Faculty of Science
Department of Applied Mathematics,
Computer Science & Statistics
Computational Web Intelligence

Faculty of Science and
Bio-Engineering Sciences
Department of Computer Science
Theoretical Computer Science

Epistemic Extensions of Answer Set Programming

Dissertation submitted in fulfilment of the requirements for the degree of Doctor of Science: Computer Science

Kim Bauters

July 2013

Promotors: Prof. Dr. Martine De Cock (Universiteit Gent)
Prof. Dr. Dirk Vermeir (Vrije Universiteit Brussel)
Dr. Steven Schockaert (Cardiff University)

Abstract

Answer Set Programming (ASP) is a declarative programming language based on the stable model semantics and geared towards solving complex combinatorial problems. The strength of ASP stems from the use of a non-monotonic operator. This operator allows us to retract previously made conclusions as new information becomes available. Similarly, in common-sense reasoning, we may arrive at conclusions based on the absence of information. When an animal is for example a bird, and we do not know that this bird is a penguin, we conclude that the bird can fly. When new knowledge becomes available (e.g. the bird is a penguin) we may need to retract conclusions. However, while ASP similarly allows us to revise knowledge, it is not an ideal framework to model common-sense reasoning. For example, in ASP we cannot model multi-context systems, where each context encodes a different aspect of the real world. Extensions of ASP have been proposed to model such multi-context systems, but the exact effect of communication on the overall expressiveness remains unclear. In addition, ASP lacks the means to easily model and reason about uncertain information. While extensions of ASP have been proposed to deal with uncertainty, namely Possibilistic Answer Set Programming (PASP), there are contexts in which the current semantics for PASP lead to unintuitive results.

In this thesis we address these issues in the followings ways. Firstly, we introduce Communicating Answer Set Programming (CASP), which is a framework that allows us to study the formal properties of communication and the complexity of the resulting system in ASP. It is based on an extension of ASP in which we consider a network of ordinary ASP programs. These communicating programs are extended with a new kind of literal based on the notion of asking questions. As such, one ASP program can conceptually query another program as to whether it believes some literal to be true or not, i.e. they can communicate. For the least complex variant of ASP, simple programs, it is shown that the addition of this easy form of communication allows us to move one step up in the polynomial hierarchy. Furthermore, we modify the communication mechanism to also allow us to focus on a sequence of communicating programs, where each program in the sequence may successively remove some of the remaining models. This mimics a sequence of leaders, where the first leader has the first say and may remove models that he or she finds unsatisfactory. Using this particular communication mechanism allows us to capture the entire polynomial hierarchy.

Secondly, we show how semantics for PASP can be defined in terms of constraints on possibility distributions. These new semantics adhere to a different intuition for negation-as-failure than current work on PASP to avoid unintuitive conclusions in specific settings. In addition, since ASP is a special case of PASP in which all the rules are entirely certain, we obtain a new characterization of ASP in terms of constraints on possibility distributions. This allows us to uncover a new form of disjunction, called weak disjunction, that has not been previously considered in the ASP literature. When examining the complexity of weak disjunction we unearth that, while the complexity of most reasoning tasks coincides with disjunction in ordinary ASP, some decision problems are easier.

Thirdly, we highlight how the weight attached to a rule in PASP can be interpreted in different ways. On the one hand, the weight can reflect the certainty with which we can conclude the head of a rule when its body is satisfied. This corresponds with how the weight is understood when defining semantics for PASP in terms of constraints on possibility distributions. On the other hand, the weight can reflect the certainty that the rule itself is correct. ASP programs with incorrect rules may have erroneous conclusions, but due to the non-monotonic nature of ASP, omitting a correct rule may also lead to errors. To derive the most certain conclusions from an uncertain ASP program, we thus need to consider all situations in which some, none, or all of the least certain rules are omitted. This corresponds to treating some rules as optional and reasoning about which conclusions remain valid regardless of the inclusion of these optional rules. Semantics for PASP are introduced based on this idea and it is shown that some interesting problems in Artificial Intelligence can be expressed in terms of optional rules.

For both CASP and the new semantics for PASP we show that most of the concepts that we introduced can be simulated using classical ASP. This provides us with implementations of these concepts and furthermore allows us to benefit from the performance of state-of-the-art ASP solvers.

Contents

Acknowledgments

Working on a PhD, writing a PhD thesis and research in general is a considerable undertaking. Without the support of many people, it would simply not be possible. Luckily for me, my academic pedigree contains some impressive people. I would like to thank my supervisors *Martine De Cock*, *Dirk Vermeir* and *Steven Schockaert*. Each member of this *dream team* has made significant contributions to my PhD and, in general, my interest in research. Dirk has guided me through this fascinating world of research ever since my master thesis. I thank him for his elaborate knowledge and his no-nonsense approach. Martine has guided me throughout my PhD and has been an excellent advisor. I thank Martine for her impeccable attention to detail and, on a personal level, for her experiences on having a spouse that works abroad. Last, but most certainly not least, is Steven. I am uncertain whether my PhD thesis would have achieved this level of quality if Steven had not been involved. I would like to thank him for driving me towards complexity theory and possibility theory, two areas of research of which I am confident that they will keep me captivated in the future.

I also had the wonderful opportunity to work with two exceptional Frenchmen over these last few years. I would like to thank *Pascal Nicolas*, who has sadly passed away, for his many enlightening ideas when it came to both communicating and possibilistic answer set programming. I would also like to thank *Salem Benferhat*, as he reignited my interest in everything possibilistic, as well as my interest in French and working abroad. If I ever end up working in Quebec, Canada, he will be the person that is to *blame* and I can only thank him for that. I strongly hope that in the future we will once more work together as there is still so much that I can learn from him.

Many colleagues from both UGent and VUB deserve a special mention as well, but there are just too many to list them all. People such as *Patricia Victor*, *Chris Cornelis*, *Timur Fayruzov*, *Nele Verbiest* and *Gustavo Torres Torres* are not merely colleagues that helped me move forward in my research, even though they were often not aware of this themselves, but are first and foremost the people with whom you make some wonderful small talk. *Jeroen Janssen*, *Marjon Blondeel* and *Sofie De Clercq* deserve special recognition in this list as we have talked for hours and hours, both on and off about research, and they are the kind of colleagues that make going to work an even greater pleasure. *Ruben Hillewaere*, and especially his music, has driven me forward when the pile of work got the better of me. Finally, there are my office mates. *Reza Khoshsiar* and *Eti Wiraningsih* demonstrated how research truly is a multi-cultural endeavour and they ignited my passion to see the world. *Charlotte Sonck* and *Virginie de Witte* showed what true colleagues really are and provided a listening ear whenever I had any kind of qualms. Even if that meant talking for hours on end, for which I do apologise.

When we look beyond the work environment, I must mention my godmother *Martine Dekeyzer* and relative *Agnes Carbonelle*. They are the kind of people that are important in your life, yet you take them for granted. Nevertheless, they watch your back at just the right times and offer encouragement whenever it is needed most. Two of my best friends, *Karon Pante* and *Cedric Seynaeve*, should not be overlooked and I can only thank them for their enduring friendship. A very special thanks furthermore goes out to my grandmother *Agnes Dejaeghere*. Ever-proud and ever-friendly, she has been my tower of strength throughout the years. She often buried herself away in favour of others, offered me a place to stay and be myself and never questioned my actions. If not because of her, I might never have ended up studying Computer Science. Sadly, she passed away suddenly and, as it happens so often, too soon. Her lasting memory remains forever vibrant within me.

I would be nothing without my parents, who nurtured me, shaped me and stood by my side throughout all these years. To them I owe my particular taste for travelling, my desire to excel, my desire to learn and my love for animals. Though at times we may not have seen eye to eye, we have always emerged stronger than before and the bond I have with them has only strengthened throughout the years. Similar thanks go out to my parents-in-law, who shaped me in many different ways and who let me see the world from a different angle.

Most important of all is *Line Caes*, my wife, to whom I have lost my heart. Witty, intelligent, beautiful and successful would be but a few of the words that pop up when asking to describe her. For better of for worse she stands by my side, unconditionally. Together we have travelled the world and I can only hope that we will continue to do so until the end of our days. Finally, *Hera* and *Iachi*, our two beloved dogs, deserve a mention on their own as their love is, as Line's love, unconditional.

1 | Introduction

Throughout history there has been significant interest in trying to model how human beings think and how they reason. Logic, in particular, is a very broad field of study that is concerned with formalizing the different ways in which we reason and what can be considered as valid forms of reasoning [Jacquette 2005]. It should therefore come as no surprise that theories of logic have been developed independently throughout the world[1]. Also, logic has been applied to many areas of science such as mathematics, philosophy, linguistics and computer science. The earliest study of formal logic in the Western world is credited to Aristotle, who wrote his famous *Analytica Priora* in the 4th century BC. In this work, the polymath Aristotle made significant contributions to the study of logic[2]. One such contribution is the idea of a syllogism, which is a logical argument where the conclusion is supported by two or more premises. An example of such a syllogism is the following:

> If it is raining, I will go see a movie
> It is raining
>
> therefore
>
> I will go see a movie

This is actually a special case of a syllogism, namely a hypothetical syllogism or modus ponens, which is a rule of inference. Specifically, modus ponens is a type of reasoning

[1]Source: https://en.wikipedia.org/wiki/Logic, retrieved March 8, 2013.
[2]Source: http://plato.stanford.edu/entries/aristotle/, retrieved March 8, 2013.

of the form "A *is true. When* A *is true, we can infer* B*. Therefore we know that* B". For this example, we have that A = "it is raining" and B = "I will go see a movie". As an alternative, we could also write this in a more compact form as:

$$\begin{array}{l} raining \rightarrow movies \\ raining \\ \hline movies \end{array}$$

The foundations laid out by Aristotle formed what is now known as Aristotelian logic, which remained the de facto logic for many centuries to come[3]. It was only in the 19th century, when the English mathematician George Boole presented his work *Laws of Thought*, that new insights would challenge the dominance of Aristotelian logic[4]. Boole proposed to express logical propositions as algebraic expressions. The algebraic manipulation of these expressions would then provide a fail-safe method of logical deduction. As such, in Boolean algebra, the values of the variables can either be true (1) or false (0). Furthermore, rather than operators such as addition, multiplication and subtraction, we use the operators conjunction $\wedge$, disjunction $\vee$ and negation $\neg$ [Corcoran 2003]. The behaviour of each of these operators can, in turn, be defined in relation to the truth of the subcomponents of its arguments. For example, if we know that $rain$ is true and $cold$ is true, we know that the conjunction $rain \wedge cold$ must also be true. Importantly, the work from Boole is seen as the starting point of the study of mathematical logic and fuelled the continuous study of logic [Corcoran 2003]. Logic now also became a mathematical study, rather than a purely philosophical one.

The development of Boolean algebra was a fundamental step towards the development of computer science, as e.g. Claude Shannon showed in 1937 how Boolean algebra could be used to improve the design of systems with electronic relays and how such relays could be used to solve Boolean algebra problems[5]. As such, it became possible to use electronic relays (which are the precursors to modern computer chips) to perform logical deduction. This led to the development of the world's first working, programmable and automated computers. Programming languages were developed to interact with these computers. Most of these programming languages are imperative programming languages, i.e. the programmer needs to describe both *what* the program should do and *how* the program should accomplish this goal.

[3]Source: http://plato.stanford.edu/entries/aristotle-logic/, retrieved March 8, 2013.

[4]It is important to note that Boole was not set to disprove Aristotelian logic. Rather, Boole approached logic differently and ended up supplementing the work of Aristotle. This is evident in modern logics, where we see influence of both Aristotelian logic and Boolean algebra.

[5]Source: http://dspace.mit.edu/bitstream/handle/1721.1/11173/34541425.pdf, retrieved March 8, 2013.

Declarative programming languages[6], on the other hand, express the logic of the computation without describing the control flow. The programmer therefore only needs to describe what the program should do, while it is left to the implementation of the programming language to define how this will be accomplished. Most programming languages based on logic are declarative programming languages. Prolog [Wielemaker et al. 2012] is an example of such a programming language and is based on a subset of first-order logic, namely Horn clauses, which are rules of the form:

$$conclusion \leftarrow premise_1, ..., premise_n.$$

Horn clauses are in some sense the modern equivalent of syllogisms [Flach and Kakas 2000] and the inference in Prolog is based on the modus ponens syllogism. Originally designed for natural language processing, Prolog also proved to be well-suited for building expert systems, control systems, etc. Other programming languages, such as (subsets of) SQL and regular expressions, are not based on logic but are nevertheless declarative. However, whereas Prolog is a general-purpose language, these languages are domain specific. Domain-specific languages are programming languages that are created specifically to solve problems in one application domain and are not meant to be used outside of that application domain[7]. Such languages often allow the problem to be expressed more clearly than with a general purpose language, as they can already incorporate features specific to that particular domain.

Answer Set Programming (ASP), which will be used throughout this thesis, is a declarative domain-specific programming language for solving hard combinatorial problems. While similar to Prolog, ASP is not Turing-complete, i.e. not all the problems solvable on a Turing machine (which we discuss in more detail in Chapter 2) can be solved using ASP. As a benefit, given a good implementation, an ASP avoids the halting problem[8], i.e. an ASP program always finishes running. In this thesis, we will look at a number of epistemic extensions of ASP. As such, we describe the history and ideas that underlie ASP and epistemic reasoning in more detail in Section 1.1 and Section 1.2. The extensions of ASP that we introduce are Communicating Answer Set Programming (CASP) and Possibilistic Answer Set Programming (PASP). CASP is a framework that allows for a number of ASP programs to communicate and as such solve more complex problems. In particular, we investigate which communication mechanisms are necessary to allow for an increase in the expressive power of ASP. We then look at semantics for PASP, which is an extension of ASP that allows for reasoning under uncertainty based on possibility theory, of which we outline the underlying ideas in Section 1.3.

[6]Source: https://en.wikipedia.org/wiki/Declarative_programming, retrieved March 8, 2013.
[7]Source: https://en.wikipedia.org/wiki/Domain-specific_language, retrieved March 8, 2013.
[8]Source: http://en.wikipedia.org/wiki/Halting_problem, retrieved April 21, 2013

1.1 Answer Set Programming

To build systems capable of common-sense reasoning, we need a programming language that offers the capability to both represent knowledge and reason about this knowledge. A language based on logic seems a natural approach to solve this particular problem. However, logics such as Boolean algebra and Aristotelian logic are monotonic, i.e. earlier conclusions cannot be retracted when new information becomes available. Nevertheless, such non-monotonicity occurs naturally in human reasoning. For example, in general, we accept that birds fly. However, if we learn that a particular bird is a penguin, we revise our conclusion because a penguin is an exception to the general rule that all birds fly.

