1(Y) \cdot m_2(Z)} \qquad \textit{and}$$

$$m_1 \oplus m_2(\emptyset) = 0$$

Dempster's Rule of Combination is commutative and associative, as shown in [22]. In our example, combining both pieces of evidence would lead to the following mass function:

$$m_1 \oplus m_2(X) = \begin{cases} 0.06 & \text{if } X = \{\text{Peter, John, Paul}\} \\ 0.36 & \text{if } X = \{\text{John, Mary}\} \\ 0.54 & \text{if } X = \{\text{John}\} \\ 0.04 & \text{if } X = \Omega \\ 0 & \text{otherwise} \end{cases}$$

Given a certain mass function, the subsets of Ω that have a mass value greater than zero are called *focal elements*, and we will denote the set of focal elements of a given mass function φ as the *core* of that mass function. A *simple support function* is a special case of a mass function, where the evidence only supports a certain subset A of Ω, and no non-zero mass value is assigned to any subset of Ω other than A, so that its core is $\{A, \Omega\}$:

Definition 15 *(simple support function [23]). Let $A \subseteq \Omega$ be an evidence with probability s. Then, the simple support function related to A is specified for $X \subseteq \Omega$ as follows:*

$$m(X) = \begin{cases} s & \text{if } X = A \\ 1\text{-}s & \text{if } X = \Omega \\ 0 & \text{otherwise} \end{cases}$$

Both m_1 and m_2 in the example are simple support functions. If we would split the evidence in m_1 as follows, it would no longer be a simple support function:

$$m_{1'}(X) = \begin{cases} 0.4 & \text{if } X = \{\text{Peter, John}\} \\ 0.2 & \text{if } X = \{\text{Paul}\} \\ 0.4 & \text{if } X = \Omega \\ 0 & \text{otherwise} \end{cases}$$

On a mass function, two other functions are defined, namely a *belief function* $Bel(X)$ and a *plausibility function* $Pl(X)$.

Definition 16 *(belief and plausibility function [23]). Let $X, Y \subseteq \Omega$, then the belief and plausibility functions can be defined in terms of the mass function as follows:*

$$Bel(X) = \sum_{Y \subseteq X} m(Y) \qquad \text{and} \qquad Pl(X) = \sum_{X \cap Y \neq 0} m(Y)$$

Informally, the belief and plausibility functions can be seen as a lower respectively upper limit on the probability of the set of hypotheses X. Note, that the plausibility of a certain subset equals 1 minus the belief of the complement of this subset: $Pl(X) = 1 - Bel(\Omega \backslash X)$. The difference between $Bel(X)$ and $Pl(X)$ can be regarded as the *ignorance* with respect to X.

The concepts described above map smoothly to concepts from the BDI$_{CTL}$ model, as shown in table 3.3.

Dempster-Shafer	BDI$_{CTL}$
Frame of discernment	Set of belief-accessible worlds at any point in the time-tree
Mass function	Probability distribution at a given point in the time-tree
Combination rule	Transition between probability distributions

Table 3.3: Dempster-Shafer theory vs. BDI$_{CTL}$

3.3 Possibilistic methods

Examples of possibilistic methods are the Certainty Factor model used in MYCIN [5], Possibilistic Logic models [10] or Mass Assignment Theory [1]. These methods do not model the probability of a certain event, but rather model human uncertainty handling. For example, in the MYCIN project the experts that were consulted assigned uncertainty values to conditional events (like `IF` the patient coughs, `THEN` the patient has a cold) which were contradictory. For example, if we have a statistical distribution of the attributes 'has a cold' and 'coughs' over a population like table 3.4, then we could infer from definition 12 that 'coughs' is an evidence for 'has a cold' with a strength of $0.2 \div 0.35 = 0.57$, a conclusion an expert might support. On the other hand, it can also be interfered that 'coughs' is evidence *against* 'has a cold' with a strength of $0.15 \div 0.35 = 0.43$. Buchanan and Shortliffe concluded, that experts do not use these statistical probabilities in practice, but rather use and *increase* or *decrease* of trust in a certain hypothesis. This notion can lead to unexpected

consequences, for example that one hypothesis can have a higher certainty factor than another, while it's probability is lower.

Although these methods have their application in e.g. expert systems that are based on a model of a human expert, 3APL is based on logic programming, and an extension of 3APL should have a firm theoretical base in logic as well. Therefore, I will not consider possibilistic methods in this thesis.

Coughs	Has a cold	Probability
1	1	0.2
1	0	0.15
0	1	0.05
0	0	0.6

Table 3.4: Statistical distribution of coughing and having a cold

3.4 Conclusion

Both Bayesian networks and Dempster-Shafer theory have been suggested as underlying theory for uncertainty in agent programming. In this section I compare both methods and argue, why Dempster-Shafer theory is the best method to use.

3.4.1 Incorporation in 3APL

3APL uses a belief base with logical formulae. This belief base changes over time when basic actions add or remove beliefs. Using Dempster-Shafer theory, the belief base could be modelled by a mass function, and basic actions can combine this mass function with another, based on their preconditions. Adding or removing a belief can be modelled by combining the mass function that constitutes the belief base with a set of hypotheses that corresponds to that belief, with a mass value of one, respectively zero. Using Bayesian networks, there is a clear correspondence between the belief base and the possibility distribution in the network structure, but there is no intuitive correspondence with the causal structure of the network. Adding or removing beliefs forces the network to be changed, with nodes and relations added or removed. The causal structure of a Bayesian network does not really fit into the 3APL structure.

3.4.2 A priori knowledge needed

In a Bayesian network, the causal network structure and all conditional probabilities have to be specified. In Dempster-Shafer theory, there is no causal structure and only the mass functions that are actually used need to be specified. We can assign a mass value to a set of hypotheses, rather than only single hypotheses, thus modelling ignorance regarding the exact distribution within that set.

3.4.3 Computational aspects

Both Dempster's Rule of Combination as the inference in a Bayesian network have been investigated for their complexity[17, 7]. Both are comparable (for practical use), with the Rule of Combination being in the $\#P$-Complete-class and inference being in the NP-class[2], which means there is no (known) polynomial-time algorithm.

3.4.4 Conclusion

Comparing these two methods, Dempster-Shafer theory fits better with the structure of 3APL. There is a strong relation between the model of the belief base (as a conjunction of logical formulae) and a set of hypotheses in Dempster-Shafer theory. This relation is much stronger than the relation between this belief base and a causal network. On the other hand, Dempster-Shafer theory has one major drawback, and this is the computational complexity of the combination rule. In the remainder of this thesis, I discuss how Dempster-Shafer theory can be incorporated in 3APL, and I investigate how this computational complexity can be overcome.

[2]See the appendix on complexity theory for a discussion of these classes

Chapter 4

Incorporating Dempster-Shafer theory in 3APL

In this chapter, I investigate the question whether the theory of Dempster and Shafer can be applied to 3APL beliefs. To illustrate this investigation, I use the metaphor of a *n-by-m* grid-world where bombs can appear in certain positions in the grid and an agent can partially perceive the environment and move around. The agent tries to sense the bombs surrounding him, thus locating all bombs and safe squares in his environment. Assume that, at a given moment during the execution of the program, the agent has the following belief base in a 2-by-2 grid-world:

BB_1: safe(1)

This indicates, that the agent believes that square 1 is a safe location. How can we relate this belief in terms of the Dempster-Shafer theory? The frame of discernment Ω can be understood as denoting the set of all models of the grid-world, as shown in table 4.1. In a 2-by-2 grid-world there are 16 models, ranging from 'all squares are safe' to 'all squares contain bombs'. We can relate the agent's current beliefs to a subset of hypotheses from Ω, where *each hypothesis is considered as a model of that belief.*

Hyp.	**1**	**2**	**3**	**4**		**Hyp.**	**1**	**2**	**3**	**4**
1	Safe	Safe	Safe	Safe		9	Bomb	Safe	Safe	Safe
2	Safe	Safe	Safe	Bomb		10	Bomb	Safe	Safe	Bomb
3	Safe	Safe	Bomb	Safe		11	Bomb	Safe	Bomb	Safe
4	Safe	Safe	Bomb	Bomb		12	Bomb	Safe	Bomb	Bomb
5	Safe	Bomb	Safe	Safe		13	Bomb	Bomb	Safe	Safe
6	Safe	Bomb	Safe	Bomb		14	Bomb	Bomb	Safe	Bomb
7	Safe	Bomb	Bomb	Safe		15	Bomb	Bomb	Bomb	Safe
8	Safe	Bomb	Bomb	Bomb		16	Bomb	Bomb	Bomb	Bomb

Table 4.1: Bomb location and associated hypothesis

For example, if we define the hypotheses as in table 4.1 then the belief formula safe(1) is a representation of the set $\{H_1, H_2, H_3, H_4, H_5, H_6, H_7, H_8\}$ of hypotheses, which is exactly the set of all models of the belief base BB_1. If we define a mass-function $m_{safe(1)}$ according to this belief base, we would assign 1 to this set, and 0 to Ω (and to all other subsets of Ω). In fact, each belief base can be represented by a mass function. Such a mass function would assign 1 to the subset of Ω which contains all hypotheses that are true with respect to the belief base, or in other words: the maximal subset of hypotheses in Ω that are models of safe(1). Notice that the set $\{H_1, H_2, H_3, H_4, H_5, H_6\}$ consists of models of safe(1) as well, but it is not the maximal subset with this property.