To overcome these problems, research has since the late 1960's focused on non-monotonic logics, which are a family of approaches designed to be able to reason in a defeasible way[9]. Essentially, defeasible inference allows us to infer conclusions that are rationally compelling but not necessarily deductively valid. Or, in other words, we want to accept conclusions when they are consistent with our current knowledge of the problem. This type of human reasoning is widely used and does not affect our ability to act rationally. If a person is harmed when crossing the street – after carefully checking that no cars were in the vicinity – by a meteoroid, no one would question his rationality. Still, crossing the road proved not to be safe. The research on defeasible logic gained significant attention in the domain of Artificial Intelligence (AI) in the late 1960's, when problems were discovered when working on expert systems [Mccarthy and Hayes 1969] and the modelling of such systems with monotonic logics. Since then, many non-monotonic logics have been proposed, including circumscription [McCarthy 1980], default logic [Reiter 1980], autoepistemic logic [Moore 1985] and stable models for negation-as-failure [Gelfond and Lifschitz 1988]. ASP falls in this last category and is also often referred to as the stable model semantics.

Negation-as-failure is a special construct denoted as '$not\ l$', where, intuitively, '$not\ l$' is true when we cannot prove that 'l' is true. This allows us to write

$$\mathit{fly}(X) \leftarrow \mathit{bird}(X), \mathit{not}\ \mathit{penguin}(X)$$

which elegantly captures the common-sense knowledge that birds fly, unless they are known to be penguins. Still, while negation-as-failure captures an intuitive idea, it proved to be hard to find acceptable semantics for negation-as-failure. At the heart of this problem lay two competing ideas of what a logic program without negation-as-failure represents. According to [Van Emden and Kowalski 1976], such a program represents the least Herbrand model, i.e. the set of all atomic logical consequences of the program. However, in [Clark 1978] the program is seen as a representation of the completion of the program,

[9]While non-monotonic logics have been studied intensely since the late 1960's, the first studies of defeasible reasoning date back to Aristotle and his work on dialectic reasoning in *Posterior Analytic*.

which, loosely speaking, interprets the implication as an equivalence (or, symbolically, $\leftarrow$ is interpreted as $\equiv$). Because the latter understanding often resulted in unwanted models, the least Herbrand models turned out to provide a better semantics for logic programs without negation-as-failure. However, when logic programs include negation-as-failure, the semantics based on the least Herbrand models resulted in unwanted models, whereas the completion gave the more satisfactory results. Finding a semantics that combined these two ideas, and that had no unwanted models whether negation-as-failure was present or not, took more than a decade of research [Lifschitz 2008].

The stable model semantics, which solve this apparent enigma, were initially defined in terms of autoepistemic logic [Moore 1985]. Autoepistemic logic introduces a modal operator L where La intuitively means "*a is believed [to be true]*". As such, in autoepistemic logic, we are able to reason about our own knowledge and beliefs. The semantics of autoepistemic logic are defined in terms of stable expansions, where the stable expansion w.r.t. a set of axioms A intuitively corresponds with that which a rational agent might believe if he knew A [Moore 1985]. In [Gelfond 1987] the expression '$not\ a$' was identified as '$\neg$La', i.e. "*It is not believed that a [is true]*". Similarly, it is possible to define the stable model semantics in terms of default logic. However, in both cases, the stable model semantics rely on an underlying logic, which made it difficult to be used in practice.

Research then focused on finding an equivalent definition of the stable model semantics that did not explicitly rely on an underlying logic. This resulted in the definition of the stable model semantics in terms of a reduct [Gelfond and Lifschitz 1988]. This paramount work lies at the basis of ASP and it is the most widely used definition of the stable model semantics. Soon after, these semantics were extended to also cover classical negation[10] and disjunction. The latter increases the expressiveness and the complexity of the stable model semantics, making it possible to model and solve a larger class of problems. It was realised in [Lifschitz 1999, Marek 1999] that ASP, i.e. logic programming based on the stable model semantics, is a programming paradigm well-suited for solving hard combinatorial problems. Such problems are modelled using ASP programs and particular solutions of such programs, i.e. the answer sets, then correspond with solutions to the original problem.

To illustrate the ASP paradigm, we look at the problem of map colouring. This problem is concerned with determining, for a given map, a colour for each region such that no two adjacent regions share the same colour. To solve this problem with ASP, we divide it into two parts. First, we need to colour each region. Second, we need to verify whether our choice is a good map colouring, i.e. whether no adjacent regions share the same colour.

[10]Classical negation differs from negation-as-failure since classical negation indicates that the proof that the statement is false can explicitly be derived.

This problem can be solved with the following rules:

$$\begin{aligned}
colour(X, red) &\leftarrow region(X), not\ colour(X, blue), not\ colour(X, green), not\ colour(X, yellow)\\
colour(X, blue) &\leftarrow region(X), not\ colour(X, red), not\ colour(X, green), not\ colour(X, yellow)\\
colour(X, green) &\leftarrow region(X), not\ colour(X, red), not\ colour(X, blue), not\ colour(X, yellow)\\
colour(X, yellow) &\leftarrow region(X), not\ colour(X, red), not\ colour(X, blue), not\ colour(X, green)\\
&\leftarrow adjacent(X, Y), colour(X, C), colour(Y, C)
\end{aligned}$$

Intuitively, the first 4 rules are used to assign a colour to each region. The first rule can be read as "*colour the region X red unless you already coloured it blue, green or yellow*". The other three rules can be similarly understood. The last rule states that when X and Y are adjacent they cannot have the same colour C.

Thus far, we described the general problem, but we did not specify a problem instance. The problem instance can also be specified in the form of rules, as above. These rules are somewhat different in that they are unconditionally true, i.e. the rules describe facts. We consider the following problem instance, which represents the Belgian provinces and Brussels:

$$\begin{array}{rrr}
region(we) \leftarrow & region(ea) \leftarrow & region(an) \leftarrow\\
region(lm) \leftarrow & region(fl) \leftarrow & region(wa) \leftarrow\\
region(br) \leftarrow & region(ha) \leftarrow & region(na) \leftarrow\\
region(li) \leftarrow & region(lu) \leftarrow & \\
adjacent(we, ea) \leftarrow & adjacent(we, ha) \leftarrow & adjacent(ea, an) \leftarrow\\
adjacent(ea, fl) \leftarrow & adjacent(ea, ha) \leftarrow & adjacent(an, fl) \leftarrow\\
adjacent(an, lm) \leftarrow & adjacent(lm, fl) \leftarrow & adjacent(lm, li) \leftarrow\\
adjacent(li, fl) \leftarrow & adjacent(li, wa) \leftarrow & adjacent(li, na) \leftarrow\\
adjacent(li, lu) \leftarrow & adjacent(lu, na) \leftarrow & adjacent(na, wa) \leftarrow\\
adjacent(na, ha) \leftarrow & adjacent(wa, fl) \leftarrow & adjacent(fl, br) \leftarrow\\
adjacent(ha, fl) \leftarrow & adjacent(ha, wa) \leftarrow &
\end{array}$$

Together these rules form the ASP program P_{map}. When an ASP program is solved, it may have zero, one or many answer sets. A program with zero answer sets represents a problem without a solution. When there is one answer set, then this corresponds with the unique solution to the problem. Otherwise, the problem has a number of answers and the user can either reason over all the answer sets (e.g. "*must Brussels and Antwerp always have the same colour?*") or ask for one or more answer sets. The ASP program P_{map}

has multiple answer sets, including the following:

$$\begin{aligned} M \;\cup\; \{ & colour(we, blue), colour(ea, green), colour(an, red), colour(li, blue),\\ & colour(ha, red), colour(br, red), colour(fl, yellow), colour(wa, green),\\ & colour(na, blue), colour(li, red), colour(lu, green)\} \end{aligned}$$

where M contains information w.r.t. the facts, i.e. the problem instance. The corresponding colouring is shown in Figure 1.1.

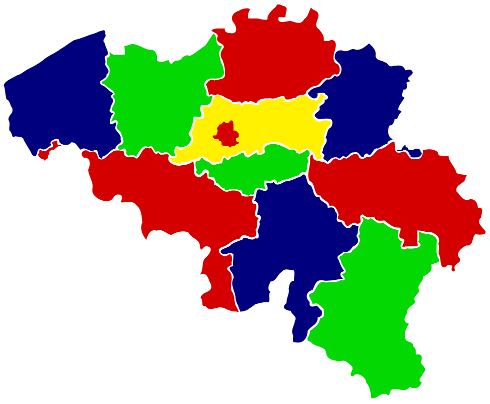


Figure 1.1: A map colouring corresponding with an answer set.

Other semantics have been devised to deal with expressions of the form '*not a*'. Specifically, in [Van Gelder et al. 1988] the well-founded semantics are introduced. These semantics are three-valued and assign to each atom the value true, false or unknown. It was realised, however, in [Baral and Subrahmanian 1993] that the well-founded semantics agree with the stable model semantics. That is, atoms that were found to be true or false remained true or false, respectively, under the stable model semantics. The well-founded semantics, however, have the benefit that they are easier to compute and can be used as a preliminary step in the computation of the answer sets of a program. This in turn led to solvers for ASP [Simons 1999, Niemelä and Simons 2000, Baral 2003], which can be used to efficiently compute the answer sets of ASP programs such as P_{map}. Over the last two decades, many others definitions of the stable model semantics have been formulated where each new definition has contributed new insights [Lifschitz 2008]. In Chapter 4 of this thesis we present another such definition, where we show how the semantics of ASP can be expressed as constraints on possibility distributions, which we will discuss in more detail in Section 1.3 and 2.3. Such a characterization allows us to define the semantics of Possibilistic Answer Set Programming (PASP), an extension of ASP to allow us to reason about uncertain information, in a natural way. Furthermore, as we will see in Section 4.3.2, using this characterization we can unearth a new form of disjunction in ASP with interesting complexity results.

While ASP is an excellent tool to model e.g. planning problems, it is not an ideal vehicle for modelling multi-agent problems or problems with uncertainty. However, such problems are widespread in common-sense reasoning. For example, we may be uncertain whether or not a booked flight will leave on time. Information of this form cannot easily be modelled in ASP, since information in ASP is either true or false and we have no means of expressing uncertainty w.r.t. such statements. Still, humans are able to reason with such information and would be able to conclude that e.g. they still need to go to the airport. The ability to communicate and exchange information is also an important part of common-sense reasoning. Let us consider the following example, which we present in more detail in Chapter 3. Two agents look at a box from different angles. The box itself has a 2×3 floor plan and the ball, which both agents try to find, is located on one square of this floor plan. To complicates matters, some sections of the box are out of sight. From the point of view of $Mr.1$ the box is divided into two parts – of which one is blocked – and he cannot see a ball in the unblocked part. The other agent, $Mr.2$, can see that the box is divided into three parts – two of which are blocked – but this agent can see a ball in the unblocked part. This is depicted in Figure 1.2. By cooperating, which requires communication, the agents can work together to determine the exact location of the ball. Once again, such information is hard to encode in ASP since ASP does not provide means to e.g. allow for communication between different ASP programs. In the next two sections, we look at a number of ideas which will allow us to devise such epistemic extensions of ASP.

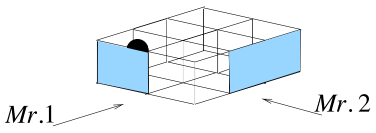


Figure 1.2: A magic box.

1.2 Epistemic Reasoning

In the previous section we discussed how the stable model semantics can be defined by equating an expression of the form '$not\ a$' with '$\neg \mathrm{L}a$', i.e. "*it is not believed that*

'a' [is true]". We thus express the semantics of ASP in terms of what is believed or known. Reasoning about what is known or believed is an interesting subdomain of logic, called epistemic logic or the Theory of Knowledge. The focus in this subdomain is on the nature of knowledge and how it relates to concepts such as justification, truth and belief. Epistemic logic forms a subset of modal logic [Chellas 1980] and was already studied by Aristotle, although the logic proposed by Aristotle at the time exhibited many flaws [Uckelman and Johnston 2010]. The work in [von Wright 1951] can be seen as the start of the formal study of epistemic logic as it is known today.

Epistemic logic can be used to provide useful insights of an individual agent. By introducing a modal operator K, i.e. "*it is known that*", it becomes possible to reason about what this agent knows. For example, we are able to differentiate between '$winner$' and 'K$winner$', i.e. we can differentiate between *'the agent is the $winner$ of the jackpot'* and *'the agent knows that he is the $winner$ of the jackpot'*. Clearly, these two things can be very different (just imagine the difference in excitement). Epistemic logic also helps to provide interesting insights when we are dealing with a group of agents. Indeed, we can not only reason about what each individual agent knows, but we can furthermore reason about what is known by the group as a whole. Still, even though ASP can be defined in terms of an epistemic logic, classical ASP does not allow for multiple ASP programs, or agents, to communicate and cooperate. Such extensions have been proposed in the literature, but it remains unclear what exactly contributes to an increase in the power of such ASP programs. This is a problem which we discuss and clarify in Chapter 3 by showing that the choice of the communication mechanism is paramount in terms of the expressive power of the resulting system.

To define the semantics of epistemic logic, we can represent the information in epistemic logic by means of possible worlds, one of which is the actual world. Each such world is compatible with the beliefs of the agent, i.e. it corresponds with a possible scenario of the actual world according to the agent. Worlds indiscernible by the agent are connected by an accessibility relation, represented by arrows. An example is given in Figure 1.3. In

$$\begin{tikzcd}
\boxed{rain, sunday} \arrow[r, bend left] & \boxed{\neg rain, sunday} \arrow[l, bend left]
\end{tikzcd}$$

Figure 1.3: Representation of an epistemic state.

this example, the agent is unable to differentiate between a world in which it rains and one in which it is not raining. At the same time, only worlds in which Sunday is true are consistent with his beliefs. As such, the agent knows that it is Sunday, i.e. K$sunday$, whereas the agent does not know whether it is raining, i.e. $\neg$K$rain$. Still, as indicated, one of these worlds is the actual world. As such we may e.g. have that $rain$ is true.

Autoepistemic logic [Moore 1985], which we already briefly discussed, is another logic that can be used for epistemic reasoning. In autoepistemic logic a modal operator L, which is read as "*it is believed that*", is considered instead of a modal operator K. As such, autoepistemic logic can be used to model an agent reflecting upon his own beliefs. The semantics were originally defined in terms of a fixpoint operator, but can also be defined in terms of possible worlds semantics [Moore 1984]. Specifically, in autoepistemic logic, the stable autoepistemic theories can be identified as the sets of formulas that are true in a complete possible world structure[11], i.e. a structure in which every world is accessible from every world [Moore 1984].

The stable model semantics, the semantics that underpin ASP, also have a strong epistemic foundation, as shown in [Loyer and Straccia 2006]. Intuitively, we already discussed how the original definition of the stable model semantics equates an expression of the form '$not\ a$' with '$\neg \mathrm{L}a$', i.e. "*it is not believed that 'a' [is true]*". In a sense, when looking at an ASP program from an epistemic point of view, negation-as-failure can thus be seen as a form of epistemic uncertainty, i.e. uncertainty caused by the practical inability to determine the truth of some statement. This is the case in Figure 1.3, where the agent is unable to determine whether it is raining. This inability may have many causes. It may be too costly for the agent to find out whether it is raining (e.g. the agent may be underground and unwilling/incapable to check wether it is raining) or the agent may simply be ignorant as to whether or not it is raining.

However, uncertainty can be more general than merely the absence of information. Indeed, we may have a degree of certainty towards a statement. For example, in Figure 1.3, the agent may have overheard a discussion where he thought they were saying that it is raining. As such, the agent may prefer his belief that it is raining over his belief that it is not raining. Or, to put it differently, the agent will be more surprised when he would discover that is not raining. This specific type of uncertainty has a qualitative characteristic, as it only allows us to rank-order the different worlds. In the next section, we describe a theory specifically developed to reason about such (qualitative) uncertainty.

1.3 Possibility Theory

Aristotle already realized that being able to deal with uncertainty is an essential component of common-sense reasoning. Aristotle was in particular interested in determining what is possible and necessary[12]. Still, at the time, the concepts of possibility and necessity were strongly rooted in modal logic and described binary concepts, i.e. statements that

[11]Specifically, we need to consider an S5 structure, i.e. a structure corresponding with the S5 modal logic: http://en.wikipedia.org/wiki/S5_(modal_logic).

[12]Source: http://plato.stanford.edu/entries/aristotle-logic/, retrieved March 16, 2013.

are either true or false. In the 1960's, Shackle presented work on degrees of potential surprise [Shackle 1961], an idea closely related to possibility theory, that allows for dealing with incomplete information. In this view, potential surprise is understood as disbelief. For example, when betting on horses we may know that one of the two horses, horse B, has an injury. If horse B still won, we would be surprised, i.e. we believe it to be more plausible that horse A will win in this race.

Later, in the 1970's, the philosopher David Lewis introduced the notion of a graded possibility in [Lewis 1973]. This notion, called comparative possibility, was defined as a form of weak ordering between the possible worlds. He furthermore extended upon this idea by showing that the notion of similarity between possible worlds and the notion of a most plausible world can be defined in terms of comparative possibility.

The first work on possibility theory, where it is named as such, is [Zadeh 1978]. In this work, Zadeh interprets membership functions of fuzzy sets as possibility distributions that act as flexible constraints on the values that can be assigned to a variable. For example, the membership function μ_{tall}, which describes the word '*tall*', induces a possibility distribution Π where Π denotes how compatible the value x is with $tall$. As such, we would e.g. find that $\Pi(1.90m) \approx 1$ as it is entirely possible that someone who is considered to be tall has an actual height of 1.90m. Similarly, $\Pi(1.40m) \approx 0$ as we would be very surprised if someone identifies a person of only 1.40m as tall. This idea immediately links back to the notion of potential surprise from [Shackle 1961]. Indeed, we would be more surprised to find a person described as tall to have an actual height of 1.40m, which we consider almost impossible, than that we would be if the person has a height of 1.90m, which we consider entirely possible.

The work from Zadeh is important for a number of reasons. First, it highlights the close link between possibility theory and fuzzy sets, where fuzzy sets combine both uncertainty (which can be modelled by possibility theory) with gradualness (or multi-valuedness). This highlights a principal distinction between possibility theory and multi-valued theories. Indeed, stating that a bottle is half full (denoted as $bottle\_full = 0.5$) is different from stating that it is somewhat possible that the bottle is full (denoted as $\Pi(bottle\_full) = 0.5$). Specifically, in the latter case we are expressing our uncertainty about whether or not the Boolean statement $bottle\_full$ is true or not. In the first case we no longer consider a Boolean statement, but rather assume that the bottle is in a state between empty and full. In addition, the difference between possibility theory and most multi-valued logics is not merely conceptual. Indeed, possibility theory is e.g. not truth-functional. As such, the truth of a statement in possibility theory is not necessarily a function of the truth of its constituents. In fact, possibility theory would become trivial when it is forced to be truth-functional [Dubois and Prade 2001].

Furthermore, the work from Zadeh shows important relations, and differences, between possibility and probability theory (see e.g. [Loève 1977]). Both theories are used to deal with uncertainty and Zadeh showed how possibility theory can be used as an approximation of probability theory. However, possibility theory differs from probability theory by the use of a pair of dual set-functions (namely possibility and necessity measures) instead of only one function (probability-measure). Probability theory, which has a single set-function and in which uncertainty is additive, is a good model of randomness and indecisiveness. Possibility theory only exploits the fact that the unit interval is a total ordering. Together with the use of a pair of dual set-functions, this makes possibility theory a good model of partial ignorance [Dubois et al. 1993] and epistemic uncertainty. As such, both possibility and probability theory capture a different facet of uncertainty and both theories complement each other.

In the last two decades Dubois and Prade considerably elaborated on the work from Zadeh w.r.t. possibility theory and developed it into an effective framework for reasoning about uncertainty. Aside from many interesting technical results, they also proposed possibilistic logic in [Dubois et al. 1994]. Possibilistic logic combines possibility theory with propositional logic to obtain a logic capable of reasoning under uncertainty. As we will see in this thesis, this forms the basis of PASP, which is an extension of ASP first proposed in [Nicolas et al. 2006] that allows us to combine non-monotonic reasoning, declarative programming and reasoning about uncertainty in a single framework. However, we will discuss in Chapter 4 that the existing semantics for PASP, do not always correspond with the intuition of the problem. This is due to their particular treatment of negation-as-failure. As such, we show in Chapter 4 how alternative semantics for PASP can be defined that adhere to a different intuition of negation-as-failure. These new semantics can be used when the existing semantics for PASP offer unintuitive results. In addition, these new semantics will allow us to provide a new characterization for ASP which will allow us to unearth a new form of disjunction. In Chapter 5 we furthermore show that the uncertainty in PASP can be interpreted in a number of ways. This will allow us in Section 5.3 to use PASP to solve a number of important problems in Artificial Intelligence.

1.4 Thesis Outline

Answer Set Programming (ASP) is capable of modelling and solving hard combinatorial problems. However, the lack of abilities in ASP to model knowledge of a network of agents, who can cooperate, and the inability to deal with uncertainty hamper its practical ability to be used for common-sense reasoning.

The aim of this thesis is to look at epistemic extensions of ASP, i.e. we are interested in extensions of ASP that allow us to reason about the knowledge described in a given context. For starters, we are interested in ways of extending ASP such that we can conceptually reason about a group of contexts. To this end, we propose CASP in Chapter 3, which is a declarative domain-specific programming language that allows for a network of ASP programs to share knowledge and collaborate towards finding a solution to complex problems. We show how the simple mechanism of asking questions is sufficient to simulate negation-as-failure. We furthermore show that the addition of focussing, another simple mechanism used to define a linear order among the agents, considerably affects the complexity and expressiveness of the resulting framework.

Furthermore, in this thesis we want to look at extensions of ASP that allow us to reason about uncertainty. Specifically, we will present new semantics for PASP in Chapter 4 and 5. PASP combines ASP with Possibilistic Logic to provide a declarative framework for non-monotonic reasoning under uncertainty. Syntactically, a weight is associated with a classic ASP rule, where the weight denotes the maximum certainty with which we can derive the conclusion. The existing semantics of PASP, however, have a number of issues which we point out in Chapter 4. To uncover alternative semantics for PASP, we show how ASP can be characterized in terms of constraints on possibility distributions. This new characterization of the stable model semantics can then trivially be extended to cover PASP, which overcomes some of the issues of the existing semantics for PASP. In addition, this new characterization reveals a new interpretation of disjunction in ASP called weak disjunction. This new form of disjunction is closer to an epistemic understanding of ASP and turns out to be non-trivial. Indeed, this new form of disjunction is more complex and expressive than normal programs, although less so than strong disjunction, i.e. the form of disjunction widely used in ASP.

While the new semantics for PASP solve a number of issues, they are not the only way in which the weights attached to the rules can be interpreted. In particular, both the semantics which we introduce in Chapter 4 and the existing semantics of PASP associate the weight, which is attached to a classic ASP rule, with the certainty of the conclusion. It is also possible to see this weight as the certainty with which the rule itself is true, i.e. the certainty that the information encoded in the rule is valid. These two views, uncertain conclusions versus uncertain rules, are contrasted in Chapter 5. We find that the new view of uncertain rules results in new decision problems and we thoroughly discuss their complexity.

In Chapter 6 we look back at CASP and PASP and we discuss how these newly introduced frameworks can be efficiently translated to classical ASP. As such, we demonstrate that it is possible to use the highly optimised solvers available for ASP to efficiently compute both communicating and possibilistic answer sets.

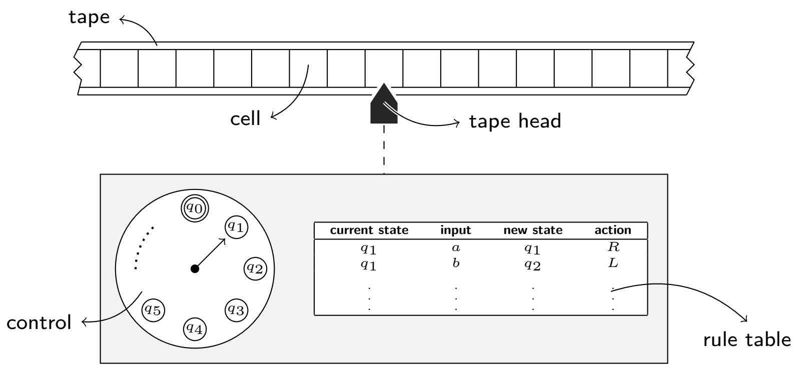
The results in this thesis have been published, or submitted for publication, in international journals and the proceedings of international conferences with peer review. Specifically, CASP was first introduced in [Bauters et al. 2010a], extended with focussing in [Bauters et al. 2011a] and studied in more detail in [Bauters et al. 2013a]. The applicability of CASP for modelling negotiations was briefly discussed in [Bauters 2011]. New semantics for PASP and the characterization of ASP in terms of constraints on possibility distributions were proposed in [Bauters et al. 2010b]. In [Bauters et al. 2011b] it was shown that this characterization naturally gave rise to a new form of disjunction in ASP. These results were bundled and studied in further detail in [Bauters et al. 2012a]. The alternative interpretation of weights in ASP was first introduced in [Bauters et al. 2012b] and further discussed and contrasted with the earlier views in [Bauters et al. 2013b].

2 | Preliminaries

In this chapter we introduce some preliminary notions from complexity theory, answer set programming, possibilistic logic and possibilistic answer set programming.

2.1 Complexity Theory

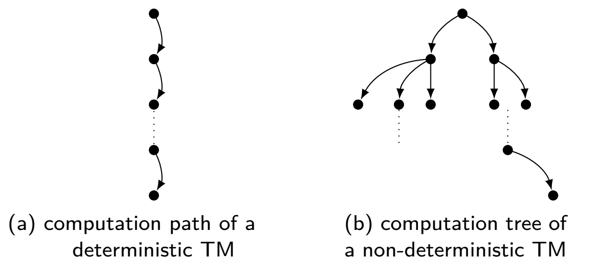
We start by recalling some notions from complexity theory. A *Turing machine* [Turing 1936, Papadimitriou 1994] is a hypothetical machine that forms the cornerstone of complexity theory. A representation of a possible Turing machine is given in Figure 2.1.



tape
cell
tape head
control
rule table
q_0 q_1 q_2 q_3 q_4 q_5
current state | input | new state | action
q_1 | a | q_1 | R
q_1 | b | q_2 | L

Figure 2.1: A graphical representation of a Turing machine.

A Turing machine consists of a *finite set of states* Q (i.e. the control), an *infinite tape* divided in *cell*s each containing a symbol from the alphabet Γ (i.e. the memory) and a *tape head* used to read/write from/to the tape. The tape head is always positioned over exactly one cell. Initially, the Turing machine is in the *initial state* $q_0 \in Q$ and the tape is filled with blank symbols $B \in \Gamma$ except for a contiguous finite sequence of cells (i.e. the input). The tape head is positioned above the first cell of this sequence. At each time step, depending on the current state $q \in Q$ and the symbol $\gamma \in \Gamma$ read from the tape, the Turing machine can transition to another state $q' \in Q$ and perform an action with the tape head. As an *action*, the tape head can write a symbol from the alphabet Γ to the tape or it can be moved either to the cell on the left (L) or the right (R) of the current cell. The way that the Turing machine behaves is described by a set of rules, i.e. the *rule table*. Each rule has the form (q, s, q', a) with $q, q' \in Q$, $s \in \Gamma$ and a an action, i.e. either a symbol $s' \in \Gamma$ or L or R. Some of the states $q \in Q$ are special. In particular, we consider a set $Q_A \subseteq Q$ of accepting states and a set $Q_R \subseteq Q$ of rejecting states. A Turing machine *halts* immediately when it enters either an accepting or a rejecting state. Consequently, we say that the input is accepted or rejected when the Turing machine has entered an accepting or rejecting state, respectively. The Turing machine may also enter a loop, where the machine runs indefinitely and never enters a state in which the machine halts. A Turing machine is *deterministic* when for every pair q, s there is exactly one rule (q, s, q', a), i.e. given the current state q and the symbol s read from the tape, there is exactly one corresponding new state q' and action a. A Turing machine is *non-deterministic* when for every pair q, s there may be zero or more rules of the form (q, s, q', a). Whenever there is no deterministic choice, the Turing machine branches into many copies where each branch follows one of the possible transitions. Thus, rather than a single computation path, a non-deterministic Turing machine has a computation tree. If at least one of these branches halts in an accepting state, we say that the input is accepted. This branching behaviour is illustrated in Figure 2.2.


(a) computation path of a deterministic TM

(b) computation tree of a non-deterministic TM

Figure 2.2: Deterministic versus non-deterministic TM.