In general, if a belief base is a certain belief formula φ, then it could be represented by a simple support function $m_\varphi(X)$ that supports only the maximal set of hypotheses in Ω that are models of φ. This can be formalized as follows:

$$m_\varphi(X) = \begin{cases} 1 & \text{if } X \subseteq \Omega \ \ \& \ \text{models}(X,\varphi) \ \ \& \ \ \forall Y \subseteq \Omega \ (\text{models}(Y,\varphi) \Rightarrow Y \subseteq X) \\ 0 & \text{otherwise} \end{cases}$$

In this definition, the relation *models(X, φ)* is defined as $\forall M \in X \ \ M \models \varphi$, where M is a model and $\models$ is the propositional satisfaction relation. The condition of the if-clause indicates that X is a subset of Ω, all hypotheses in X are models of φ, and X is maximal, i.e. for every Y with the same properties, it holds that $Y \subseteq X$. In other words, X is the maximal set of hypotheses that are models of φ. In the sequel, for the sake of readability I will use $X \models^\Omega \varphi$ as a shorthand for this complex condition. Using this shorthand notation, the mass function that represents the belief base φ can be rewritten as:

$$m_\varphi(X) = \begin{cases} 1 & \text{if } X \models^\Omega \varphi \\ 0 & \text{otherwise} \end{cases}$$

The belief base BB_1 can then be represented by :

$$m_{safe(1)}(X) = \begin{cases} 1 & \text{if } X \models^\Omega \text{safe}(1) \\ 0 & \text{otherwise} \end{cases}$$

4.1 Adding beliefs

If we add another belief formula to the belief base, the resulting belief base can be represented by the combination of the mass function of both belief formulae. Suppose we add *safe(2)* to the belief base, with the following mass function:

$$m_{safe(2)}(X) = \begin{cases} 1 & \text{if } X \models^\Omega \text{safe}(2) \\ 0 & \text{otherwise} \end{cases}$$

We can combine both pieces of evidence using Dempster's rule of combination. Since the only non-empty intersection of sets defined by either $m_{safe(1)}$ or $m_{safe(2)}$ is the subset $\{X \mid X \models^\Omega \text{safe}(1) \wedge \text{safe}(2)\}$, the resulting mass function $m_1 = m_{safe(1)} \oplus m_{safe(2)}$ is defined as follows[1]:

$$m_1(X) = \begin{cases} 1 & \text{if } X \models^\Omega \text{safe}(1) \wedge \text{safe}(2) \\ 0 & \text{otherwise} \end{cases}$$

Note that X corresponds to the subset $\{H_1, H_2, H_3, H_4\}$ of hypotheses.

Apart from these beliefs, which can be either true or false, we could imagine a situation where a belief is uncertain. We might conclude, on the basis of certain evidence, that a location *probably* contains a bomb, so that the belief formula *bomb(3)*, with a probability value of 0.7, is added to the belief base. In order to incorporate such cases, I introduce the concept of a *basic belief formula* to represent uncertain belief formulae, and define it as follows:

Definition 17 *(Basic belief formula). Let φ be a belief formula and $p \in [0..1]$. Then the pair $\varphi : p$, which indicates that φ holds with probability p, will be called a basic belief formula*[2]*.*

With these basic belief formulae, the above mentioned belief base can be represented as { safe(1): 1, safe(2): 1, bomb(3): 0.7 }. Of course, we could represent *bomb(3): 0.7* as a mass function, as we did with beliefs safe(1) and safe(2). This mass function would assign a probability value of 0.7 to the set of hypotheses, all having a bomb on location 3, and (because we have no further information) a probability value of 0.3 tot Ω:

$$m_{bomb(3)}(X) = \begin{cases} 0.7 & \text{if } X \models^\Omega \text{bomb}(3) \\ 0.3 & \text{if } X = \Omega \\ 0 & \text{otherwise} \end{cases}$$

[1] I will use simple indices for the combined mass functions to improve readability

[2] The term *basic belief formula* should not be confused with an *atomic belief formula*. Note, that the probability assigned to a basic belief formula cannot be (further) distributed to the atomic formulae that constitute the basic belief formula

If we would combine m_1 and $m_{bomb(3)}$ using Dempster's rule of combination, we would get the following mass function:

$$m_2 = m_1 \oplus m_{bomb(3)}(X) = \begin{cases} 0.7 & \text{if } X \models^{\Omega} \text{safe}(1) \wedge \text{safe}(2) \wedge \text{bomb}(3) \\ 0.3 & \text{if } X \models^{\Omega} \text{safe}(1) \wedge \text{safe}(2) \\ 0 & \text{otherwise} \end{cases}$$

The mass function m_2 represents our updated belief base. Note that $X \models^{\Omega} \text{safe}(1) \wedge \text{safe}(2)$ is exactly the set $\{H_1, H_2, H_3, H_4\}$, and $X \models^{\Omega} \text{safe}(1) \wedge \text{safe}(2) \wedge \text{bomb}(3)$ is the set $\{H_3, H_4\}$.

4.2 Deleting beliefs

We can also delete belief formulae from our belief base. For example, we could conclude that square 3 does not contain a bomb after all during the execution of our program. Deletion of a formula is modelled as the addition of its negation. Since $\neg\text{bomb}(3) = \text{safe}(3)$ in our example, this corresponds to the maximal set of hypotheses according to which there is *no* bomb on location 3:

$$m_{safe(3)}(X) = \begin{cases} 1 & \text{if } X \models^{\Omega} \text{save}(3) \\ 0 & \text{otherwise} \end{cases}$$

Combining m_2 and $m_{safe(3)}$ leads to the following mass function:

$$m_3 = m_2 \oplus m_{safe(3)}(X) = \begin{cases} 1 & \text{if } X \models^{\Omega} \text{safe}(1) \wedge \text{safe}(3) \\ 0 & \text{otherwise} \end{cases}$$

Of course, we could also conclude that a certain belief becomes less probable instead of impossible. In that case, the negation of the formula under consideration will be added with a certainty value, for example *safe(3): 0.3*. We would represent this formula as:

$$m_{safe(3)'}(X) = \begin{cases} 0.3 & \text{if } \models^{\Omega} \text{safe}(3) \\ 0.7 & \text{if } X = \Omega \\ 0 & \text{otherwise} \end{cases}$$

Combining this alternative mass function $m_{safe(3)'}$ and m_2 leads to the following mass function:

$$m_4 = m_2 \oplus m_{safe(3)'}(X) = \begin{cases} 0.59 & \text{if } X \models^{\Omega} \text{safe}(1) \wedge \text{save}(3) \\ 0.41 & \text{if } X \models^{\Omega} \text{safe}(1) \wedge \text{bomb}(3) \\ 0 & \text{otherwise} \end{cases}$$

4.3 Composite beliefs

Until now only used atomic formula were used in the examples. However, we can also model disjunctions, conjunctions and negation of beliefs as sets of hypotheses by mapping disjunction, conjunction and negation of beliefs, to respectively unions, intersections, and complements of sets of hypotheses. This is illustrated in table 4.2. For an illustration of composite beliefs, consider the following two formulae in the already mentioned grid-world:

1. $\text{safe}(2) \wedge (\text{safe}(3) \vee \text{safe}(4))$
2. $\text{safe}(1) \vee (\neg\text{safe}(2) \wedge \text{safe}(3))$

These formulae correspond to the following sets in our example, respectively:

1. the set $\{H_1, H_2, H_3, H_9, H_{10}, H_{11}\}$
2. the set $\{H_1, H_2, H_3, H_4, H_5, H_6, H_7, H_8, H_{13}, H_{14}\}$

If formula 1 has a probability of p, and formula 2 has a probability of q, then these formula could be represented by basic belief formulae as follows:

belief formula	Dempster-Shafer
safe(X)	set of possible worlds (hypotheses) in which safe(X) is true
safe(X) $\vee$ safe(Y)	union of the set of possible worlds (hypotheses) in which safe(X) is true and the set in which safe(Y) is true
safe(X) $\wedge$ safe(Y)	intersection of the set of possible worlds (hypotheses) in which safe(X) is true and the set in which safe(Y) is true
$\neg$safe(X)	complement of the set of possible worlds (hypotheses) in which safe(X) is true

Table 4.2: Logical connectives and Dempster-Shafer sets of hypotheses

$$m_1(X) = \begin{cases} p & \text{if } X \models^{\Omega} \text{safe}(2) \wedge (\text{safe}(3) \vee \text{safe}(4)) \\ 1-p & \text{if } X = \Omega \\ 0 & \text{otherwise} \end{cases} \quad \text{and}$$

$$m_2(X) = \begin{cases} q & \text{if } X \models^{\Omega} \text{safe}(1) \vee (\neg\text{safe}(2) \wedge \text{safe}(3)) \\ 1-q & \text{if } X = \Omega \\ 0 & \text{otherwise} \end{cases}$$

Obviously, the conjunction of these two statements is:

safe(1) $\wedge$ safe(2) $\wedge$ (safe(3) $\vee$ safe(4))

And from the table follows, that this result corresponds to the set $\{H_1, H_2, H_3\}$, which is the intersection of $\{H_1, H_2, H_3, H_9, H_{10}, H_{11}\}$ and $\{H_1, H_2, H_3, H_4, H_5, H_6, H_7, H_8, H_{13}, H_{14}\}$. Therefore, using Dempster's Rule of Combination leads to the following mass function:

$$m_1 \oplus m_2(X) = \begin{cases} p \cdot q & \text{if } X \models^{\Omega} \text{safe}(1) \wedge \text{safe}(2) \wedge (\text{safe}(3) \vee \text{safe}(4)) \\ p \cdot (1-q) & \text{if } X \models^{\Omega} \text{safe}(2) \wedge (\text{safe}(3) \vee \text{safe}(4)) \\ (1-p) \cdot q & \text{if } X \models^{\Omega} \text{safe}(1) \vee (\neg\text{safe}(2) \wedge \text{safe}(3)) \\ (1-p) \cdot (1-q) & \text{if } X = \Omega \\ 0 & \text{otherwise} \end{cases}$$

4.4 Inconsistency problem

The issue of inconsistency in Dempster's rule of combination deserves further attention. In the original rule, as defined in definition 14, combinations that lead to an empty set have a mass probability of zero, and the other combinations are scaled to make sure all mass probabilities add to one. This leads to unexpected results when two mass functions with a high degree of conflict are combined. This can be demonstrated with an often-used example (e.g. in [13]):

In a murder case there are three suspects: Peter, Paul and Mary. There are two witnesses, who both give highly inconsistent testimonies, which can be represented with the following mass functions:

$$m_1(X) = \begin{cases} 0.99 & \text{if } X = \text{'killer is Peter'} \\ 0.01 & \text{if } X = \text{'killer is Paul'} \\ 0 & \text{otherwise} \end{cases}$$

$$m_2(X) = \begin{cases} 0.99 & \text{if } X = \text{'killer is Mary'} \\ 0.01 & \text{if } X = \text{'killer is Paul'} \\ 0 & \text{otherwise} \end{cases}$$

Combining these two mass functions leads to a certain belief that Paul is the killer, although there is hardly any support in either of the witnesses' testimonies. Although m_1('killer is Paul') $\cdot$ m_2('killer is Paul') = 0.0001, this value is normalized and divided by 0.0001, since the mass value for the combination of X = 'killer is Paul' is the only mass value that is assigned to a non-empty set:

$$m_1 \oplus m_2(X) \begin{cases} 1 & \text{if } X = \text{'killer is Paul'} \\ 0 & \text{otherwise} \end{cases}$$