A Turing machine as described above is a theoretical model of computability that can easily be understood. Still, it is powerful enough to encompass everything that is computable. Specifically, the Church-Turing thesis states that a function is algorithmically computable if and only if it is computable by a Turing machine.

Example 1

Consider the Turing machine M such that $Q = \{go, end\}$, $Q_A = \{end\}$, $Q_R = \{\}$, $\Gamma = \{0, 1, B\}$ and $q_0 = go$. Furthermore, the rule table consists of the following three rules:

$$(go, 0, go, 1) \qquad (go, 1, go, R) \qquad (go, B, end, B)$$

When the Turing machine reads a 0 on the tape, it writes a 1. When it reads a 1, it moves the tape head to the right. When it reaches the end of the input (i.e. it reads a blank B symbol), it goes into an accepting state. As such, we have successfully defined a program that will overwrite the input with 1's.

In complexity theory we are often interested in languages, where a *language* over an alphabet Σ is a subset of Σ^*, i.e. the set of all strings over symbols in Σ including the empty string. A Turing machine is said to accept a language L if and only if the Turing machine accepts the input when the input is a member of L and rejects the input otherwise. We are particularly interested in *decider*s, i.e. Turing machines that always either accept or reject given an input. Problems that can be solved by deciders are also often referred to as *decision problem*s, i.e. those problems for which the solution is either yes or no.

The complexity class P is defined as the set of languages accepted on a deterministic Turing machine in time $O(n^c)$, with n the input length and c a natural number, i.e. the Turing machine makes at most $O(n^c)$ steps before reaching either an accept or reject state. Equivalently, the complexity class P can thus be defined as the set of decision problems that can be solved in polynomial time on a deterministic Turing machine [Papadimitriou 1994]. The complexity class NP is defined as the class of decision problems that can be solved in polynomial time on a non-deterministic Turing machine, i.e.

- if the answer is *yes*, at least one computational path exists that accepts the input;
- if the answer is *no*, all the computational paths reject the input.

An equivalent definition of the complexity class NP is that it is the set of decision problems for which the proof that the answer is *yes* can be verified in polynomial time by a deterministic Turing machine, i.e. in P [Papadimitriou 1994].

These two complexity classes give rise to a whole range of new complexity classes. The complexity classes Σ_k^{P} and Π_k^{P} are defined as follows, for $i \in \mathbb{N}$ [Papadimitriou 1994]:

$$\begin{aligned}\Sigma_0^{\mathsf{P}} &= \Pi_0^{\mathsf{P}} = \mathsf{P}\\ \Sigma_{i+1}^{\mathsf{P}} &= \mathsf{NP}^{\Sigma_i^{\mathsf{P}}}\\ \Pi_{i+1}^{\mathsf{P}} &= \mathsf{co}\left(\Sigma_{i+1}^{\mathsf{P}}\right)\end{aligned}$$

where $\mathsf{NP}^{\Sigma_i^{\mathsf{P}}}$ is the class of decision problems that can be solved in polynomial time on a non-deterministic Turing machine with an Σ_i^{P} oracle, i.e. assuming a procedure that the Turing machine can call to solve Σ_i^{P} decision problems in constant time. We have that $\mathsf{co}\left(\Sigma_{i+1}^{\mathsf{P}}\right)$ is the class of problems whose complement is a decision problem in $\Sigma_{i+1}^{\mathsf{P}}$, i.e. the problem where we reverse the yes and no answer. We also consider the complexity class BH_2 [Cai et al. 1988], which is the class of all languages L such that $L = L_1 \cap L_2$, where L_1 is in NP and L_2 is in coNP. For a general complexity class C, a problem is C-*hard* if any problem in C can be efficiently reduced to this problem. In particular, this means that we have a log-space reduction to this problem when the problem is in P or a polynomial time reduction to this problem when the problem is in some complexity class other than P. A problem is said to be C-*complete* if the problem is in C and the problem is C-hard.

Before we define some well-known problems in the complexity classes that we just defined, we need to introduce some additional terminology. An expression such as x_1 is called an *atom*. Expressions such as x_1 and $\neg x_2$ are called *literals*, i.e. an atom or an atom preceded by classical negation. A *clause* is a disjunction of literals, e.g. $(x_1 \vee \neg x_2 \vee \neg x_3)$. An expression is said to be in conjunctive normal form (CNF) when it is a conjunction of clauses, e.g. $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_4 \vee x_5)$. Similarly, an expression is said to be in disjunctive normal form (DNF) when it is a disjunction of terms where each term is a conjunction of literals.

The boolean satisfiability problem (SAT) is the decision problem of determining for some boolean expression ϕ whether an assignment of true or false to the variables exists that makes the expression true in the usual sense. This problem is NP-complete [Cook 1971].

Example 2

Consider the expression $\phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2)$ where $\vee$, $\wedge$ and $\vee$ denote classical disjunction, conjunction and negation, respectively. An assignment that makes ϕ true is, e.g. where x_1 is true and x_2 is false and the value of x_3 can either be true or false. The expression $\psi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge \neg x_1 \wedge x_2 \wedge x_3$ on the other hand is not satisfiable, i.e. there does not exist an assignment of truth values to make the expression ψ true.

The unSAT problem is the complementary problem of SAT. Thus this is the problem of verifying, given a boolean expression ϕ, whether ϕ has no assignment that makes the expression true. This problem is coNP-complete [Cook 1971]. SAT-unSAT is the canonical BH_2-complete problem and consists of determining for some pair (T, S) of propositional theories in conjunctive normal form (CNF) whether T is satisfiable and S is unsatisfiable.

A generalization of SAT is the Quantified Boolean Formula (QBF) problem. In this generalization, both existential and universal quantifiers can be applied to each variable. Let $p(X_1, X_2, ..., X_n)$ be a propositional formula defined over the variables $X_1, X_2, ..., X_n$. Deciding the validity of a QBF $\phi = \exists X_1 \forall X_2 \ldots \Theta X_n \cdot p(X_1, X_2, ..., X_n)$ with X_i, $1 \leq i \leq n$, sets of variables, $\Theta = \exists$ if n is odd and $\Theta = \forall$ otherwise, is the canonical Σ_n^{P}-complete problem. Deciding the validity of a QBF $\phi = \forall X_1 \exists X_2 ... \Theta X_n \cdot p(X_1, X_2, ..., X_n)$ with $\Theta = \forall$ if n is odd and $\Theta = \exists$ otherwise, is the canonical Π_n^{P}-complete problem. Moreover, these results also hold when we restrict ourselves to problems with $p(X_1, X_2, ..., X_n)$ in disjunctive normal form (DNF), except when the last quantifier is an $\exists$.[1]

Example 3

The QBF $\exists x_1 \forall x_2, x_3 \cdot \phi$ with ϕ as defined in Example 2 is not satisfiable. Indeed, if x_1 is true then ϕ is not satisfiable when x_2 is true. If x_1 is false, then ϕ is not satisfiable when x_2 is also false. However, the QBF $\exists x_1, x_3 \forall x_2 \cdot \phi$ is satisfiable since, when x_1 is false and x_3 is true, ϕ is true regardless of the assignment of x_2.

2.2 Answer Set Programming

We now introduce Answer Set Programming (ASP) [Gelfond and Lifschitz 1988]. In this subsection, we first define the syntax and the semantics of ASP. Then, we discuss the complexity of various classes of ASP. Finally, we discuss how an ASP program can be simulated using a set of clauses, which will be used in Chapter 6 to simulate normal programs.

[1]Given a QBF with the last quantifier an $\exists$ and a formula in disjunctive normal form, we can reduce the problem in polynomial time to a new QBF without the last quantifier. To do this, for every variable quantified by this last quantifier we remove those clauses in which both the quantified variable and its negation occur (contradiction) and then remove all occurrences of the quantified variables in the remaining clauses as well as the quantifier itself. The new QBF is then valid if and only if the original QBF is valid.

2.2.1 Syntax

To define ASP programs, we start from a finite set of atoms $\mathcal{A}$. A *literal* is defined as an atom 'a' or its classical negation '$\neg a$'. For a set of literals L, we use $\neg L$ to denote the set $\{\neg l \mid l \in L\}$ where, by definition, $\neg\neg a = a$. A set of literals is said to be *consistent* when $L \cap \neg L = \emptyset$, i.e. L does not contain two contradictory literals. The set of all literals is written as $\mathcal{L} = \mathcal{A} \cup \neg\mathcal{A}$. A *naf-literal* is either a literal 'l' or an expression of the form '$not\ l$', where 'not' denotes negation-as-failure. Intuitively, we have that '$not\ l$' is true when we have no proof for 'l'.

A *disjunctive rule* is an expression of the form

$$l_0; ...; l_k \leftarrow l_{k+1}, ..., l_m, not\ l_{m+1}, ..., not\ l_n$$

where l_i is a literal for every $0 \leq i \leq n$. We say that $l_0; ...; l_k$ is the *head* of the rule (interpreted as a disjunction) and that $l_{k+1}, ..., l_m, not\ l_{m+1}, ..., not\ l_n$ is the *body* of the rule (interpreted as a conjunction). For a given rule r we use $head(r)$ and $body(r)$ to denote the set of naf-literals in the head and the body of the rule, respectively. Specifically, we use $body_+(r)$ to denote the set of literals in the body that are not preceded by the negation-as-failure operator 'not' and $body_-(r)$ for those literals that are preceded by 'not'.

Example 4

Consider the rule

$$r = (bbq; beach \leftarrow sunny, \neg work, not\ rain).$$

We have that $bbq; beach$ is the head of the rule, while $sunny, \neg work, not\ rain$ is the body of the rule. Furthermore, we have that $head(r) = \{bbq, beach\}$ and $body(r) = \{sunny, \neg work, not\ rain\}$. More precisely, we have that $body_+(r) = \{sunny, \neg work\}$ and $body_-(r) = \{rain\}$.

Specific types of rules can be identified based on their syntactic properties. In particular, we distinguish the following types.

- When a rule r has an empty body, i.e. $body(r) = \emptyset$, we say that the rule is a *fact rule*. This is a shorthand notation for $(l_0; ...; l_k \leftarrow \top)$ with $\top$ a special language construct denoting tautology.
- When a rule r has an empty head, i.e. $head(r) = \emptyset$, we say that the rule is a *constraint rule*. This is a shorthand notation for $(\bot \leftarrow l_{k+1}, ..., l_m, not\ l_{m+1}, ..., not\ l_n)$ with $\bot$ another special language construct denoting contradiction.

- When a disjunctive rule r does not contain negation-as-failure, i.e. when $m = n$, we say that the rule is a *positive disjunctive rule*.
- When a disjunctive rule r has at most one literal in the head, i.e. $|\,head(r)\,| \leq 1$, we say that the rule is a *normal rule*.
- When a normal rule r does not contain negation-as-failure, i.e. $body_{-}(r) = \emptyset$, we say that the rule is a *simple rule*.
- When a simple rule r does not contain classical negation, i.e. all the literals are atoms, we say that the rule is a *definite rule*.

Definition 1

A *disjunctive (resp. positive disjunctive, normal, simple, definite) program* is a finite set of disjunctive (resp. positive disjunctive, normal, simple, definite) rules.

Example 5

$$\begin{aligned}
sunny &\leftarrow \\
day\_off &\leftarrow \\
bbq; beach &\leftarrow sunny, \neg work, not\ rain \\
\neg work &\leftarrow day\_off \\
&\leftarrow broke
\end{aligned}$$

The first two rules are examples of fact rules. The third rule is a disjunctive rule. It is not a positive disjunctive rule, since it uses negation-as-failure in its body. The penultimate rule is a simple rule. It is not a definite rule, since it uses classical negation in the head. The last rule is an example of a constraint rule. This set of rules form a disjunctive program. It is not a positive disjunctive program, however, due to the third rule in the program.

2.2.2 Semantics

Thus far we discussed how the language of ASP is constructed. We now attach a meaning to an ASP program. Intuitively, we want the semantics of ASP to correspond with the conclusions that can be derived from the knowledge that is expressed using an ASP program.

We start by defining the *Herbrand base* $\mathcal{B}_P$ of a disjunctive program P, which is the set of atoms appearing in P. The set of literals relevant for a disjunctive ASP program is defined as $Lit_P = (\mathcal{B}_P \cup \neg\mathcal{B}_P)$. An *interpretation* I of a disjunctive program P is any set of literals $I \subseteq Lit_P$. A *consistent interpretation* is an interpretation I that does not contain both 'a' and '$\neg a$' for some $a \in I$.

Definition 2

A consistent interpretation I is a *model* of a positive disjunctive rule r if $head(r) \cap I \neq \emptyset$ or $body(r) \not\subseteq I$, i.e. the body is false or the head is true.

In particular, a consistent interpretation I is a *model* of a constraint rule r if $body(r) \not\subseteq I$. If for an interpretation I and a constraint rule r we have that $body(r) \subseteq I$, then we say that the interpretation I *violates* the constraint rule r. Notice that for a fact rule we require that $head(r) \cap I \neq \emptyset$, i.e. at least one of the literals in the head must be true. Indeed, otherwise I would not be a model of r. The body of the rule can thus be seen as the preconditions under which the head of the rule, the conclusions, are true.

Definition 3

An interpretation I of a positive disjunctive program P is a *model* of P either if I is consistent and for every rule $r \in P$ we have that I is a model of r, or if $I = Lit_P$.

It follows from this definition that Lit_P is always a model of P, and that all other models of P (if any) are consistent interpretations, which we will further on also refer to as *consistent models*.

Definition 4

We say that an interpretation I is an *answer set* of the positive disjunctive program P if I is a minimal model of P w.r.t. set inclusion, i.e. there does not exist another model I' of P such that $I' \subset I$.

Example 6

Consider the program P_6 with the rules

$$\begin{aligned} red; green; blue &\leftarrow ball \\ ball &\leftarrow \\ &\leftarrow blue \end{aligned}$$

It is easy to see that this is a positive disjunctive program. We can verify that this program has two minimal models, namely $M_1 = \{ball, red\}$ and $M_2 \{ball, green\}$. Both M_1 and M_2 are answer sets of P_6.

Thus far we only considered positive disjunctive programs, i.e. programs without negation-as-failure. For programs with negation-as-failure, the minimal models need not necessarily correspond with our intuition.

Example 7

Consider the program P_7 with the rules

$$\begin{aligned} sunny &\leftarrow \\ bbq &\leftarrow sunny, not\ rain \end{aligned}$$

Intuitively, this program encodes that it is sunny, and that when it is sunny and there is no indication that it rains we will hold a barbecue. This program has two minimal models. The minimal models $\{sunny, rain\}$ and $\{sunny, bbq\}$ both contain knowledge that was not explicitly present. Indeed, the first minimal model assumes that $rain$ is true, whereas the second minimal model assumes that there is no evidence to support that $rain$ is true.

Intuitively, '$not\ l$' is understood as "*it cannot be proven that l is true*". Given this intuition, we only want the second minimal model from P_7 in Example 7. To this end, the semantics of an ASP program with negation-as-failure are based on the idea of a stable model [Gelfond and Lifschitz 1988].

Definition 5

The *reduct* P^I of a disjunctive program P w.r.t. the interpretation I is defined as the set of rules:

$$\begin{aligned} P^I = \{l_0; ...; l_k \leftarrow l_{k+1}, ..., l_m \mid (\{l_{m+1}, ..., l_n\} \cap I = \emptyset) \\ \text{and } (l_0; ...; l_k \leftarrow l_{k+1}, ..., l_m, not\ l_{m+1}, ..., not\ l_n) \in P\}. \end{aligned}$$

Intuitively, we guess some interpretation I and use this guess to remove negation-as-failure from the original program. We discard those rules r where $body_-(r) \cap I \neq \emptyset$, i.e. those rules that are trivially satisfied by I (since the body can intuitively not be satisfied by I), and we remove the naf-literals of the form '$not\ l$' from the remaining rules.

Definition 6

An interpretation I is said to be an *answer set* of the disjunctive program P when I is an answer set of the positive disjunctive program P^I (hence the notion of stable model).

Whenever a disjunctive program P has *consistent answer sets*, i.e. answer sets that are consistent interpretations, we say that P is a *consistent program*.[2] When P has the answer set Lit_P, then this is the unique [Baral 2003] *inconsistent answer set* and we say that P is an *inconsistent program*.

Example 8

Consider the normal program P_8 with the rules:

$$apple \leftarrow \qquad green \leftarrow not\ blue \qquad blue \leftarrow not\ green \qquad \leftarrow blue$$

We have that $\mathcal{B}_P = \{apple, blue, green\}$. Three consistent interpretations of P are $\{apple, blue, green\}$, $\{apple, green\}$ and $\{apple, blue\}$. The first and last interpretation violate the constraint rule. Furthermore, the first interpretation is not minimal w.r.t. the other two interpretations. Only the second interpretation is a model of P. In particular, for $I = \{apple, green\}$ we have that P^I is the program with the set of rules

$$apple \leftarrow \qquad green \leftarrow \qquad \leftarrow blue$$

for which it is immediately clear that $\{apple, green\} = I$ is the minimal model and thus the answer set of P^I, i.e. I is an answer set of P. Furthermore, we say that P is a consistent program since it has consistent answer sets.

Answer sets of simple programs can also be defined in a more procedural way.

Definition 7

The *immediate consequence operator* T_P is defined for a simple program P w.r.t. an interpretation I as:

$$T_P(I) = \{l_0 \mid (l_0 \leftarrow l_1, ..., l_m) \in P \text{ and } l_1, ..., l_m \subseteq I\}\,.$$

[2]Notice that also the empty set can be a consistent answer set.

It is easy to see that this operator is monotonic and the operator is defined over a complete lattice. Due to [Tarski 1955] we then know that the least fixpoint of T_P w.r.t. set inclusion can be computed by repeatedly applying T_P starting from the empty interpretation $\emptyset$. We denote this least fixpoint by $P^\star$.

Proposition 1: from [Baral 2003]

Let P be a simple program without constraint rules. When the interpretation $P^\star$ is consistent, $P^\star$ is the unique and consistent answer set of P.

When we also want to take programs with constraint rules under consideration, we can use the observation that, in general, a program P can be written as $P = P' \cup C$ with C the set of constraint rules in P. When we allow constraint rules, an interpretation is a (consistent) answer set of $P = P' \cup C$ iff I is a (consistent) answer set of P and I is a model of C, i.e. I does not violate any constraints in C.

2.2.3 Complexity of ASP

When we want to analyse the complexity of ASP, we are not only interested in the answer sets themselves but also in a number of reasoning tasks. In particular, we are interested in whether a consistent answer set exists and whether a given literal is true in some or all answer sets of the program.

Definition 8

Let P be a disjunctive ASP program. We use $\models^{\mathrm{b}}$ and $\models^{\mathrm{c}}$ to denote *brave inference* and *cautious inference*, respectively, in ASP, i.e.

- $P \models^{\mathrm{b}} l$ iff $\exists M \cdot M$ is an answer set of P and $l \in M$;
- $P \models^{\mathrm{c}} l$ iff $\nexists M \cdot M$ is an answer set of P and $l \notin M$.

Depending on the syntactic properties of our ASP program, the computational complexity of these reasoning tasks can vary considerably.

Proposition 2: from [Baral 2003, Eiter and Gottlob 1995a]

Let P be an answer set program. Answer set existence, i.e. determining wether P has a consistent answer set is:

- Σ_2^{P}-complete when P is a disjunctive program;

- NP-complete when P is a positive disjunctive program;
- NP-complete when P is a normal program;
- P-complete when P is a simple program;
- P-complete when P is a definite program.

Proposition 3: from [Baral 2003, Eiter and Gottlob 1995a]

Let P be an answer set program. Let 'l' be a literal. Brave reasoning, i.e. determining whether $P \models^{\mathrm{b}} l$, is:

- Σ_2^{P}-complete when P is a disjunctive program;
- Σ_2^{P}-complete when P is a positive disjunctive program;
- NP-complete when P is a normal program;
- P-complete when P is a simple program;
- P-complete when P is a definite program.

Proposition 4: from [Baral 2003, Eiter and Gottlob 1995a]

Let P be an answer set program. Let 'l' be a literal. Cautious reasoning, i.e. determining whether $P \models^{\mathrm{c}} l$, is:

- Π_2^{P}-complete when P is a disjunctive program;
- coNP-complete when P is a positive disjunctive program;
- coNP-complete when P is a normal program;
- P-complete when P is a simple program;
- P-complete when P is a definite program.

Recalling the canonical problems from Section 2.1, this means that we can use brave reasoning over a disjunctive program to simulate a QBF of the form $\exists X_1 \forall X_2 \cdot p(X_1, X_2)$. However, to simulate such a QBF we need a technique called *saturation*. This technique was first proposed in [Eiter and Gottlob 1995b] and plays an important role in a number of proofs in this thesis.

Proposition 5: from [Baral 2003]

Let $\phi \equiv \exists X_1 \forall X_2 \cdot p(X_1, X_2)$ be a QBF with $p(X_1, X_2)$ an expression in DNF, i.e. $p(X_1, X_2)$ is of the form $\theta_1 \vee ... \vee \theta_n$ with θ_i for $1 \leq i \leq n$ a conjunction of literals constructed from the variables in X_1 and X_2. Let the corresponding disjunctive program P_ϕ be defined as the set of rules:

$$\begin{aligned}
&\{x; x' \leftarrow \mid x \in (X_1 \cup X_2)\} && (2.1)\\
&\cup \{sat \leftarrow \theta'_i \mid 1 \leq i \leq n\} && (2.2)\\
&\cup \{x \leftarrow sat \mid x \in X_2\} &&\\
&\cup \{x' \leftarrow sat \mid x \in X_2\} && (2.3)
\end{aligned}$$

with θ'_i obtained from θ_i by replacing every occurrence of $\neg x$ by x', e.g. when $\theta_i = (x \wedge \neg y \wedge z)$ we have $\theta'_i = (x \wedge y' \wedge z)$. The QBF ϕ is satisfiable if and only if $P_\phi \models^{\mathrm{b}} sat$.

We refer the reader to [Baral 2003] for the proof of Proposition 5. Because the saturation technique used in Proposition 5 plays an important role in a number of proofs in Chapter 4 and 5, we briefly explain the intuition behind saturation below and illustrate it with 2 small examples.

The intuition of the program is that an assignment is guessed in (2.1). The rules in (2.2) verify whether this assignment makes the expression $p(X_1, X_2)$ true. The saturation is applied in the rules (2.3) and makes use of the definition of an answer set as a minimal model to enforce that 'sat' will only be contained in an answer set when the expression $p(X_1, X_2)$ is true for every possible assignment of variables in X_2. Indeed, let M be a model of P_ϕ that contains literals corresponding to the variables of X_1 and X_2, i.e. M defines an assignment of X_1 and X_2, such that the expression $p(X_1, X_2)$ is true. Due to the rules (2.2) we have $sat \in M$ and, due to the rules (2.3), M must contain literals corresponding with all the possible assignments of X_2.

Now suppose that there exists some other model M' of P_ϕ that defines the same assignment of X_1 as M, but another assignment for X_2. In particular, assume that M' defines an assignment of X_2 such that the expression $p(X_1, X_2)$ is false. Then due to the rules (2.2) this means that $sat \notin M'$. However, due to the rules (2.3) we had that M contains literals corresponding with every possible assignment of X_2, i.e. we must have $M' \subset M$. Clearly, in this case M is not a minimal model of P_ϕ and thus not an answer set. Only if we are unable to find an M' that defines an assignment of X_2 such that the expression $p(X_1, X_2)$ is false will we be able to have an answer set M of P_ϕ such that $sat \in M$ and thus $P_\phi \models^{\mathrm{b}} sat$.

Example 9

Consider the QBF $\phi \equiv \exists x_1, x_2 \forall y_1, y_2 \cdot p(x_1, x_2, y_1, y_2)$ with $p(x_1, x_2, y_1, y_2)$ the formula $(x_1 \wedge y_1) \vee (x_2 \wedge y_2)$. Notice how this QBF is not satisfiable. Indeed, if we

take y_1 and y_2 to be false, then $p(x_1, x_2, y_1, y_2)$ is false for every choice of assignment for x_1, x_2.
We have that P_ϕ according to Proposition 5 is the set of rules:

$$\begin{array}{lll} x_1; x_1' \leftarrow & y_1; y_1' \leftarrow & \\ x_2; x_2' \leftarrow & y_2; y_2' \leftarrow & \\ \quad sat \leftarrow x_1, y_1 & \quad y_1 \leftarrow sat & y_1' \leftarrow sat \\ \quad sat \leftarrow x_2, y_2 & \quad y_2 \leftarrow sat & y_2' \leftarrow sat \end{array}$$

Let us consider the interpretations $I_1 = \{x_1, x_2, y_1, y_2\}$ and $I_2 = \{x_1, x_2, y_1', y_2'\}$. The first interpretation is a representation of an assignment where we take all the variables to be true. Specifically, in that case we have that the formula $p(x_1, x_2, y_1, y_2)$ is true. The second interpretation corresponds with an identical assignment of x_1 and x_2, but where we take y_1 and y_2 to be false. For this assignment, as we already discussed, we have that the formula $p(x_1, x_2, y_1, y_2)$ is false.

It is clear that I_1 is a model of the topmost four rules. However, we need to extend I_1 for it to be a model of the bottommost six rules. Indeed, we need to add the set of literals $\{sat, y_1', y_2'\}$ to I_1 to make it a model of P_ϕ. As such, we find that $I_1' = \{x_1, x_2, y_1, y_2, sat, y_1', y_2'\}$ is a model of P_ϕ. We can verify that I_2 is a model of P_ϕ, without the need for adding literals. As such, we have found that I_1' and I_2 are models of P_ϕ, although $I_2 \subset I_1'$. It can in fact be shown that of the models I_1' and I_2 only I_2 is an answer set of P_ϕ. Thus, by using saturation, we prevented I_1' from being a minimal model.

Example 10

Consider the QBF $\phi' \equiv \exists x_1, x_2 \forall y_1, y_2 \cdot p(x_1, x_2, y_1, y_2)$ with $p(x_1, x_2, y_1, y_2)$ the formula $(x_1 \wedge x_2) \vee (y_1 \wedge y_2)$. This is a QBF with the same variables but with a different formula. This QBF is satisfiable. Indeed, when assigning true to x_1 and x_2 the formula $p(x_1, x_2, y_1, y_2)$ is true for every choice of assignment for y_1 and y_2.
We have that $P_{\phi'}$ according to Proposition 5 is the set of rules:

$$\begin{array}{lll} x_1; x_1' \leftarrow & y_1; y_1' \leftarrow & \\ x_2; x_2' \leftarrow & y_2; y_2' \leftarrow & \\ \quad sat \leftarrow x_1, x_2 & \quad y_1 \leftarrow sat & y_1' \leftarrow sat \\ \quad sat \leftarrow y_1, y_2 & \quad y_2 \leftarrow sat & y_2' \leftarrow sat \end{array}$$

We again consider the interpretations $I_1 = \{x_1, x_2, y_1, y_2\}$ and $I_2 = \{x_1, x_2, y'_1, y'_2\}$. As before, we need to extend I_1 with $\{sat, y'_1, y'_2\}$ for it to be a model of $P_{\phi'}$. We can then verify that $I'_1 = \{x_1, x_2, y_1, y_2, sat, y'_1, y'_2\}$ is indeed a model of $P_{\phi'}$. This time around, however, $I_2 = \{x_1, x_2, y'_1, y'_2\}$ is not yet a model of $P_{\phi'}$. Indeed, we need to extend I_2 with the set of atoms $\{sat, y_1, y_2\}$ to obtain $I'_2 = \{x_1, x_2, y'_1, y'_2, sat, y_1, y_2\}$, which is a model of $P_{\phi'}$. Unlike in the previous example, we do not have $I'_2 \subset I'_1$ but instead find that $I'_1 = I'_2$. Furthermore, it can be shown that I'_1 is a minimal model, i.e. an answer set of $P_{\phi'}$. We thus find that sat is true in an answer set of $P_{\phi'}$, i.e. we have verified that the QBF ϕ' is satisfiable.

2.2.4 Translating ASP programs to sets of clauses

Given that normal programs and the boolean satisfiability problem (SAT) share the same complexity, and because efficient solvers for SAT exist, a lot of effort has been made in the literature to translate a normal program into an equivalent SAT problem. For the class of tight normal programs this translation is straightforward. To define this class, we first need to define the *dependency graph* of an ASP program. The dependency graph of an answer set program P is a directed graph with signed edges (either $+$ or $-$) such that vertices of the graph are the literals mentioned in P. There is a *directed positive edge* from l_i to l_0 if there is a rule $r \in P$ such that $head(r) = l_0$ and $l_i \in body_+(r)$. Similarly, there is a *directed negative edge* from l_i to l_0 if there is a rule $r \in P$ such that $head(r) = l_0$ and $l_i \in body_-(r)$. We say that there is a *positive path* from l_1 to l_2 if and only if there is a path in the dependency graph from vertex l_1 to vertex l_2 consisting only of positive edges. A normal program P is said to be *tight* if it does not contain a *positive cycle*, i.e. a positive path starting and ending in a vertex l.