Sentz [22] describes a number of alternatives for this rule of combination. The most prominent (according to Sentz) is Yager's *modified Dempster's rule*[29]. Ultimately, this rule attributes the probability of combinations, which lead to the empty set, to Ω [3]. A similar approach is demonstrated by Smets [24], which states that in the case of inconsistent mass functions, the closed world assumption (the assumption that one of the three suspects is the murderer) is not valid. The probability of empty sets should be attributed to $\emptyset$, as a sort of 'unknown third'. In both cases, the mass value of $X =$ 'killer is Paul' is not normalized. This would lead to a mass function of:

$$m_1 \oplus m_2(X) \begin{cases} 0.0001 & \text{if } X = \text{'killer is Paul'} \\ 0.9999 & \text{if } X = \Omega \\ 0 & \text{otherwise} \end{cases} \quad \text{(Yager), respectively}$$

$$m_1 \oplus m_2(X) \begin{cases} 0.0001 & \text{if } X = \text{'killer is Paul'} \\ 0.9999 & \text{if } X = \emptyset \\ 0 & \text{otherwise} \end{cases} \quad \text{(Smets)}$$

Jøsang [13] poses an alternative, namely the *consensus operator*, which attributes the means of the probabilities of two inconsistent beliefs to the combination, rather than their multiplication:

$$m_1 \oplus m_2(X) \begin{cases} 0.495 & X = \text{'killer is Peter'} \\ 0.495 & X = \text{'killer is Mary'} \\ 0.01 & X = \text{'killer is Paul'} \\ 0 & X = \emptyset \end{cases}$$

These approaches (Dempster, Yager/Smets and Jøsang) can be summarized using the 'murder case' example, as shown in table 4.3:

Suspect	W_1	W_2	Dempster	Yager/Smets	Jøsang
Peter	0.99	0	0	0	0.495
Paul	0.01	0.01	1	0.0001	0.01
Mary	0	0.99	0	0	0.495
$\emptyset/\Omega$	0	0	0	0.9999	0

Table 4.3: Attribution of mass to inconsistent combinations

Note, that the issue of inconsistency directly relates to the choice of the frame of discernment Ω. In this example, the frame of discernment is restricted to be the set of three *mutually exclusive* hypotheses, namely {Paul, Peter, Mary}. If, on the other hand, our frame would be $\Omega =$ {Paul, Peter, Mary, Peter or Mary}, and we would map the phrase 'killer is Peter' to the subset {Peter, Peter or Mary} and the phrase 'killer is Mary' to the subset {Mary, Peter or Mary}, then there would be no inconsistency at all. I deal with the choice of our frame of discernment in the next section.

4.5 The frame of discernment

Until now, we have mapped agent beliefs to a given set of 16 hypotheses in our 2-by-2 grid-world. Unfortunately, the frame of discernment that corresponds to a given agent program is unknown, and, just as important, there is no unique frame of discernment in such a program. We might just as well add a totally irrelevant hypothesis H_{17}, stating 'All squares contain apples'. We do not know if a certain hypothesis, say H_{16}, can become true during the execution of the program. This implies that the relation between frame of discernment and beliefs is a many-to-many mapping.

This problem can, however, be solved. The number of beliefs an agent can hold during execution of the program is finite in 3APL, since only *basic actions* can update the belief base. The update

[3] To be more exact, Yager differentiates between *ground probabilities q(X)* and *basic probabilities m(X)*. The empty set can have a $q(\emptyset) \geq 0$. When combining, these ground probabilities are used and the mass is attributed *after* the combination, where $m(X) = q(X)$ for $X \neq \emptyset$, and $m(\Omega) = q(\Omega) + q(\emptyset)$.

corresponding to a basic action is specified as the post-condition of the basic action which is determined by the programmer before running the program. Therefore, in a given 3APL program all possible beliefs are given either by the initial belief base or by the specification of the basic actions. Therefore, we can construct a *theoretical* frame of discernment that includes a set of hypotheses such that each belief that the agent can hold during its execution can be mapped to a subset of the frame of discernment. Shafer states [23, p.281], that in general the frame of discernment cannot be determined beforehand (i.e. without knowing which evidence might be relevant), and that we tend to enlarge it as more evidence becomes available. But, on the other side, if Ω is too large, holding too much irrelevant hypotheses, the probability of any hypothesis is unreasonably small. By stating that the frame of discernment should be large enough to hold all relevant hypotheses with respect to the program under consideration, Ω will be neither too small nor too large.

In this thesis, I demand that Ω should be such that for each belief that an agent can hold during its execution (i.e. each combination of the basic belief formulae) there must be at least one nonempty subset of hypotheses in Ω. In other words, each conjunction of basic belief formulae has a nonempty subset of hypotheses from Ω that are models of the conjunction of basic belief formulae.

4.6 Mass calculation

As we have seen, any given belief base can be represented with a mass function. Generally, a belief formula $b_i$with probability p_i divides the set of all hypotheses Ω into:

$$m_{b_i}(\mathrm{X}) = \begin{cases} p_i & \text{if } X \models^{\Omega} b_i \\ 1 - p_i & \text{if } X = \Omega \\ 0 & \text{otherwise} \end{cases}$$

The combination of belief formulae $b_1 \dots b_n$ can thus be represented with a mass function $m_{1\dots n} = m_1 \oplus \dots \oplus m_n$, related to beliefs $b_1, \dots, b_n$, where the number of subsets of Ω that are used to define m_k and have a mass value > 0 is equal to 2^n. When a belief formula m_n is combined with an already existing combination $m_1 \oplus \dots \oplus m_{n-1}$, the resulting mass function $m_1 \oplus \dots \oplus m_n$ is defined by the non-empty intersections of all subsets of Ω in m_n, with all subsets of Ω in $m_1 \oplus \dots \oplus m_{n-1}$. Since simple support functions are used to represent our beliefs, the number of resulting subsets is doubled with each added belief formula.

Because n belief formulae lead to a mass function of 2^n combinations, keeping a mass function in memory and updating it when the belief base changes will lead to a combinatorial explosion in both processing time and memory requirements. As Orponen [17] showed, Dempsters's Rule of Combination is #P-Complete[4]. Wilson [27] has provided a number of algorithms to overcome this problem using for example Monte Carlo methods, and Barnett [2, 3] has shown, that the calculation of the combination is linear if only singleton subsets are used or if the subsets are atomic with respect to the evidence. The latter restriction is problematic since we are not aware of the actual hypotheses that constitute the frame of discernment.

For example, recall table 4.1 with hypotheses of bomb location in a 4 x 4 grid-world, and suppose we have two simple support functions that have $\{ H_1, H_2, H_3, H_4 \}$ and Ω, respectively $\{ H_1, H_2 \}$ and Ω as their core, so the evidence available is $\{ \{ H_1, H_2 \}, \{ H_1, H_2, H_3, H_4 \}, \Omega \}$. The predicate 'atomic with respect to the evidence' for a subset $\theta \neq \emptyset$, denoted as $\mathbf{At}_{\varepsilon}(\theta)$, is defined in [3] to be true if and only if θ is either a subset or a disjoint[5] of every focal element in the available evidence. In this example, Ω is known to be $\{ H_1, \dots, H_{16} \}$. The atomic subsets of Ω with respect to the evidence are $\{ H_1, H_2 \}$, $\{ H_3, H_4 \}$, the atoms $H_1 \dots H_4$ and the powerset of $\Omega \setminus \{ H_1 \dots H_4 \}$. If Ω is not known beforehand, as we have suggested in the previous section, we cannot determine which subsets are atomic. But, as we will see, there is no need to update - or, for that matter, even calculate - the entire mass function, because we only need to combine simple support functions and our frame of discernment is such, that no inconsistency can occur.

Of course, an empty belief base has a trivial mass function of:

$$\mathrm{m}(X) = \begin{cases} 1 & \text{if } X = \Omega \\ 0 & \text{otherwise} \end{cases}$$

[4] See the appendix on complexity theory

[5] A is a disjoint of B if A and B have no elements that overlap

If we add one basic belief formula $b_1 : p_1$, we compute the following mass function:

$$m_{b_1}(X) = \begin{cases} p_1 & \text{if } X \models^{\Omega} b_1 \\ 1 - p_1 & \text{if } X = \Omega \\ 0 & \text{otherwise} \end{cases}$$

Adding $b_2 : p_2$ leads to the following mass function:

$$m_{b_1} \oplus m_{b_2}(X) = m(X) = \begin{cases} p_1 \cdot (1 - p_2) & \text{if } X \models^{\Omega} b_1 \\ p_2 \cdot (1 - p_1) & \text{if } X \models^{\Omega} b_2 \\ p_1 \cdot p_2 & \text{if } X \models^{\Omega} b_1 \wedge b_2 \\ (1 - p_1) \cdot (1 - p_2) & \text{if } X = \Omega \\ 0 & \text{otherwise} \end{cases}$$

Assuming that $b_1 \wedge b_2$ is not equal to b_1 or b_2, which might be the case if e.g. $b_1 = a$ and $b_2 = a \vee b$. In the following, I will treat the belief formulae to be such, that their intersections are separate clauses. This has no consequences, as I will show in section 4.8.

Note that these consecutive mass combinations can be generalized to a situation with n basic belief formulae, and that the combined mass function will grow exponentially. Fortunately, there is no need to calculate the entire mass function. If we need the mass value for a certain subset X of Ω, we can calculate it using the probabilities in the belief base without the need to calculate the entire mass function. Before formulating and proving this theorem, I will first show that we can simplify Dempster's Rule of Combination if we use simple support functions to represent basic belief formulae, and if we define Ω to be such that each conjunction of basic belief formulae that the agent can hold during its execution maps to a nonempty subset of Ω, as we discussed in section 4.5. The two demands are intuitive: the first demand states, that the evidence that is represented by each basic belief formula only supports one subset of Ω, and the second guarantees that the conjunction of basic belief formulae has a model.