The completion [Clark 1978] $comp(P)$ of a normal program P (without classical negation) is the following set of propositional formulas:

$$\begin{aligned}&\Big\{a \equiv \bigvee body'(r) \mid r \in P \text{ and } head(r) = a\Big\}\\ &\cup \{a \equiv \bot \mid \nexists r \in P \cdot head(r) = a\}\\ &\cup \{\neg body'(r) \mid r \in P \text{ and } head(r) = \emptyset\}\end{aligned}$$

where $body'(r) = \top$ whenever $body(r) = \emptyset$. Otherwise, $body'(r)$ is obtained from $body(r)$ by replacing not with $\neg$.

Example 11

Consider the tight normal program P_{11} with the rules

$$\begin{aligned} wealthy &\leftarrow win\_lottery & work\_hard &\leftarrow \\ wealthy &\leftarrow work\_hard, not\ prison & &\leftarrow prison. \end{aligned}$$

We have that $comp(P)$ is

$$\begin{aligned} wealthy &\equiv (win\_lottery) \vee (work\_hard \wedge \neg prison) \\ work\_hard &\equiv \top \\ win\_lottery &\equiv \bot \\ &\neg prison \end{aligned}$$

The minimal model is $\{wealthy, \neg win\_lottery, work\_hard, \neg prison\}$, which is equivalent to the unique answer set $M = \{wealthy, work\_hard\}$ of P.

We have that M is an answer set of a tight normal program (without classical negation) if and only if it is a classical model of its completion [Fages 1994].

In general, any normal program can be translated to a set of clauses. However, no linear translation is known. Furthermore, such translations cannot be both faithful, i.e. solutions of the translation correspond with those of the original program and vice versa, and modular, i.e. the translation of $P \cup P'$ is the union of the translations of P and P' [Janhunen 2006]. Since we want faithful translations, the only option is to consider non-modular translations. The translation from [Lin and Zhao 2003] transforms a normal program into an inherently tight program, i.e. a program where all the completion models are stable models. As such, it allows to simply compute the completion to transform the inherently tight version of the normal program into a set of clauses. This particular translation is quadratic. An even more compact translation was proposed in [Janhunen 2004] and is currently the only known sub-quadratic translation to a set of clauses.

2.3 Possibility Theory

Possibility theory [Dubois and Prade 1988] is a theory of uncertainty capable of dealing with incomplete information. The theory is defined in terms of a *possibility distribution* π, which maps elements ω from a (finite) universe onto the interval $[0,1]$, i.e. $\pi : \Omega \rightarrow [0,1]$.[3] The function π represents the knowledge of an agent and in particular encodes to what extent the agent finds ω plausible to be the real world. By convention, $\pi(\omega) = 0$ means that ω is impossible and $\pi(\omega) = 1$ means that no available information prevents ω from being the actual world. A possibility distribution π is said to be *normalized* if $\exists \omega \in \Omega \cdot \pi(\omega) = 1$, i.e. at least one interpretation is entirely plausible. Conversely, when $\forall \omega \in \Omega \cdot \pi(\omega) = 0$ we say that the possibility distribution π is *vacuous*. A normalized possibility distribution expresses consistent belief and is thus preferred, as a possibility distribution that is not normalized indicates the presence of conflicting information. Possibility degrees are mainly interpreted qualitatively: when $\pi(\omega) > \pi(\omega')$, ω is considered more plausible than ω'. For two possibility distributions π_1 and π_2 with the same domain Ω we write $\pi_1 \geq \pi_2$ when $\forall \omega \in \Omega \cdot \pi_1(\omega) \geq \pi_2(\omega)$ and $\pi_1 > \pi_2$ when $\pi_1 \geq \pi_2$ as well as $\pi_1 \neq \pi_2$. When we impose constraints on possibility distributions, we are usually only interested in the *least specific possibility distributions*, i.e. the greatest possibility distribution w.r.t. the ordering $>$, that satisfies the constraints.

A possibility distribution π induces two uncertainty measures. The *possibility measure* Π is defined by [Dubois and Prade 1988]:

$$\Pi(A) = \max \{\pi(\omega) \mid \omega \in A\} \text{ with } A \subseteq \Omega$$

and evaluates the extent to which a world ω in A is consistent with the beliefs expressed by π. The dual *necessity measure* N is defined by:

$$N(A) = 1 - \Pi(\overline{A}) \text{ with } A \subseteq \Omega$$

and evaluates the extent to which all possible worlds belong to A. We always have that $N(\Omega) = 1$ since $\Pi(\{\}) = 0$. However, we only have $\Pi(\Omega) = 1$ and, conversely, $N(\{\}) = 0$ when the possibility distribution is normalized. To identify the possibility/necessity measure associated with a specific possibility distribution π_{X}, we will use a subscript notation, i.e. Π_{X} and N_{X} are the corresponding possibility and necessity measure, respectively. We omit the subscript when the possibility distribution is clear from the context.

2.4 Possibilistic Logic

Now we introduce the concepts of possibilistic logic [Dubois et al. 1994], which is a logic capable of dealing with uncertainty based on possibility theory. Thus far, we have seen that interpretations in ASP can be partial. As such, they are different from interpretations

[3]The codomain is often the unit interval $[0,1]$ but can in general be any totally ordered scale.

in classical logic. The semantics of possibilistic logic, on the other hand, are defined w.r.t. classical interpretations. We represent such an interpretation as a set of atoms ω, where $\omega \models a$ if $a \in \omega$ and otherwise $\omega \models \neg a$, with $\models$ the satisfaction relation from classical logic. The set of all interpretations is defined as $\Omega = 2^{\mathcal{A}}$, with $\mathcal{A}$ a finite set of atoms. At the semantic level, possibility distributions over Ω are considered.

Example 12

Consider the possibility distribution π_{12} defined as:

$$\begin{aligned} \pi(\{a,b,c\}) &= 0 & \pi(\{b,c\}) &= 0 \\ \pi(\{a,b\}) &= 1 & \pi(\{b\}) &= 0.4 \\ \pi(\{a,c\}) &= 0 & \pi(\{c\}) &= 0 \\ \pi(\{a\}) &= 0.7 & \pi(\{\}) &= 0.2 \end{aligned}$$

The possibility distribution π_{12} is normalized since the world $\{a, b\}$ is entirely possible. We can see that a world in which 'a' and 'b' are true simultaneously is preferred over worlds in which either 'a' or either 'b' is true since $\pi_{12}(\{a, b\}) > \pi_{12}(\{a\})$ and $\pi_{12}(\{a, b\}) > \pi_{12}(\{b\})$. We can also see that, given this possibility distribution, we do not entertain the possibility that 'c' is true since for every world $\omega \in \Omega$ with $\omega \models c$ we have $\pi_{12}(\omega) = 0$.

The two uncertainty measures from possibility theory can then be used to rank propositions. Indeed, the possibility measure Π can now be written as

$$\Pi(p) = \max \{\pi(\omega) \mid \omega \models p\}$$

which evaluates the extent to which a proposition p is consistent with the beliefs expressed by π [Dubois et al. 1994]. The dual *necessity measure* N defined as

$$N(p) = 1 - \Pi(\neg p)$$

evaluates the extent to which a proposition p is entailed by the available beliefs expressed by π [Dubois et al. 1994]. In particular, the notations $\Pi(p)$ and $N(p)$ with p a proposition are thus defined as shorthands for $\Pi(\{w \mid w \models p\})$ and $N(\{w \mid w \models p\})$. Note that we now always have $N(\top) = 1$ for any possibility distribution, while $\Pi(\top) = 1$ (and $N(\bot) = 0$) only holds when the possibility distribution is normalized, i.e. only normalized possibility distributions can express consistent beliefs [Dubois et al. 1994].

Example 13

Consider the possibility distribution π_{12} from Example 12. Since $\pi(\{a, b\}) = 1$ we find that $\Pi(a) = \Pi(b) = 1$, i.e. it is consistent to believe that 'a' and 'b' will be true in the actual world. Since $\{a, b\} \models \neg c$ we furthermore have that $N(c) = 1 - \Pi(\neg c) = 0$. Indeed, given this possibility distribution, we have no reason to conclude that 'c' is true. We furthermore find that $N(a) = 0.6$ and $N(b) = 0.3$, i.e. we are more certain that 'a' is true.

An important property of necessity measures is their min-decomposability w.r.t. conjunction: $N(p \wedge q) = \min(N(p), N(q))$ for all propositions p and q. However, for disjunction only the inequality $N(p \vee q) \geq \max(N(p), N(q))$ holds. As possibility measures are the dual measures of necessity measures, they have the property of max-decomposability w.r.t. disjunction, while for the conjunction we have that only the inequality $\Pi(p \wedge q) \leq \min(\Pi(p), \Pi(q))$ holds.[4]

At the syntactic level, a *possibilistic knowledge base* consists of pairs (p, c) where p is a propositional formula and $c \in]0, 1]$ expresses the certainty that p is the case. Formulas of the form $(p, 0)$ are not explicitly represented in the knowledge base since they encode trivial information. A formula (p, c) is interpreted as the constraint $N(p) \geq c$, i.e. a possibilistic knowledge base Σ corresponds to a set of constraints on possibility distributions. Typically, there can be many possibility distributions that satisfy these constraints. In practice, we are usually only interested in the *minimally specific possibility distributions*, which are the possibility distributions that make minimal commitments, i.e. the maximal possibility distributions w.r.t. the ordering $>$. For the constraints induced by a possibilistic logic base, there is a unique minimally specific distribution, which is called the least specific distribution [Dubois et al. 1994].

Example 14

Consider the possibilistic knowledge base consisting of the pairs:

$$\begin{array}{ll} (\neg c, 1) & (a \vee b \vee c, 0.8) \\ (\neg a \vee b, 0.3) & (\neg b \vee a, 0.6). \end{array}$$

[4]This is a notable difference with fuzzy logic, which is truth-functional. Furthermore, both logics handle different sources of information. Indeed, in fuzzy logic we are able to express that a bottle *is* half empty, i.e. we deal with multi-valuedness. On the other hand, in possibilistic logic we are able to express that N(full) = 0.5, i.e. we are somewhat certain that the bottle is full. Still, in the actual world the bottle will either be full or empty, as we have no means of expressing multi-valuedness with possibility theory alone.

This knowledge base imposes the constraints

$$N(\neg c) = 1 \qquad N(a \vee b \vee c) \geq 0.8$$
$$N(\neg a \vee b) \geq 0.3 \qquad N(\neg b \vee a) \geq 0.6$$

or, equivalently

$$\Pi(c) = 0 \qquad \Pi(\neg a \wedge \neg b \wedge \neg c) \leq 0.2$$
$$\Pi(a \wedge \neg b) \leq 0.7 \qquad \Pi(b \wedge \neg a) \leq 0.4$$

The least specific possibility distribution that satisfies these constraints is π_{12} from Example 12.

2.5 Possibilistic Answer Set Programming

Finally, we introduce Possibilistic Answer Set Programming (PASP), which combines ASP and possibility theory by associating a weight with each rule. Throughout this thesis, we will use the name PASP for a family of approaches that share a common syntax and which all rely on possibility theory and ASP. In this section we introduce the semantics from [Nicolas et al. 2006], which we will refer to as PASP$_\mathsf{G}$.[5]

2.5.1 Syntax

A possibilistic disjunctive (resp. positive disjunctive, normal, simple, definite) program is a set of pairs $p = (r, c)$ with r a disjunctive (resp. positive disjunctive, normal, simple, definite) rule and $c \in]0, 1]$ a certainty associated with r. As in possibilistic logic we will not, in general, consider pairs with $c = 0$ since such rules encode trivial information. A pair such as $p = (r, c)$ with r a disjunctive rule of the form $(l_0; ...; l_k \leftarrow l_{k+1}, ..., l_m, not\, l_{m+1}, ..., not\, l_n)$ will also often be written in the form:

$$\boldsymbol{\lambda} : l_0; ...; l_k \leftarrow l_{k+1}, ..., l_m, not\ l_{m+1}, ..., not\ l_n.$$

For a possibilistic rule $p = (r, c)$ we use p^* to denote r, i.e. the classical rule obtained by ignoring the certainty. Similarly, for a possibilistic program P we use P^* to denote the set of rules $\{p^* \mid p \in P\}$. The set of all weights found in a possibilistic program P

[5]As we will see in Chapter 4, the treatment of negation-as-failure in the semantics of [Nicolas et al. 2006] can be equated to the use of a Gödel negator.

is denoted by $cert(P) = \{c \mid p = (r, c) \in P\}$. We also use the extended set of weights $cert^+(P) = \{c \mid c \in cert(P)\} \cup \{1 - c \mid c \in cert(P)\} \cup \{0, 1/2, 1\}$. [6]

Example 15

Consider the possibilistic normal program P_{15} with the following set of possibilistic normal rules:

$$\begin{aligned} \mathbf{0.1}\colon & normal \leftarrow \\ \mathbf{1}\colon & abnormal \leftarrow not\ normal \\ \mathbf{0.8}\colon & problematic \leftarrow abnormal \end{aligned}$$

This program describes an automated computer system. Intuitively, as we will see in detail when discussing the semantics, this program models that we have very limited evidence that this system is still operating normally. If the system is no longer operating normally, it is operating abnormally. And if the system is operating abnormally, it is very likely that keeping the system running will result in problematic behaviour. We have that $cert(P_{15}) = \{0.1, 0.8, 1\}$.

2.5.2 Semantics

The semantics of PASP define how one should treat the weight associated with a rule. In PASP$_\mathsf{G}$ this weight is the necessity with which the head of the rule can be concluded, given that the body is known to hold. If it is uncertain whether the body holds, the necessity with which the head can be derived is the minimum of the weight associated with the rule and the degree to which the body is necessarily true.

However, before we can present the semantics, we need to introduce a generalization of the concept of an interpretation. In classical ASP, an interpretation can be seen as a mapping $I : Lit_P \to \{0, 1\}$, i.e. a literal $l \in Lit_P$ is either true or false. This notion is generalized in PASP to a *valuation*, which is a function $V : Lit_P \to [0, 1]$. The underlying intuition of $V(l) = c$ is that the literal 'l' is true with certainty 'c'. Note that, like interpretations in ASP, these valuations are of an epistemic nature, i.e. they reflect what we know about the truth of atoms. For notational convenience, we often also use the set

[6]Programs such as the program with the single rule ($\mathbf{1}\colon a \leftarrow not\ a$) will give rise to the answer set $\left\{a^{1/2}\right\}$ in the semantics which we present in Chapter 4. As such, we require the intermediate weight $1/2$ due to the particular treatment of negation-as-failure in that chapter.

notation $V = \{l^c, \ldots\}$. In accordance with this set notation, we write $V = \emptyset$ to denote the valuation in which each literal is mapped to 0. For $c \in [0,1]$ and V a valuation, we use V^c to denote the set $\{l \mid l \in Lit_P, V(l) \geq c\}$. We also use $V^{\underline{c}} = \{l \mid l \in Lit_P, V(l) > c\}$.