To facilitate further considerations, I define p_φ as the probability assigned to a certain belief formula φ, and define m_φ to be a simple support function associated with φ. The core of m_φ is denoted as $C(m_\varphi)$. I introduce the concept *M-completeness* to denote the above mentioned condition on Ω, and define it as follows:

Definition 18 *(M-complete). Let M be a set of mass functions and Ω be a set of hypotheses, then Ω will be called M-complete if and only if* $\forall m_\phi, m_\psi \in M_\Omega(\{X \mid X \models^{\Omega} \phi\} \in C(m_\phi) \;\; \& \;\; \{X \mid X \models^{\Omega} \psi\} \in C(m_\psi) \Rightarrow \{X \mid X \models^{\Omega} \phi \wedge \psi\} \in \Omega)$

Theorem 1 *Let S_Ω be the set of all basic belief formulae, and M be the set of all mass functions associated with basic belief formulae from S_Ω. Let Ω be M-complete, and let ϕ and ψ be two non-equivalent basic belief formulae (i.e. $\neg(\phi \equiv \psi)$). Then $\sum_{Y \cap Z \neq \emptyset} m_\phi(Y) \cdot m_\psi(Z) = 1$, and for each $X \subseteq \Omega$ there are at most one $Y \subseteq \Omega$ and one $Z \subseteq \Omega$ relevant for the Dempster's combination rule such that this rule can be simplified to $m_\phi \oplus m_\psi(X) = m_\phi(Y) \cdot m_\psi(Z)$ where $Y \cap Z = X$.*

Proof 1 *Since ϕ and ψ are two basic belief formulae, the only $Y, Z \subseteq \Omega$ for which $m_\phi(Y) \neq 0$ and $m_\psi(Z) \neq 0$ are elements of the core of m_ϕ and m_ψ, i.e. $C(m_\phi) = \{M_Y, \Omega\}$ where $M_Y \models^{\Omega} \phi$ and $C(m_\psi) = \{M_Z, \Omega\}$ where $M_Z \models^{\Omega} \psi$. In other words, Y ranges over $C(m_\phi)$ and Z ranges over $C(m_\psi)$. For all other subsets Y and Z from Ω, we have $m_\phi(Y) = 0$ and $m_\psi(Z) = 0$ such that $m_\phi(Y) \cdot m_\psi(Z) = 0$ which does not influence the summation $\sum_{Y \cap Z = X} m_\phi(Y) \cdot m_\psi(Z)$ in the numerator of the Dempster's Rule of Combination.*

Given that Y and Z range over $C(m_\phi)$ and $C(m_\psi)$ respectively, it is clear that for each $X \subseteq \Omega$ we can have at most one $Y \in C(m_\phi)$ for which $m_\phi(Y) \neq 0$ and at most one $Z \in C(m_\psi)$ for which $m_\psi(Z) \neq 0$ such that $Y \cap Z = X$. More specifically, for any $X \subseteq \Omega$, the subsets Y and Z can be determined as follows:

If $X = \Omega$, then $Y = \Omega$ and $Z = \Omega$.
If $X \models^{\Omega} \phi$, then $Y = X$ and $Z = \Omega$.
If $X \models^{\Omega} \psi$, then $Y = \Omega$ and $Z = X$.
If $X \models^{\Omega} \psi \wedge \psi$, then $Y \models^{\Omega} \phi$ and $Z \models^{\Omega} \psi$.

For all other $X \subseteq \Omega$, there are no Y and Z. From our definition of Ω follows, that for these Y and Z we have $Y \cap Z \neq \emptyset$, and since we use simple support structures $\sum_{Y \cap Z \neq \emptyset} m_\phi(Y) \cdot m_\psi(Z) = 1$. This proves that the denominator does not influence the result of the combination rule. Therefore, Dempster's Rule of Combination can be simplified to $m_\phi \oplus m_\psi(X) = m_\phi(Y) \cdot m_\psi(Z)$ where $Y \cap Z = X$. □

From the associativity of Dempster's Rule of Combination[22] follows, that $Y_1, \ldots, Y_n$ in the formula $Y_1 \cap \ldots \cap Y_n = X$ can be determined in a similar way. For example, in a belief base consisting of three basic belief formulae ϕ, ψ, and χ, then where $X \models^\Omega \psi \wedge \psi$, $Y_\phi \models^\Omega \phi$, $Y_\psi \models^\Omega \psi$ and $Y_\chi = \Omega$. This result can be used to reduce the complexity of Dempster's Rule of Combination.

In the second theorem I formulate a straightforward method to calculate the mass of any combination of basic belief formula, and prove that this calculation leads to the same result as the simplified Rule of Combination.

Theorem 2 *Let S_Ω be the set of all basic belief formulae, and M be the set of all mass functions associated with basic belief formulae from S_Ω. Let Ω be M-complete. For each subset $X \subseteq \Omega$, there exists an unique bi-partition of the set of basic belief formulae S_Ω, say S_X^+ and S_X^-, such that the general case of Dempster's Rule of Combination can be simplified as follows:*

$$\bigoplus_{i=1\ldots n} m_i(X) = \prod_{\varphi \in S_X^+} p_\varphi \cdot \prod_{\varphi \in S_X^-} (1 - p_\varphi)$$

Proof 2 *Let Y and Z be subsets of Ω. Based on theorem 1, Dempster's Rule of Combination can be reduced to $m_1 \oplus m_2(X) = m_1(Y) \cdot m_2(Z)$, where $Y \cap Z = X$. The mass function that is formed by n consecutive combinations, is then equal to*

$$\forall X, Y_1, \ldots, Y_n \subseteq \Omega \; : \; \bigoplus_{i=1\ldots n} m_i(X) = \prod_{i=1\ldots n} m_i(Y_i), \quad \text{where} \bigcap_{i=1\ldots n} Y_i = X \tag{4.1}$$

Given the mass functions $m_{\phi_1}, \ldots, m_{\phi_n} \in M_\Omega$ and for any $X \subseteq \Omega$ and $1 \leq i \leq n$, there exists at most one $Y_i \subseteq \Omega$ ranging over the core $C(m_{\phi_i})$ such that $Y_1 \cap \ldots \cap Y_n = X$ and $m_{\phi_i}(Y_i) \neq 0$ (for the same reason as in theorem 1). According to the definition of the simple support function, $m_{\phi_i}(Y_i)$ can be either p_{ϕ_i} or $1 - p_{\phi_i}$ for $1 \leq i \leq n$. Let then $S_X^+ = \{\phi \mid Y_i \models^\Omega \phi \;\&\; \phi \in S_\Omega\}$ which is the set of all basic belief formula for which the corresponding mass function assigns p_{ϕ_i} to the subset Y_i (rather than $1 - p_{\phi_i}$). Using $S_X^- = S_\Omega \backslash S_X^+$ proves the theorem. □

4.7 Updating the belief base

In order to incorporate new information (e.g. by observation or communication) in their beliefs, agents need to update their belief base. Since both an existing belief base (consisting of basic belief formulae) and a new basic belief formulae can both be represented by mass functions, we can add this new basic belief formula to the belief base which in its turn can be represented by a mass function. This would yield the same result as combining each single support function associated with the basic belief formulae, as I proved in theorem 2. However, if the belief base already contains this belief formula, we can update it using the associative nature of Dempster's Rule of Combination. For example, suppose a belief base, which consists of two basic belief formulae $b_1 : p_1$ and $b_2 : p_2$, is updated with the basic belief formula $b_1 : p_3$. The new probability of b_1 can be calculated since $m_{b_1} \oplus m_{b_2} \oplus m'_{b_1} = m_{b_1} \oplus m'_{b_1} \oplus m_{b_2} = (m_{b_1} \oplus m'_{b_1}) \oplus m_{b_2}$. From Dempster's rule of combination follows, that the new probability of b_1 is $p_1 + p_3 - p_1 \cdot p_3$. If the belief base has to be updated with a basic belief formula which is evidence *against* a certain proposition (i.e. we have a negated belief formula as postcondition) than the new probability of b_1 is $p_1 - p_1 \cdot p_3$.

These results can be motivated as follows. Note that the resulting mass function has to be a simple support function to fulfill the requirements I discussed in theorem 2. Positive evidence is combined with an already existing mass function, and because two equal simple support functions are combined, the result is also a simple support function. For negative evidence, the new simple support function is constructed by multiplying the mass in the original simple support function that was assigned to the evidence, with the mass that is not assigned to its negation in the postcondition,

and assigning the remaining mass to Ω. While I think this is intuitive and useful in practice, further research has to be done on a formal justification of dealing with negative evidence under a Closed World Assumption.

4.8 Querying the belief base

We can test (query) if a proposition φ can be derived from a belief base Γ using the belief and plausibility functions I discussed in chapter 3. These belief and plausibility functions (defined in terms of a certain mass function) return the total mass assigned to models of φ and the total mass that is *not* assigned to models of the *negation* of φ. Using this functions, we can test if φ can be derived from Γ within a certain probability interval $[L, U]$ (denoted as $\Gamma \models_{[L,U]} \varphi$). This can be done as follows. As discussed in section 3, the belief base Γ can be represented by a mass function m_Γ that assigns a mass value to each subset of the frame of discernment. As the belief formula φ can be represented as a subset of the frame of discernment, we can consider the above test as computing the lower and upper limits on the probability that the mass function m_Γ assigns to the subset of Ω that represents the belief formula φ, i.e. we can compute the probability that m_Γ assigns to $X \subseteq \Omega$ where $X \models^\Omega \varphi$. Since $Bel(X) = \sum_{Y \subseteq X} m(Y)$ and we have defined the shorthand $X \models^\Omega \varphi$ as ($X \subseteq \Omega$ & models (X, φ) & $\forall Y \subseteq \Omega$ (models$(Y, \varphi) \Rightarrow Y \subseteq X)$), we can define $Bel(X \mid X \models^\Omega \varphi)$ as the sum of all $m(Y)$, where ($Y \subseteq \Omega$ & models(Y, φ)).

Recall from section 4.6 that I treated the formulae in the belief base without actually evaluating the formulae. It might be the case that e.g. $b_1 \wedge b_2$ evaluates to b_1. But since we have defined the Bel function as the sum of the mass function of all subsets Y of Ω that satisfy models(Y, Ω), and the models of $b1 \wedge b_2$ are the same as the models of b_1, this leads to the same results as if we had calculated the mass value of b_1 as the sum of b_1 and $b_1 \wedge b_2$.

Likewise, $Pl(X \mid X \models^\Omega \varphi)$ can be defined as the sum of all $m(Y)$, where ($Y \subseteq \Omega$ & $\neg$models$(Y, \neg\varphi)$). As the deduction of beliefs in 3APL is based on the Closed World Assumption, $\Gamma \models_{CWA} \varphi \Leftrightarrow \Gamma \not\models_{CWA} \neg\varphi$, and therefore $Bel(X \mid X \models^\Omega \varphi) = Pl(X \mid X \models^\Omega \varphi)$. Consequently, the calculated probability of a certain proposition is a single value, rather than an interval.

As an example, let's consider the belief base $\{ b_1 : 0.7, b_2 : 0.3, b_3 : 0.6\}$, and test to what extent $b_1 \vee (b_2 \wedge b_3)$ can be deducted:

To calculate $Bel(X \mid X \models^\Omega b_1 \vee (b_2 \wedge b_3))$ we must add all $m(X)$ that satisfy ($X \subseteq \Omega$ & models$(X, b_1 \vee (b_2 \wedge b_3))$), and the only non-zero $m(X)$ that satisfy this condition are $\{ X \mid X \models^\Omega b_1\}$, $\{ X \mid X \models^\Omega b_1 \wedge b_2\}$, $\{ X \mid X \models^\Omega b_1 \wedge b_3\}$, $\{ X \mid X \models^\Omega b_1 \wedge b_2 \wedge b_3\}$, and $\{ X \mid X \models^\Omega b_2 \wedge b_3\}$. We calculate these mass elements using the formula presented earlier. Note that we only need to calculate five mass elements, instead of the entire mass function, which has eight elements:

$m(X \mid X \models^\Omega b_1) = 0.196$
$m(X \mid X \models^\Omega b_1 \wedge b_2) = 0.084$
$m(X \mid X \models^\Omega b_1 \wedge b_3) = 0.294$
$m(X \mid X \models^\Omega b_1 \wedge b_2 \wedge b_3) = 0.126$
$m(X \mid X \models^\Omega b_2 \wedge b_3) = 0.054$

$Bel(X \mid X \models^\Omega b_1 \vee (b_2 \wedge b_3))$ equals $0.196 + 0.084 + 0.294 + 0.126 + 0.054 = 0.754$. This means the probability of the proposition $b_1 \vee (b_2 \wedge b_3)$ can be derived from our belief base with a certainty value 0.754.

Chapter 5

Syntax and semantics of 3APL enhanced with uncertainty

In this section I propose some changes on the syntax and the semantics of 3APL, as defined in chapter 2 of this thesis. I will only list the changed definitions in this chapter.

A plan in 3APL is either a basic action, a test, an abstract plan or a composite plan, as described in chapter 2. We dealt with basic actions in the previous section. Both abstract plans and composite plans will remain the same. The test $\varphi?$ must be changed to reflect the uncertainty of the proposition tested. I propose to enhance the test with an upper and lower boundary. This defines an interval that must incorporate the mass of the proposition, for the test to succeed.

The test $\varphi?_v^w$ succeeds if the mass of φ is greater than or equal to v, and the mass of φ is smaller than or equal to w. The original test $\varphi?$ can be described as $\varphi?_1^1$, which indicates that φ is certain, and $\neg\varphi$ can be described as $\varphi?_0^0$, which indicates that φ is impossible. An exact match can be tested with $v = w$.

The actual implementation of the test goal is a query from the belief base and a test if the obtained certainty value lies between v and w. A rule in a 3APL rule base has a *guard* to test if the rule is applicable. This guard can be implemented as a query from the belief base. A 3APL basic action has a pre-condition and a post-condition, which can be seen as respectively a query from the belief base and an update of the belief base if the query succeeds. Both are described earlier.

5.1 Syntax

Our belief base language needs to be enhanced with an uncertainty measure (a real number in the range [0..1]). The belief queries are also altered, since a query is performed by determining if a certain formula can be derived from the belief base *between a lower and upper limit.* The plan language and the rules do not change.

Definition 19 *(extended belief base language)*
The formulae of the belief base language is defined as follows: Let $\psi \in L_{PRED}$ be a ground formula, and let $\phi, \phi_1, \ldots, \phi_n \in L_{PRED}$. Let $p \in [0 \ldots 1]$. The belief language L_B is a set of formulae defined as follows:

Belief language L_B:

- $\psi : p, \forall_{x_1,\ldots,x_n}(\phi_1 \wedge \ldots \wedge \phi_n \rightarrow \phi : p) \in L_B$

Where $\forall_{x_1,\ldots,x_n}(\varphi)$ denotes the universal closure of the formula φ for every variable $x_1, \ldots, x_n$ occurring in φ.

Definition 20 *(extended belief queries)*
Let L_{PRED} be the base language, and let $v, w \in [0 \ldots 1]$. Then, the belief query language L_{BQ} with typical formula β is defined as follows:

- *if $\phi_1, \ldots, \phi_n \in L_{PRED}$, then $\mathbf{B}(\phi_1 \wedge \ldots \wedge \phi_n)_v^w, \neg\mathbf{B}\phi_1 \wedge \ldots \wedge \phi_n)_v^w \in L_{BQ}$,*
- *$\top_1^1 \in L_{BQ}$.*

A 3APL configuration must be changed to include the frame of discernment, since this frame is used in the mass functions that constitute the belief base:

Definition 21 *(extended configuration)*
A configuration of a 3APL agent is a tuple $< \Omega, \sigma, \Pi, \theta >$*, in which* Ω *is the frame of discernment,* $\sigma \subseteq L_B$ *is the belief base of the agent,* $\Pi \subseteq L_P$ *is the plan base of the agent, and* θ *represents the substitution that binds domain variables to domain terms. The belief bases are assumed to be grounded.*

5.2 Semantics

In traditional 3APL, a basic action precondition, a reasoning rule's guard, and a test goal all are defined as $\sigma \models \phi$, where $\sigma \in L$ is the agent's belief base and ϕ is a belief formula. The formula $\sigma \models \phi$ returns a substitution if ϕ can be derived from the belief base. A logical formula φ can be deducted from a belief base Γ if the set of worlds that are true in Γ are a subset of the set of worlds that are true in φ. For example, take $\Gamma = b_1 \wedge b_2$[1] and $\varphi = b_1$. From the truth table in table 5.1 follows, that $b_1 \wedge b_2$ is true in world w_1 only, and b_1 is true in worlds w_1 and w_2, so the models of $\Gamma \subseteq$ the models of φ.

World	b_1	b_2	$b_1 \wedge b_2$
w_1	1	1	1
w_2	1	0	0
w_3	0	1	0
w_4	0	0	0

Table 5.1: truth values in Γ

In 3APL with uncertainty, the belief base is represented by a mass function, which is the combination of all simple support functions which represent the beliefs in the belief base. Instead of a truth assignment, there is a certainty distribution over the subsets of Ω. If we take a belief base Γ' with two basic belief formulae $b_1 : p_1$ and $b_2 : p_2$, then the combination of the mass functions associated with b_1 and b_2 has the following distribution:

Subset	evaluates to	mass value
$X \mid X \models^{\Omega} b_1 \wedge b_2$	$\{w_1\}$	$p_1 \cdot p_2$
$X \mid X \models^{\Omega} b_1$	$\{\ w_1, w_2\ \}$	$p_1 \cdot (1 - p_2)$
$X \mid X \models^{\Omega} b_2$	$\{\ w_1, w_3\ \}$	$p_2 \cdot (1 - p_1)$
Ω	$\{\ w_1, w_2, w_3, w_4\ \}$	$(1 - p_1) \cdot (1 - p_2)$
otherwise		0

Table 5.2: certainty distribution of Γ'

The Bel(X) function is defined in section 4.2 as the sum of the mass values of all sets of worlds, which are a subset of X. So, Bel$(X \mid X \models^{\Omega} b_1)$ is the sum of the mass values of all sets of worlds, which are a model of b_1. Since the model of b_1 is $\{\ w_1, w_2\ \}$, Bel$(X \mid X \models^{\Omega} b_1) = p_1 \cdot p_2 + p_1 \cdot (1 - p_2)$.

We can now further define the semantics of the test goal $\mathbf{B}\varphi_v^w$ (with $v, w \in [0..1]$): it returns a substitution *iff.* $w \geq Bel_m(X \mid X \models^{\Omega} \varphi) \geq v$, where m is the mass function that is the result of the combination of all mass functions associated with basic belief formulae in the belief base. In a formal definition:

[1] Remember that a belief base is a conjunction of formulae that are all believed by the agent

Definition 22 *(extended semantics of belief formulae)*
Let $< \Omega, \sigma, \Pi, \theta >$ *be an agent configuration, let* $\phi \in L_B$ *and* $\boldsymbol{B}\phi_v^w \in L_{BQ}$*. Let* Bel_m *be the belief function defined for the mass function that is the result of the combination of all mass functions associated with basic belief formulae in* σ*. Then*

- $< \Omega, \sigma, \Pi, \theta > \models \boldsymbol{B}\phi_v^w \iff w \geq Bel_m(X \mid X \models^{\Omega} \phi) \geq v$

Note that $\neg\mathbf{B}\varphi_v^w$ means, that φ can not be deducted within the given margins v and w, but it might be deducted within larger margins. The complication with this interpretation is, that $\neg\mathbf{B}\varphi_v^w$ means that the probability of φ is either $< v$ or $> w$. To overcome this complication a formulation like $\mathbf{B}\varphi_v^1$ or $\mathbf{B}\varphi_0^w$ can be used, meaning that φ can be deducted with at least probability v or at most w, respectively.

Chapter 6

A prototype implementation

In this chapter I discuss a prototype implementation for a deduction of uncertain beliefs. The complete Prolog-code can be found in the appendix, here I discuss choices made in the implementation. First I provide an example of interaction with the Prolog interpreter to show the working of the program.

6.1 An example of interaction with the Prolog Interpreter

In this section I present an example of a short interaction with the Prolog program in the appendix. The program was consulted with SWI-Prolog, version 5.4.4, freely available for download. First, I enter some facts in the belief base.

```
?- add_fact(b1,0.4).
Yes
?- add_fact(b2,0.6).
Yes
?- add_fact(and(b2,b3),0.2).
Yes
?- add_fact(or(b3,and(b3,b1)),0.3).
Yes
?- show.
b1, 0.4
b2, 0.6
and(b2, b3), 0.2
or(b3, and(b3, b1)), 0.3
```

Based on this belief base, I calculate the belief in $b_1 \vee b_2$ and test, as a precondition of a basic action, whether this formula is deductable within certain limits:

```
?- support(or(b1,b2)).
0.808
?- pre(or(b1,b2),[0.6,0.9]).
Yes
?- pre(or(b1,b2),[0.9,1.0]).
No
```

Next, I add the formula $b_5 \wedge b_2$, with a probability of 0.2, to the belief base, as the postcondition of a basic action:

```
?- post([[and(b5,b2),0.2]]).
Yes
?- show.
b1, 0.4
b2, 0.6
and(b2, b3), 0.2
or(b3, and(b3, b1)), 0.3
and(b5, b2), 0.2
```

This added formula changes the support of the formula $b_1 \vee b_2$:

```
?- support(or(b1,b2)).
0.8464
```

This example of interaction shows the working of the program.