A valuation is said to be *consistent* when $V^{\underline{0}}$ is consistent. In such a case, there always exists a normalized possibility distribution π_V such that $N_V(l) = V(l)$.

We can now formally introduce the semantics. Let the c-cut P_c of a possibilistic program P, with $c \in [0,1]$, be defined as:

$$P_c = \{r \mid (r, c') \in P \text{ and } c' \geq c\},$$

i.e. the rules in P with an associated certainty higher than or equal to 'c'.

Definition 9

Let P be a possibilistic simple program and V a valuation. The immediate consequence operator T_P is defined as:

$$T_P(V)(l_0) = \max\{c \in [0,1] \mid V^c \models l_1, ..., l_m \text{ and } ((l_0 \leftarrow l_1, ..., l_m), c') \in P_c\}.$$

The intuition of Definition 9 is that we can derive the head only with the certainty of the weakest piece of information, i.e. the necessity of the conclusion is restricted either by the certainty of the rule itself or the lowest certainty of the literals used in the body of the rule. Note that the immediate consequence operator defined in Definition 9 is equivalent to the one proposed in [Nicolas et al. 2006], although we formulate it somewhat differently. Also, the work from [Nicolas et al. 2006] only considered definite programs, even though adding classical negation does not impose any problems and is a fairly trivial extension. Indeed, even if we did not extend the definitions to also allow for classical negation it would still be possible to simulate classical negation in ASP [Baral 2003] (and thus also in PASP).

As before, we use $P^\star$ to denote the fixpoint obtained by repeatedly applying T_P starting from the minimal valuation $V = \emptyset$, i.e. the least fixpoint of T_P w.r.t. set inclusion. A valuation V is said to be the *possibilistic answer set* of a possibilistic simple program if $V = P^\star$ and V is consistent. Possibilistic answer sets of possibilistic normal programs are defined using a reduct. Let L be a set of literals. The reduct P^L of a possibilistic normal program is defined as [Nicolas et al. 2006]:

$$P^L = \{(head(r) \leftarrow body_+(r), c) \mid (r, c) \in P \text{ and } body_-(r) \cap L = \emptyset\}.$$

A consistent valuation V is said to be a possibilistic answer set of the possibilistic normal program P iff $\left(P^{(V^{\underline{0}})}\right)^\star = V$, i.e. if V is the possibilistic answer set of the reduct $P^{(V^{\underline{0}})}$.

Example 16

Consider the possibilistic normal program P_{15} from Example 15. It is easy to verify that $\left\{normal^{0.1}\right\}$ is a possibilistic answer set of P. Indeed, $P^{\{normal\}}$ is the set of rules:

$$\mathbf{0.1}\colon normal \leftarrow \qquad\qquad \mathbf{0.8}\colon problematic \leftarrow abnormal$$

from which it trivially follows that $P^{\star} = \left\{normal^{0.1}\right\}$ since the body of the second rule is never true.

In Chapter 4 we discuss how the semantics of PASP_G can have unintuitive results due to the particular treatment of negation-as-failure in the presence of uncertainty. Similarly, the semantics proposed in [Nieves et al. 2007] to extend PASP_G to disjunctive possibilistic answer set programming share the same issues. To overcome these problems, we present new semantics for PASP in Chapter 4 that adhere to a different intuition. Furthermore, we highlight how these new semantics for PASP allow us to unearth a new form of disjunction, both in the possibilistic and the classical case. Then, in Chapter 5, we show how the treatment of weights in PASP is not unique. Specifically, we show how the weights can also be interpreted as the certainty of the rule itself, rather than the certainty of the conclusion (i.e. the certainty of the head of the rule).

3 Communicating Answer Set Programming

3.1 Introduction

A multi-context system (MCS) [Giunchiglia and Serafini 1994] is a framework to formalise the reasoning about belief in a multi-agent system, which is a system that allows us to model agents, their environment and their interaction with each other. In general, the agents in a multi-agent framework can be located on physically distinct machines, which implies that the computations in a multi-agent framework often need to be made in a decentralised way. In a MCS, however, we are mainly interested in the structure of the problem and we want to be able to compute the solutions in a centralised way. The idea of a MCS is that we have a number of contexts that each have access to only a subset of the available information. Each context has a local model and local reasoning capabilities, i.e. each context is a finite representation of the reasoner's beliefs and its reasoning capabilities. As discussed in Chapter 1, the ability to communicate and exchange information is an important part of common-sense reasoning. This is no different in a MCS, where we also define an information flow between the different contexts. Because of this information flow, information can be exchanged and we can solve multi-dimensional problems (e.g. a police investigation with multiple witnesses) where each context/agent only has the knowledge contained in one or a few of the dimensions (e.g. the witness only saw the burglar enter the building). As such, a MCS can be used for epistemic reasoning, where the knowledge/beliefs of an agent are modelled in a context and where communication allows for information to be exchanged to solve the problem at hand.

A standard example to illustrate the usefulness of multi-context systems is shown in Figure 3.1.[1] The figure depicts two people who are looking at a box. The box is called magic because neither agent can make out its depth. The information the agents have is further limited because parts of the box are blinded. By cooperation, both agents can pinpoint the exact location of the ball. Indeed, $Mr.2$ sees a ball on his left side. From this information $Mr.1$ knows that there is a ball and that it must therefore be on his left (since he cannot see a ball on his right). This knowledge can be relayed back to $Mr.2$. Both agents now know the exact position and depth of the ball.

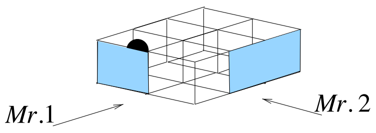


Figure 3.1: A magic box.

ASP seems to be an ideal language to specify the individual contexts in a MCS. Indeed, since we are modelling the belief of an agent, we may, at some point, need to revise earlier decisions as new information becomes available. Indeed, the need for such a non-monotonic operator was propounded in [Roelofsen and Serafini 2005]. To allow us to model these types of problems in ASP, however, we need to extend ASP with means of communication. Indeed, as we have seen in Chapter 2, classical ASP does not have the mechanisms to allow for communication between ASP programs or for communication between an ASP program with an outside source.

Unsurprisingly, the combination of ASP, or logic programming in general, with communication, or multi-agent systems in general, is nothing new. Extensions of logic programming that offer such a combination have been explored for a variety of reasons. For example, such an extension can be used to describe the (rational) behaviour of the agents in a multi-agent network, as in [Dell'Acqua et al. 1999]. Alternatively, it can be used to combine different flavours of logic programming languages [Luo et al. 2005, Eiter et al. 2008]. It can also be used to externally solve tasks for which ASP is not suited, while remaining in a declarative framework [Eiter et al. 2006].

The combination that we are interested in is where ASP is combined with communication to allow for a form of cooperation as in a multi-context system (MCS). Specifically, this

[1]Illustration from [Roelofsen and Serafini 2005] used with permission from the authors.

allows the contexts to share information which in turn allows them to conjointly solve a difficult problem. As in a MCS, we only consider multiple contexts on a conceptual level, i.e. we model these different contexts in a single program, for which we want compute the solutions in a centralised way. We are particularly interested in how such a conceptual collaboration of different ASP programs affects the expressiveness of the overall system. Still, unearthing the complexity of the addition of communication is difficult. Complexity results from [Brewka et al. 2007] show that computing the answer sets of a logic program extended with a form of communication is in NP, i.e. communication does not affect the expressiveness of this system. However, the work from [Van Nieuwenborgh et al. 2007] describes a multi-context system based on ASP where each context is modelled using the extended answer set semantics, which is a fairly expressive variant of ASP. The network is a linear "hierarchical" network (i.e. information only flows in one direction) where the idea of a failure feedback mechanism is used. Intuitively, a failure feedback mechanism allows the previous agent in a network to revise his conclusion when the conclusion leads to an unresolvable inconsistency for the next agent in the network. The work in [Van Nieuwenborgh et al. 2007] gives rise to a higher expressiveness, namely Σ_n^{P} for a hierarchical network of n agents. In other words: the specific form of communication used in [Van Nieuwenborgh et al. 2007] affected the expressiveness of the overall system.

In this chapter we present a systematic study of the additional expressiveness offered by allowing ASP programs to communicate. The structure of this chapter is as follows. In Section 3.2 we introduce Communicating Answer Set Programming (CASP), which are ordinary ASP programs extended with a new kind of literal that allows for communication between the individual programs. This new form of communication is based on the notion of asking questions, where one ASP program can query another program as to whether it believes some literal to be true or not. CASP is different from previous works in that we start from simple and normal ASP programs for the contexts. This allows us to better identify what exactly contributes to an increase in expressiveness when we allow ASP programs to communicate. In addition, also our communication mechanism is simple and does not rely on any kind of feedback as in [Van Nieuwenborgh et al. 2007]. While we thus consider simple programs and a simple communication mechanism, we show in Section 3.3 that communicating simple programs are sufficiently expressive to simulate (communicating) normal programs. Importantly, this shows that communication does increase the expressiveness. Still, in line with the results from [Brewka et al. 2007], we find that combining both communication and negation-as-failure does not further increase the expressiveness.

Furthermore, we illustrate that the introduction of communication in ASP ensues that the concept of minimality becomes ambiguous. Recall that answer sets are minimal models of the program. For example, if $\{a\}$ and $\{a, b\}$ are the only models of a program P, then $\{a\}$

is the answer set since $\{a\} \subseteq \{a, b\}$. In CASP, however, information is deduced *in a given context*. To denote that we can derive 'a' and 'b' from context Q and 'c' from context R we would write $\{Q\!:\!a, Q\!:\!b, R\!:\!c\}$ in accordance with the notation that we will introduce in Section 3.2. Now assume that the sets $\{Q\!:\!a, Q\!:\!b, R\!:\!c\}$ and $\{Q\!:\!a, R\!:\!c, R\!:\!d\}$ are the only two models of a given CASP program. On a global level, i.e. when we do not take the context into account and interpret e.g. $Q\!:\!a$ as a literal, then both models are minimal models. On a local level this need not be the case. For example, when we only look at the information deduced from Q, then only the second model is an answer set since $\{a, b\} \supseteq \{a\}$. If we look at the information deduced from R, then only the first model is an answer set. This idea, where we determine the minimality on the level of a single program (or context) is called focussing and is developed in Section 3.4. Specifically, we will show that this idea can be applied repeatedly, where we focus on a sequence of distinct ASP programs used in a communicating program. These multi-focused answer sets considerably increase the expressiveness compared to the expressiveness of each individual program. Indeed, we step up one level in the polynomial hierarchy for each time we focus. Using multi-focused answer set programs thus allows us to express any problem in PSPACE (i.e. the set of all decision problems that can be solved by a Turing machine using a polynomial amount of space). As such, we can solve certain complex problems for which the current semantics of ASP lack the expressivity. In the case study from Section 3.5 we show how such complex problems can indeed be intuitively modelled using the framework introduced in this chapter.

3.2 Communicating Programs

The underlying intuition of communication between ASP programs is that of a function call or, in terms of agents, asking questions to other agents. This communication is based on a new kind of literal '$Q\!:\!l$', as in [Giunchiglia and Serafini 1994, Roelofsen and Serafini 2005, Brewka and Eiter 2007]. If the literal l is not in the answer set of program Q then $Q\!:\!l$ is false; otherwise $Q\!:\!l$ is true. The semantics presented in this section are closely related to the minimal semantics of [Brewka and Eiter 2007] and especially the semantics of [Buccafurri et al. 2008].

Let $\mathcal{P}$ be a finite set of program names. A *$\mathcal{P}$-situated literal* is an expression of the form $Q\!:\!l$ with $Q \in \mathcal{P}$ and l a literal. For $R \in \mathcal{P}$, a $\mathcal{P}$-situated literal $Q\!:\!l$ is called *R-local* if $Q = R$. For a set of literals L, we use $Q\!:\!L$ as a shorthand for $\{Q\!:\!l \mid l \in L\}$. For a set of $\mathcal{P}$-situated literals X and $Q \in \mathcal{P}$, we use X_Q to denote $\{l \mid Q\!:\!l \in X\}$, i.e. the projection of X on Q. A set of $\mathcal{P}$-situated literals X is *consistent* iff X_Q is consistent for all $Q \in \mathcal{P}$. By $\neg X$ we denote the set $\{Q\!:\!\neg l \mid Q\!:\!l \in X\}$ where we define $Q\!:\!\neg\neg l = Q\!:\!l$. An *extended $\mathcal{P}$-situated literal* is either a $\mathcal{P}$-situated literal or a $\mathcal{P}$-situated literal preceded by not. For a set of $\mathcal{P}$-situated literals X, we use $not(X)$ to denote the set $\{not\ Q{:}l \mid Q{:}l \in X\}$. For a set of extended $\mathcal{P}$-situated literals X we denote by X_{pos} the set of $\mathcal{P}$-situated literals in X, i.e. those extended $\mathcal{P}$-situated literals in X that are not preceded by negation-as-failure, while $X_{\text{neg}} = \{Q{:}l \mid not\ Q{:}l \in X\}$.

A *$\mathcal{P}$-situated disjunctive rule* is an expression of the form $Q{:}\gamma \leftarrow (\alpha \cup not(\beta))$ where γ is a set of literals, called the head of the rule, and $(\alpha \cup not(\beta))$ is called the body of the rule with α and β sets of $\mathcal{P}$-situated literals. A $\mathcal{P}$-situated disjunctive rule $Q{:}\gamma \leftarrow (\alpha \cup not\ (\beta))$ is called R-local whenever $Q = R$. A *$\mathcal{P}$-component disjunctive program Q* is a finite set of Q-local $\mathcal{P}$-situated disjunctive rules. Henceforth we shall use $\mathcal{P}$ to both denote the set of program names and to denote the set of actual $\mathcal{P}$-component disjunctive programs. A *communicating disjunctive program $\mathcal{P}$* is then a finite set of $\mathcal{P}$-component disjunctive programs.