6.2 Prolog prototype source code

In this section I discuss the most important aspects of the Prolog source code. The entire source code can be found in the appendix. Here I describe the interaction with the user, the mass calculation and *Bel* function implementation, updating of the belief base, and the axioms that are used to test if a formula φ is a model of a formula *chi*.

6.2.1 Interaction

Interaction is provided by means of a number of clauses which are to be called by the user. The interaction with the belief base is provided by clauses which query the belief base, and can be called as pre-condition of a basic action, test goal or guard in a practical reasoning rule, and clauses which update the belief base, as a postcondition of a basic action. For the user's convenience, there are also clauses to show the entire belief base and to remove all beliefs, and to show the support for a certain proposition.

A query of the belief base is implemented as a test that verifies if the belief function for that particular query falls within an upper and lower limit, as discussed in the section on enhanced semantics (section 5.2).

```
%% empty belief base
rem_all :- retract(fact(_,_)),fail.

%% show belief base
show :- fact(X, S), write((X, S)), nl, fail.

%% show certainty of belief X?
support(X) :- calc_deductable(X, B), write(B) , nl.

%% test goal, guard and precondition are all implemented as
%% belief clause, with the condition to test and the
%% range of its limits as parameters

%% test goal
test(Cond, [Low, High]) :- bel(Cond, [Low, High]).

%% guard
guard(Cond, [Low, High]) :- bel(Cond, [Low, High]).

%% precondition
pre(Cond, [Low, High]) :- bel(Cond, [Low, High]).

%% our belief function: test if the probability of Cond is between Low and High.
%% it first calculates the support for Cond that is deductable from the
%% belief base, then tests if it is between the limits specified. Sup gets
%% instantiated in the clause calc_deductable.
bel(Cond, [Low, High]) :- calc_deductable(Cond, Sup), Sup >= Low, Sup =< High.

%% the postcondition is implemented as an update of the beliefbase. A set of
%% beliefs and/or negation of beliefs is given as its input. If a belief is
%% to be added to the belief base, the clause add_fact is called, otherwise
%% the clause rem_fact is called.
post([]).
post([[not(Belief),Prob]|More]) :- rem_fact(Belief, Prob), !, post(More).
post([[Belief,Prob]|More])      :- add_fact(Belief, Prob), post(More).
```

6.2.2 Querying

In this section, the mass calculation algorithm is implemented. The clause `calc_deductable` first finds all focal elements of the combined mass function that are subsets of the models of our query, i.e. if we want to find the mass for b_1 while our belief base has b_1 and b_2 as beliefs, the focal elements relevant to our query are b_1 and $b_1 \wedge b_2$. To be more precise, in order to calculate $Bel(X \mid X \models^{\Omega} b_1)$ we must find all X that satisfy $(X \subseteq \Omega \ \ \& \ \ \text{models}(X, b_1))$, and given our belief base, these are the maximal subsets of Ω that are models of b_1, respectively $b_1 \wedge b_2$.

After that, the list `PosList` is transformed to `PosBel` by calculating the mass value of each item, using the clause `get_mass`. The values in `PosBel` are then added to return the support of our query.

```
calc_deductable(Query, Bel) :-
  intersect(Query, PosList),
  maplist(get_mass, PosList, PosBel),
  addlist(PosBel, Bel).
```

The clause `intersect` builds a list of beliefs, generates the powerset of this belief list and then selects the subset of that powerset where all elements have the property that our query is deductable from it. For example, for a list $[b_1, b_2]$ and a query b_1, the powerset is $[b_1, b_2, [b_1, b_2], []]$ and the subset of this set from which elements b_1 is deductable, is the set $[b_1, [b_1, b_2]]$, which set I will denote as B_d.

```
intersect(Query, BelIntersect) :-
  findall(Belief, fact(Belief, _), Beliefs),
  bagof(B, entails(B, Query, Beliefs), BD),
  list_to_set(BD,BelIntersect).
```

The clause `get_mass` is called for every element in B_d. It first builds the list `AllBeliefs` from the belief base (B_a), and generates the list of beliefs that are *not* in B_d, namely the list B_{nd}. These two lists are precisely the bi-partition S_X^+ and S_X^- as mentioned in theorem 2.

Then, these lists are used to generate the probabilities of the elements of both sets (`SP` and `SM`), and the product of the items in `SP` and the product of $1-$ the items in `SM` are multiplied and returned in `Mass` . This procedure reflects the calculation mentioned in theorem 2.

```
get_mass(Beliefs, Mass) :-
  findall(Bel, fact(Bel,_), AllBeliefs),
  subtract(AllBeliefs, Beliefs, NotBeliefs),
  findall(Splus, get_belief(_,Splus,Beliefs), SP),
  findall(Sminus, get_belief(_,Sminus,NotBeliefs), SM),
  mullist(SP, SumP),
  mulnlist(SM, SumM),
  Mass is SumP * SumM.

%% select beliefs from the belief base that match the propositions in BelList.
get_belief(B,S,BelList) :- fact(B,S), member(B,BelList).
```

6.2.3 Updating

In a post-condition, a basic belief formula can be added to the belief base if it does not already exists, or recalculated if it does exist in the belief base. In that situation, a distinction is made whether the postcondition is evidence for or against the belief formula. Evidence for the belief formula is denoted as $\{ \varphi : p \}$, evidence against to formula as $\{ \neg\varphi : p \}$. Note, that $\{ \varphi : 0 \}$ and $\{ \neg\varphi : 0 \}$ have no effect[1], $\{ \varphi : 1 \}$ has the same result as adding the belief formula in 3APL without uncertainty, and $\{ \neg\varphi : p \}$ has the same result as removing the belief formula from the belief base.

```
%% add belief X with uncertainty S
add_fact(X, S) :- assert(fact(X, S)).   %% not deductable, just add
```

[1]This effective combines the already existing belief base with a mass function with $X = \Omega$ as the only clause with non-zero mass function, which has no effect on the mass assignment.

```
add_fact(X, S) :-
  fact(X, S1),                          %% deductable?
  dempster_add(S, S1, SNew),            %% yes, compute new value
  retract(fact(X, S1)),                 %% remove old fact
  assert(fact(X, SNew)).                %% add new one

%% remove belief X with uncertainty S
rem_fact(_, _) :- true.                 %% not deductable, take no action
rem_fact(X, S) :-
  fact(X, S1),                          %% deductable?
  dempster_rem(S, S1, SNew),            %% yes, compute new value
  retract(fact(X, S1)),                 %% remove old fact
  add_if_sup(X, SNew).                  %% add new one iff. P > 0

add_if_sup(X, SNew) :- SNew > 0, assert(fact(X, SNew)).
```

The assigned mass is calculated with the algorithm discussed in section 4.7.

```
%% functions to calculate new probability of a proposition based on
%% new evidence using dempster's rule of combination
dempster_add(Extra, Old, New) :- New is -((Old + Extra) , (Old * Extra)).
dempster_rem(Contra, Old, New) :- New is -(Old, (Contra * Old)).
```

6.2.4 Deductability

The `deductable(Query,Model)` tests if `Query` can be deducted from `Model`. It is implemented as axioms, for example $\Gamma \models \varphi \wedge \chi \Longleftrightarrow \Gamma \models \varphi \wedge \Gamma \models \chi$.

```
%% deductable is a list of axioms to determine if Q |=(cwa) P
deductable(_, (and(P, no(P)))) :- !,fail.
deductable(_, (and(no(P), P))) :- !,fail.
deductable(_, (or(P, no(P)))) :- !,true.
deductable(_, (or(no(P), P))) :- !,true.
deductable(P, [P]).
deductable(P, P).
deductable(no(no(P)), P).
deductable(P, no(no(P))).
deductable(P, and(P, _)).
deductable(P, and(_, P)).
deductable(or(P, _), P).
deductable(or(_, P), P).
deductable(_, (or(P, no(P)))).
deductable(_, (or(no(P), P))).
deductable(P, [Q|R])    :- deductable(P, Q) ; deductable(P, R).
deductable(P, of(Q, R)) :- deductable(P, Q), deductable(Q,R).
deductable(and(P,Q), R) :- deductable(P, R), deductable(Q, R).
deductable(or(P,Q), R)  :- deductable(P, R); deductable(Q, R).
deductable(impl(P,Q), R):- deductable(Q, R); deductable(no(P), R).
deductable(equ(P,Q), R) :- deductable(and(P,Q), R); deductable(and(no(P), no(Q)), R).
```

Chapter 7

Conclusion and future work

7.1 Conclusion

In section 2.5, the research question was formulated as follows: how can the belief base of an agent in 3APL be modified to hold uncertain beliefs, how can these beliefs be updated and how can belief queries be changed to reason with uncertain beliefs. After a discussion of Dempster-Shafer theory, these questions were theoretically addressed in chapter 4, in chapter 5 a syntax and semantics of modified 3APL was given, and in chapter 6 a prototype implementation was discussed.

While I have shown that Dempster-Shafer theory can be successfully applied to 3APL, and certain drawbacks of this theory can be overcome, some theoretical and practical problems still remain. These will be discussed in the following section.

7.2 Future work

In this section I suggest directions for future research on topics not addressed in this thesis. In particular, I discuss the 3APL interpreter, agent deliberation with uncertain beliefs, and the complexity of mass and belief functions.

7.2.1 Changing the 3APL interpreter

Based on the prototype implementation discussed in chapter 6, the current version of the 3APL interpreter and the development platform has to be enhanced. Not only the queries and updates of the belief base have to be enhanced, the syntax of the programming language must be altered in order to represent uncertain beliefs. In particular, the preconditions in the basic actions, the guards in practical reasoning rules, and the test goals need an interval that denotes a lower and upper limit as a test condition. The postconditions in the basic actions and the beliefs as they are represented in the belief base need a single value to denote their probability.

I propose to use the notation $\varphi[v, w]$ to represent the lower and upper limit of the deductability of φ, and the notation $\varphi : p$ to denote the probability of a belief. An example of these notations in a 3APL program with uncertainty can be found in the appendix.

7.2.2 Agent deliberation

During the execution of an agent program, deliberations on various types of decision take place, like planning a task, selecting a goal if multiple goals are possible, deciding to revise a plan or execute it, and so on. Normally, these deliberation issues are hard-coded in the programming platform, in traditional 3APL these decisions are fixed in the deliberation cycle. However in [8] it is argued, that these choices should be left to the programmer, and a programming language for the deliberation cycle is proposed. In this section, I investigate the consequences of uncertain beliefs for this approach, and I discuss the use of Partially Observable Markov Decision Processes[6] (hereafter POMDPs) in the deliberation cycle, to sketch possible further research in this direction.

In [8], a deliberation language is introduced, in order to give the agent designer full control over the deliberation process. A *plan* is defined as a sequence of practical reasoning rules, and a number of meta-statements are defined in order to execute a goal, select and apply a plan, generate, replace, and drop plans. The language also includes programming constructs to conditionally, respectively successively apply a meta-statement based on a belief formula.

Of course, the reasoning about plans can be fine tuned because we use a certainty measure on a belief instead of binary truth-values. For example, a practical reasoning rule as `Walk(X)` ← `rain` | `TakeBus(X)` could be refined to `Walk(X)` ← `rain[0.6,1.0]` | `TakeBus(X)`.

Uncertain beliefs can also be used to model careful or risk-taking agents. This would need an alternative notion of the guard in practical reasoning rules and a sort of utility function on these rules. For example, in an agent program which models a participant in the stock market, there could be rules to buy shares (with a possible high profit, but also with a lot of risks) or bonds (with less profit but also fewer risks). A careful agent would rather select the bond-rule if the certainty of high profit is low, and a risk-taking agent would select the share-rule even with such a low certainty.

POMDPs are Markov decision processes, with a set of *states*, a set of *actions*, a set of *effects* of these actions and a set of *immediate values* of these actions. In POMDPs, the actual state is not necessarily known to the agent. If utility functions are defined for sorting the practical reasoning rules, then uncertain beliefs can also be used as a probability distribution on the possible states in terms of POMDP, and the deliberation process can be modelled as a POMDP.

7.2.3 Complexity of belief functions

While I have proven in the thesis that calculating the mass value of a focal element has linear complexity (given certain constraints), I have not investigated efficient algorithms for determining which focal elements constitute the belief of a certain subset. The Prolog prototype code uses the rather coarse `findall` clause. Although my intuition says there should be an efficient algorithm, the complexity of the belief function in 3APL with uncertainty is subject to further investigations.

Appendices

Prolog implementation code

```
%% Modeling uncertainty in 3APL - Master's thesis Johan Kwisthout
%%
%% Prolog prototype code
%%
%% This software provides the algorithms described in the above thesis
%% to use uncertain beliefs in 3APL. User interaction is as follows:
%%
%% post([fact|more_facts]).      : change the belief base as a result of a postcondition.
%%     fact                      : [proposition, probability]
%%     proposition               : p | not(p) | and(p,q) | or(p,q) | impl(p,q) | equ(p,Q
%%
%% pre(condition, [low, high]).  : returns true if the given condition can be derived
%%                                 with a certainty within the interval
%%
%% show.                         : show the beliefbase
%% rem_all.          : empty the beliefbase
%% support(proposition).         : show the support for a certain proposition

%% let the prolog engine know 'fact' is dynamic
:- dynamic fact/2.

%% multi-purpose functions

conc([],L,L).
conc([X|L1],L2,[X|L3]) :- conc(L1,L2,L3).

subs([],[]).
subs(S,[_|T]) :- subs(S,T).
subs([H|S],[H|T]) :- subs(S,T).

%% multiply all items in a list and put result in Total
mullist([], 1).
mullist([L1|L2], Total) :- mullist(L2,L3), Total is L1 * L3.

%% multiply (1 - p) for all items in a list and put result in Total
mulnlist([], 1).
mulnlist([L1|L2], Total) :- mulnlist(L2,L3), Total is (1 - L1) * L3.

%% add all items in a list and put result it total
addlist([], 0).
addlist([L1|L2], Total) :- addlist(L2,L3), Total is L1 + L3.

%% entails returns a list of items from Beliefs from which Query is deductable
entails(B, Query, Beliefs) :- subs(B, Beliefs), deductable(Query, B).

%% deductable is a list of axioms to determine if Q |=(cwa) P
deductable(_, (and(P, no(P)))) :- !,fail.
deductable(_, (and(no(P), P))) :- !,fail.
```

```
deductable(_, (or(P, no(P)))) :- !,true.
deductable(_, (or(no(P), P))) :- !,true.
deductable(P, [P]).
deductable(P, P).
deductable(no(no(P)), P).
deductable(P, no(no(P))).
deductable(P, and(P, _)).
deductable(P, and(_, P)).
deductable(or(P, _), P).
deductable(or(_, P), P).
deductable(_, (or(P, no(P)))).
deductable(_, (or(no(P), P))).
deductable(P, [Q|R])    :- deductable(P, Q) ; deductable(P, R).
deductable(P, of(Q, R)) :- deductable(P, Q), deductable(Q,R).
deductable(and(P,Q), R) :- deductable(P, R), deductable(Q, R).
deductable(or(P,Q), R)  :- deductable(P, R); deductable(Q, R).
deductable(impl(P,Q), R):- deductable(Q, R); deductable(no(P), R).
deductable(equ(P,Q), R) :- deductable(and(P,Q), R); deductable(and(no(P), no(Q)), R).

%% specific helperfunctions for the post-condition part

%% functions to calculate new probability of a proposition based on
%% new evidence using dempster's rule of combination
dempster_add(Extra, Old, New) :- New is -((Old + Extra) , (Old * Extra)).
dempster_rem(Contra, Old, New) :- New is -(Old, (Contra * Old)).
%% add new belief
add_if_sup(X, SNew) :- SNew > 0, assert(fact(X, SNew)).

%% add belief X with uncertainty S
add_fact(X, S) :-
  fact(X, S1),                          %% deductable?
  dempster_add(S, S1, SNew),            %% yes, compute new value
  retract(fact(X, S1)),                 %% remove old fact
  assert(fact(X, SNew)).                %% add new one

add_fact(X, S) :- assert(fact(X, S)).   %% not deductable, just add

%% remove belief X with uncertainty S
rem_fact(X, S) :-
  fact(X, S1),                          %% deductable?
  dempster_rem(S, S1, SNew),            %% yes, compute new value
  retract(fact(X, S1)),                 %% remove old fact
  add_if_sup(X, SNew).                  %% add new one iff. P > 0

rem_fact(_, _) :- true.                 %% not deductable, ok

%% The next 4 functions form the core of our algorithm:
%%  get_mass(list of beliefs in the mass element, Mass)
%%  computes e.g. m(b1 & b2), where Beliefs = [b1,b2].

%% select beliefs from the belief base that match the propositions
%% in BelList
get_belief(B,S,BelList) :- fact(B,S), member(B,BelList).

%% calculate the mass of the conjunction based on Beliefs
get_mass(Beliefs, Mass) :-
  findall(Bel, fact(Bel,_), AllBeliefs),
  subtract(AllBeliefs, Beliefs, NotBeliefs),
  findall(Splus, get_belief(_,Splus,Beliefs), SP),
  findall(Sminus, get_belief(_,Sminus,NotBeliefs), SM),
```

```
  mullist(SP, SumP),
  mulnlist(SM, SumM),
  Mass is SumP * SumM.

%% The next 3 functions determine which mass elements we need.
%% First we generate elements, then we determine which of them
%% satisfy phi |= query, then we calculate the mass of these
%% elements and add them.

%% generate intersections that must be used to calculate Bel(Query)
intersect(Query, BelIntersect) :-
  findall(Belief, fact(Belief, _), Beliefs),
  bagof(B, entails(B, Query, Beliefs), BD),
  list_to_set(BD,BelIntersect).

%% calculate the probability of Query
calc_deductable(Query, Bel) :-
  intersect(Query, PosList),
  maplist(get_mass, PosList, PosBel),
  addlist(PosBel, Bel).

%% interface functions for interaction

%% empty belief base
rem_all :- retract(fact(_,_)),fail.

%% show belief base
show :- fact(X, S), write((X, S)), nl, fail.

%% show certainty of belief X?
support(X) :- calc_deductable(X, B), write(B) , nl.

%% test goal
test(Cond, [Low, High]) :- bel(Cond, [Low, High]).

%% guard
guard(Cond, [Low, High]) :- bel(Cond, [Low, High]).

%% precondition
pre(Cond, [Low, High]) :- bel(Cond, [Low, High]).

%% our belief function: test if the probability of Cond is between Low and High?
bel(Cond, [Low, High]) :- calc_deductable(Cond, Sup), Sup >= Low, Sup =< High.

%% postcondition: update the beliefbase
post([]).
post([[not(Belief),Prob]|More]) :- rem_fact(Belief, Prob), !, post(More).
post([[Belief,Prob]|More])      :- add_fact(Belief, Prob), post(More).
```

A simple 3APL program with uncertain beliefs

This simple example of a 3APL program with uncertain beliefs shows an agent that eats until it is almost certainly not hungry anymore. The basic action *Eat()* halves the possibility of the agent being hungry, which models the fact that eating something doesn't make sure one is completely satisfied. In traditional 3APL, this would be impossible to model. We might have *gradual* increase of satisfaction (e.g. from veryHungry via moderateHungry to slightlyHungry) but this still are certain known facts instead of certainty measures.

```
PROGRAM "eating_agent.3apl"

CAPABILITIES:
{ hungry[0.1,1.0] } Eat() { NOT hungry:0.5 }

BELIEFBASE:
hungry:0.8

GOALBASE:
satisfied()

RULEBASE:
satisfied() <- hungry[0.1,1.0] | Eat(); satisfied(),
satisfied() <- hungry[0.0,0.1] |.
```

The program flow of this 3APL program will be as follows

Time	Goal base	Belief base
t_0	satisfied	hungry:0.4
t_1	satisfied	hungry:0.2
t_2	satisfied	hungry:0.1
t_3	empty	hungry:0.1

Symbols and definitions

List of symbols used

$\Gamma \models \phi$	Propositional satisfaction relation: the models of Γ are a subset of the models of ϕ
$\square$	Symbol to denote the end of a proof
$\oplus$	Orthogonal sum or combination
$\forall x\phi$	For all x in ϕ (universal quantor)
$\exists x\phi$	For at least one x in ϕ (existential quantor)
$\top$, $\bot$	Formula that is always true, resp. false
$\phi \wedge \psi$, $\phi \vee \psi$, $\neg\phi$	Logical conjunction, disjunction, resp. negation
$A \cap B$, $A \cup B$, A^c	Intersection, union, resp. complement of sets
$A \backslash B$	The set that has all elements of A that are not elements of B
$\Box$, $\Diamond$	Necessity resp. possibility operator
$\equiv$	Equivalence relation
$X \models^{\Omega} \varphi$	X is the maximal subset of Ω consisting of models of φ

Index of definitions introduced in this thesis

Index of theorems and proofs introduced in this thesis

Short introduction to modal logic

In this appendix, I give a short introduction in modal logic to explain the concepts used in this thesis to computer science readers who lack this knowledge. Modal logic is built on the concept of a *model* M, which is a set of possible worlds w which give a valuation to a certain propositional formula. For example, for the formula p there are two possible worlds: one in which p is true, and one in which p is false. Likewise, there are four possible worlds for the formule $p \wedge q$. If a formula in true in at least one possible world, the formula is called *possible* (normally denoted with $\Diamond p$), if it is true in all possible worlds the formula is called *necessary*, denoted as $\Box p$.

In modal logic, the following statements are equivalent:

- $\Sigma \models \varphi$
- For every model M and world w, if $M, w \models \Sigma$, then $M, w \models \varphi$
- The set of worlds in which Σ is true, is a subset of the set of worlds in which φ is true

We can specify the notions of possibility and necessity in a certain context. In *epistemic logic*, $\Box\varphi$ is translated to 'I (or: the agent) *know* that φ' or, alternatively, 'I *belief* that φ', noted as $K\varphi$ respectively $B\varphi$. Although there is no alternative symbol for $\Diamond\varphi$, it can be translated as $\neg K\neg\varphi$, meaning 'I do not know that $\neg\varphi$. In another context, namely *temporal logic*, the formulae $\Box\varphi$ and $\Diamond\varphi$ are translated as 'from now on φ' respectively 'in the future φ'.

In Computation Tree Logic, there exist *state formulae*, which describe a certain point in the time tree, and *path formulae*, which describe a path in this tree. The modal operators *necessary* and *optionally* are defined on path formulae, and denote that the formula holds in *all* paths from this tree, respectively in *at least one* such path. There are also modal operators which describe temporal aspects in state formulae. The possibility and necessity operator have the meaning 'at any point in this path' respectively 'at all points in this path'. We can describe sentences like 'If the bus is delayed, I'll never catch my train' with this Computational Tree Logic. There are multiple possible paths, suppose one of them denoting a bus delay. Let 'Catch(train)' mean 'I catch my train', and let $\Box_P$, $\Box_S$, $\Diamond_P$, and $\Diamond_S$ denote the necessity and possibility operator on path and state goals, respectively. Then we can describe this sentence as $\Diamond_P\neg\Diamond_S$ Catch(train). This means, there is at least one path in which the formula Catch(train) will never be true.

Short introduction to complexity theory

In this appendix I will give a short introduction on computational complexity theory, a field in Computer Science that studies the computation time and space required to solve a certain problem. A more comprehensive survey can be found in e.g.[11].

To formalize the notion of computation and 'solving a problem', the abstract concept of a Turing Machine is used. Such a Turing Machine M computes the function $f_M : \{0,1\}^\star \rightarrow \{0,1\}^\star$, defined by $f_M(x) = y$, if M halts with output y when given input x. In other words, it is assumed that the computational problem can be translated to a function which transforms a certain input string to a certain output string, where both strings are a representation of the problem and its solution.

Computational decision problems can be distinguished by their complexity class. Amongst other, important classes are:

- P
- NP
- NP-complete
- NP-hard
- $\#P$-complete

The class P consists of problems that are solvable in polynomial time, i.e. there exists a Turing Machine M that computes $f_M(x)$ in polynomial time with respect to the length of X. An example of this class is: *Given a number n, decide whether n is prime.* It is easy to see that this question can be decided using n divisions.The class NP consists of problems $f_M(x) = y$ where the *proof* that y is a solution of $f_M(x)$ can be checked in polynomial time. For example, while we cannot determine which assignment to variables make a given logical formula true in polynomial time[1], we can certainly check whether a solution y to this problem is correct in polynomial time. Although it seems very plausible (at least) that the class P is not equal to NP, the question whether $P = NP$ is still open. A one million dollar prize has been offered to the first person who proves or disproves $P = NP$.

Some special classes of problems are $NP-complete$ and $NP-hard$. A problem is $NP-complete$ if it is in NP, and every problem in NP can be translated to this problem, so if we would have a polynomial-time solution to an $NP-complete$ problem, we would have a solution to *all* NP problems. An example of an NP-complete problem is: 'Does a variable assignment exists that will satisfy a given logical formula in DNF-format?'. An NP-hard problem has the same characteristics, apart from that it does not have to be an NP-problem itself. All NP-problems can be reduced to the question 'Given a program and its input, will it run forever?' (the famous Halting Problem), but it has been proven that the halting problem is undecidable and therefore it is not in NP.

The question 'Is there a variable assignment that will satisfy a given logical formula in DNF-format?' can be of course be extended to '*How many* variable assignments exist that will satisfy a given logical formula in DNF-format?'. This class of problems is referred to as $\#P$-class of problems (pronounced *sharp-P*).

[1] To be more precise: there is no polynomial-time algorithm known, and it is believed that none exists

Bibliography

[1] J.F. Baldwin, J. Lawry, and T.P. Martin. A mass assignment theory of the probability of fuzzy events. *Fuzzy Sets Syst.*, 83(3):353–367, 1996.

[2] J.A. Barnett. Computational methods for a mathematical theory of evidence. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 868–875, 1981.

[3] J.A. Barnett. Calculating Dempster-Shafer plausibility. *IEEE transactions on pattern analysis and machine intelligence*, 13:599–603, 1991.

[4] L. Braubach, A. Pokahr, and W. Lamersdorf. Jadex: A short overview. In *Main Conference Net.ObjectDays 2004*, pages 195–207, 9 2004.

[5] B.G. Buchanan and E.H. Shortliffe. *Rule-based expert systems: The MYCIN experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, Reading, MA, 1984.

[6] A.R. Cassandra. *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, Brown University, Providence, R.I., 1998.

[7] G. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42(2-3):393–405, 1990.

[8] M. Dastani, F. de Boer, F. Dignum, and J-J.Ch. Meyer. Programming agent deliberation: an approach illustrated using the 3APL language. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'03), Melbourne*, pages 97–104, 2004.

[9] M. Dastani, B. van Riemsdijk, F. Dignum, and J.-J. Meyer. A programming language for cognitive agents:goal directed 3APL. In M. Dastani, J. Dix, and A. El Fallah-Seghrouchni, editors, *Proceedings of the First Workshop on Programming Multiagent Systems:Languages, frameworks, techniques, and tools (ProMas03)*, LNAI 3067, 2004.

[10] D. Dubois, J. Lang, and H. Prade. Possibilistic logic. In *Handbook of logic in artificial intelligence and logic programming (vol. 3): nonmonotonic reasoning and uncertain reasoning*, pages 439–513. Oxford University Press, Inc., New York, NY, USA, 1994.

[11] O. Goldreich. Complexity theory - a survey. Published on the Internet at http://www.wisdom.weizmann.ac.il/∼oded/cc-sur1.html, retrieved 05/29/05, 2000.

[12] F.V. Jensen. *Bayesian networks and decision graphs*. Springer-Verlag, New York, Inc., 2001.

[13] A. Jøsang. The consensus operator for combining beliefs. *Artificial Intelligence Journal*, 142(1-2):157–170, 2002.

[14] B. Milch and D. Koller. Probabilistic models for agent's beliefs and decisions. In *UAI '00: Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 389–396, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[15] A.F. Moreira and R.H. Bordini. An operational semantics for a BDI agent-oriented programming language. In J.-J. C. Meyer and M. J. Wooldridge, editors, *Proceedings of the Workshop on Logics for Agent-Based Systems (LABS-02), April 22-25, Toulouse, France*, pages 45–59, 2002.

[16] L. Morgenstern. The problem with solutions to the frame problem. In Kenneth M. Ford and Zenon Pylyshyn, editors, *The Robot's Dilemma Revisited: The Frame Problem in Artificial Intelligence*, pages 99–133. Ablex Publishing Co., Norwood, New Jersey, 1996.

[17] P. Orponen. Dempster's rule of combination is #P-complete. *Artificial Intelligence*, 44:245–253, 1990.

[18] L. Padgham and M. Winikoff. *Developing Intelligent Agent Systems: A Practical Guide*. John Wiley and Sons, 2004.

[19] S. Parsons and P . Giorgini. An approach to using degrees of belief in BDI agents. In B. Bouchon-Meunier, R.R. Yager, , and L.A. Zadeh, editors, *Information, Uncertainty, Fusion*. Kluwer, Dordrecht, 1999.

[20] S. Parsons, N.R. Jennings, J. Sabater, and C. Sierra. Agent specification using multi-context systems. In *Selected papers from the UKMAS Workshop on Foundations and Applications of Multi-Agent Systems*, pages 205–226, London, UK, 2002. Springer-Verlag.

[21] A.J. Rao and M.P. Georgeff. Modelling rational agents within a BDI-architecture. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*. Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[22] K. Sentz. *Combination of Evidence in Dempster-Shafer Theory*. PhD thesis, Binghamton University, 2002.

[23] G. Shafer. *A Mathematical Theory of Evidence*. Princeton Univ. Press, Princeton, NJ, 1976.

[24] P. Smets. The combination of evidence in the transferable belief model. *IEEE Pattern Analysis and Machine Intelligence*, 12:447–458, 1990.

[25] M.J. Smithson. *Ignorance and Uncertainty: Emerging Paradigms*. Cognitive Science Series. Springer Verlag, New York, 1989.

[26] M. Verbeek. 3APL as programming language for cognitive robotics. Master's thesis, Utrecht University, 2003.

[27] N. Wilson. Algorithms for Dempster-Shafer theory. In D.M. Gabbay and P. Smets, editors, *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, volume 5: Algorithms, pages 421–475. Kluwer Academic Publishers, 2000.

[28] M. Wooldridge. Intelligent agents. In G. Weiss, editor, *Multiagent Systems*. The MIT Press, 1999.

[29] R. Yager. On the Dempster-Shafer framework and new combination rules. *Information Sciences*, 41:93–137, 1987.