A *$\mathcal{P}$-situated normal rule* is an expression of the form $Q{:}l \leftarrow (\alpha \cup not(\beta))$ where $Q{:}l$ is a single $\mathcal{P}$-situated literal. A *$\mathcal{P}$-situated simple rule* is an expression of the form $Q{:}l \leftarrow \alpha$, i.e. a $\mathcal{P}$-situated normal rule without negation-as-failure. A *$\mathcal{P}$-component normal (resp. simple) program Q* is a finite set of Q-local $\mathcal{P}$-situated normal (resp. simple) rules. A *communicating normal (resp. simple) program $\mathcal{P}$* is then a finite set of $\mathcal{P}$-component normal (resp. simple) programs.

In the remainder of this chapter we drop the $\mathcal{P}$-prefix whenever the set $\mathcal{P}$ is clear from the context. Whenever the name of the component disjunctive program Q is clear, we write l instead of $Q{:}l$ for Q-local situated literals. Note that a communicating disjunctive (resp. normal, simple) program with only one component program thus trivially corresponds to a classical disjunctive (resp. normal, simple) program. Finally, for notational convenience, we write communicating program when it is clear from the context whether the program is a communicating simple program or a communicating normal program. Similarly, we simply write answer set when it is clear from the context that it is a communicating answer set.

Example 17

Consider the communicating normal program $\mathcal{P} = \{Q, R\}$ with the following situated rules:

$$\begin{array}{lll} Q{:}a \leftarrow R{:}a & Q{:}b \leftarrow & Q{:}c \leftarrow Q{:}c \\ R{:}a \leftarrow Q{:}a & R{:}b \leftarrow not\ Q{:}c. & \end{array}$$

$Q{:}a$, $Q{:}b$, $Q{:}c$, $R{:}a$ and $R{:}b$ are situated literals. The situated simple rules on the top line are Q-local since we respectively have $Q{:}a$, $Q{:}b$ and $Q{:}c$ in the head of these rules. The situated normal rules on the bottom line are R-local. Hence $Q = \{a \leftarrow R\!:\!a, b \leftarrow, c \leftarrow c\}$ and $R = \{a \leftarrow Q\!:\!a, b \leftarrow \mathit{not}\ Q\!:\!c\}$.

Similar as for a classical program, we can define the *Herbrand base* for a component program Q as the set of atoms $\mathcal{B}_Q = \{a \mid Q\!:\!a \text{ or } Q\!:\!\neg a \text{ appearing in } Q\}$, i.e. the set of atoms occurring in the Q-local situated literals in Q. Similarly, we are then able to define $\mathcal{B}_\mathcal{P} = \left\{Q\!:\!a \mid Q \in \mathcal{P} \text{ and } a \in \bigcup_{R \in \mathcal{P}} \mathcal{B}_R\right\}$ as the *Herbrand base* of the communicating program $\mathcal{P}$.

Example 18

We consider the communicating normal program $\mathcal{P} = \{Q, R\}$ from Example 17. We have $\mathcal{B}_Q = \{a, b, c\}$, $\mathcal{B}_R = \{a, b\}$ and $\mathcal{B}_\mathcal{P} = \{Q\!:\!a, Q\!:\!b, Q\!:\!c, R\!:\!a, R\!:\!b, R\!:\!c\}$.

We say that a (partial) interpretation I of a communicating disjunctive program $\mathcal{P}$ is any consistent subset $I \subseteq (\mathcal{B}_\mathcal{P} \cup \neg\mathcal{B}_\mathcal{P})$. Given an interpretation I of a communicating disjunctive program $\mathcal{P}$, the *reduct* Q^I for $Q \in \mathcal{P}$ is the component disjunctive program obtained by deleting

- each rule with an extended situated literal '$\mathit{not}\ R\!:\!l$' such that $R\!:\!l \in I$;
- each remaining extended situated literal of the form '$\mathit{not}\ R\!:\!l$';
- each rule with a situated literal '$R\!:\!l$' that is not Q-local such that $R\!:\!l \notin I$;
- each remaining situated literal '$R\!:\!l$' that is not Q-local.

Note that this definition actually combines two types of reducts together. On the one hand, we remove negation-as-failure according to the given knowledge. On the other hand, we also remove situated literals that are not Q-local, again according to the given knowledge. The underlying intuition of the reduct remains unchanged compared to the classical case: we take the information into account which is encoded in the guess I and we simplify the program so that we can easily verify whether or not I is stable, i.e. whether or not I is a minimal model of the reduct. Analogous to the definition of the reduct for disjunctive programs [Gelfond and Lifschitz 1991], the reduct of a communicating disjunctive program thus defines a way to reduce a program relative to some guess I. The reduct of a communicating disjunctive program is a communicating disjunctive program (without negation-as-failure) that only contains component disjunctive programs Q with Q-local situated literals. That is, each remaining component disjunctive program Q corresponds to a classical disjunctive program.

Example 19

Let us again consider the communicating normal program $\mathcal{P} = \{Q, R\}$ from Example 17. Given $I = \{Q\!:\!a, Q\!:\!b, R\!:\!a, R\!:\!b\}$ we find that $Q^I = \{a \leftarrow, b \leftarrow, c \leftarrow c\}$ and $R^I = \{a \leftarrow, b \leftarrow\}$. We can easily treat Q^I and R^I separately since they now correspond to classical programs.

Definition 10

We say that an interpretation I of a communicating disjunctive program $\mathcal{P}$ is an *answer set* of $\mathcal{P}$ if and only if $\forall Q \in \mathcal{P} \cdot (Q\!:\!I_Q)$ is the minimal model w.r.t. set inclusion of Q^I. In other words: an interpretation I is an answer set of a communicating disjunctive program $\mathcal{P}$ if and only if for every component program Q we have that the projection of I on Q is an answer set of the component program Q^I under the classical definition.

In the specific case of a communicating normal program $\mathcal{P}$ we can equivalently say that I is an answer set of $\mathcal{P}$ if and only if we have that $\forall Q \in \mathcal{P} \cdot (Q\!:\!I_Q) = \left(Q^I\right)^{\star}$.

Example 20

The communicating normal program $\mathcal{P} = \{Q, R\}$ from Example 17 has two answer sets, namely $\{Q\!:\!b, R\!:\!b\}$ and $\{Q\!:\!a, Q\!:\!b, R\!:\!a, R\!:\!b\}$.

Example 21

Consider the magic box as illustrated in Figure 3.1. Let us assume that $Mr.1$ names the locations north and south (where he does not see the ball in the southern location) and that $Mr.2$ names the locations left, middle and right with the ball in the leftmost location. We have the communicating simple program $\mathcal{P}$ with the rules:

$$\begin{aligned}
&Mr1 : south \leftarrow Mr1 : \neg north && Mr1 : left \leftarrow Mr2 : left \\
&Mr1 : north \leftarrow Mr1 : \neg south && Mr1 : middle \leftarrow Mr2 : middle \\
&Mr1 : \neg south \leftarrow && Mr1 : right \leftarrow Mr2 : right \\
\\
&Mr2 : left \leftarrow && Mr2 : north \leftarrow Mr1 : north \\
& && Mr2 : south \leftarrow Mr1 : south
\end{aligned}$$

The unique communicating answer set of $\mathcal{P}$ is

$$\{Mr1 : north, Mr1 : left, Mr2 : north, Mr2 : left\}\,.$$

Both $Mr.1$ and $Mr.2$ thus know the exact location of the ball in this magic box.

Note that while most approaches do not allow *self-reference*s of the form $Q{:}a \leftarrow Q{:}a$, in our approach this poses no problems as it is semantically equivalent to $a \leftarrow a$. Also note that our semantics allow for "*mutual influence*" as in [Brewka and Eiter 2007, Buccafurri et al. 2008] where the belief of an agent can be supported by the agent itself, via belief in other agents, e.g. $\{Q{:}a \leftarrow R{:}a, R{:}a \leftarrow Q{:}a\}$. Furthermore we want to point out that the belief between agents is the belief as identified in [Lifschitz et al. 1999], i.e. the situated literal $Q{:}l$ is true in our approach whenever "$\neg not\ Q{:}l$" is true in the approach introduced in [Lifschitz et al. 1999] for nested logic programs and treating $Q{:}l$ as a fresh atom.

Before we introduce our first proposition, we generalise the immediate consequence operator for (classical) normal programs to the case of communicating simple programs. Specifically, the operator $T_{\mathcal{P}}$ is defined w.r.t. an interpretation I of $\mathcal{P}$ as

$$T_{\mathcal{P}}(I) = I \cup \{Q{:}l \mid (Q{:}l \leftarrow \alpha) \in Q, Q \in \mathcal{P}, \alpha \subseteq I\}$$

where α is a set of $\mathcal{P}$-situated literals. It is easy to see that this operator is monotone. Together with a result from [Tarski 1955] we know that this operator has a least fixpoint. We use $\mathcal{P}^\star$ to denote this fixpoint obtained by repeatedly applying $T_{\mathcal{P}}$ starting from the empty interpretation. Clearly, this fixpoint can be computed in polynomial time. Furthermore, just like the immediate consequence operator for (classical) normal programs, this generalised operator only derives the information that is absolutely necessary, i.e. the fixpoint $\mathcal{P}^\star$ is globally minimal.

Proposition 6

Let $\mathcal{P}$ be a communicating simple program. We then have that:

- there always exists at least one answer set of $\mathcal{P}$;
- there is always a unique answer set of $\mathcal{P}$ that is globally minimal;
- we can compute this unique globally minimal answer set in polynomial time.

Example 22

Consider the communicating simple program $\mathcal{P}$ with the rules

$$Q\!:\!a \leftarrow R\!:\!a \qquad\qquad R\!:\!a \leftarrow Q\!:\!a \qquad\qquad Q\!:\!b \leftarrow .$$

This communicating simple program has two answer sets, namely $\{Q\!:\!a, Q\!:\!b, R\!:\!a\}$ and $\{Q\!:\!b\}$. We have that $\mathcal{P}^\star = \{Q\!:\!b\}$, i.e. $\{Q\!:\!b\}$ is the answer set that can be computed in polynomial time. Intuitively, this is the answer set of the communicating simple program $\mathcal{P}$ where we treat every situated literal as an ordinary literal. For example, if we replace the situated literal $Q\!:\!a$ (resp. $Q\!:\!b$, $R\!:\!a$) by the literals qa (resp. qb, ra) we obtain the simple program

$$qa \leftarrow ra \qquad\qquad qb \leftarrow \qquad\qquad ra \leftarrow qa$$

which has the unique answer set $\{qb\}$, with qb the literal that replaced $Q\!:\!b$. Note that the procedure involving the generalised fixpoint does not allow us to derive the second answer set. In general, no polynomial procedure will be able to verify whether there is some answer set in which a given literal is true (unless P=NP).

Although finding an answer set of a communicating simple program can be done in polynomial time, we will see in the next section that brave reasoning (the problem of determining whether a given situated literal $Q\!:\!l$ occurs in any answer set of a communicating simple program) is NP-hard. Furthermore, cautious reasoning (the problem of determining whether a given situated literal $Q\!:\!l$ occurs in all answer sets of a communicating simple program) is coNP-hard.

3.3 Simulating Negation-as-Failure with Communication

The addition of communication to ASP programs can provide added expressiveness over simple programs and a resulting increase in computational complexity for brave reasoning and cautious reasoning. To illustrate this observation, in this section we show that a communicating simple program can simulate normal programs.[2] Furthermore, we illustrate that, surprisingly, there is no difference in terms of computational complexity between communicating simple programs and communicating normal programs; a communicating simple program can be constructed which simulates any given communicating normal program.

[2]Recall that simple programs are P-complete and normal programs are NP-complete [Baral 2003].

We start by giving an example of the transformation that allows us to simulate (communicating) normal programs using communicating simple programs. A formal definition of the simulation is given below in Definition 11. The correctness is proven by Propositions 7 and 8.

Example 23

Consider the communicating normal program $\mathcal{P}$ with the rules

$$\begin{aligned} Q_1 : a &\leftarrow not\ Q_2 : b \\ Q_2 : b &\leftarrow not\ Q_1 : a. \end{aligned}$$

Note that if we were to take $Q_1 = Q_2$ then this example corresponds to a normal program. In our simulation, the communicating normal program $\mathcal{P}$ is transformed into the following communicating simple program $\mathcal{P}' = \{Q_1', Q_2', N_1, N_2\}$:

$$\begin{aligned} Q_1' : a &\leftarrow N_2 : \neg b^\dagger & N_1 : a^\dagger &\leftarrow Q_1' : a \\ Q_2' : b &\leftarrow N_1 : \neg a^\dagger & N_2 : b^\dagger &\leftarrow Q_2' : b \\ Q_1' : \neg a^\dagger &\leftarrow N_1 : \neg a^\dagger & N_1 : \neg a^\dagger &\leftarrow Q_1' : \neg a^\dagger \\ Q_2' : \neg b^\dagger &\leftarrow N_2 : \neg b^\dagger & N_2 : \neg b^\dagger &\leftarrow Q_2' : \neg b^\dagger. \end{aligned}$$

The transformation creates two types of component programs or *'worlds'*, namely Q_i' and N_i. The component program Q_i' is similar to Q_i but occurrences of extended situated literals of the form $not\ Q_i : l$ are replaced by $N_i : \neg l^\dagger$, with $l^\dagger$ a fresh literal. The non-monotonicity associated with negation-as-failure is simulated by introducing the rules $\neg l^\dagger \leftarrow N_i : \neg l^\dagger$ and $\neg l^\dagger \leftarrow Q_i' : \neg l^\dagger$ in Q_i' and N_i, respectively. Finally, we add rules of the form $l^\dagger \leftarrow Q_i' : l$ to N_i, creating an inconsistency when N_i believes $\neg l^\dagger$ and Q_i' believes l.

The resulting communicating simple program $\mathcal{P}'$ is an equivalent program in that its answer sets correspond to those of the original communicating normal program, yet without using negation-as-failure. Indeed, the answer sets of $\mathcal{P}$ are $\{Q_1 : a\}$ and $\{Q_2 : b\}$ and the answer sets of $\mathcal{P}'$ are $\{Q_1' : a\} \cup \left\{Q_2' : \neg b^\dagger, N_2 : \neg b^\dagger, N_1 : a^\dagger\right\}$ and $\{Q_2' : b\} \cup \left\{Q_1' : \neg a^\dagger, N_1 : \neg a^\dagger, N_2 : b^\dagger\right\}$.

Note that the simulation given in Example 23 can in fact be simplified. Indeed, in this particular example there is no need to have two additional component programs N_1 and N_2 since Q_1 and Q_2 do not share literals. Also, in this particular example, we need not use '$a^\dagger$' and '$b^\dagger$' since the simulation would work just as well if we simply considered 'a' and 'b' instead. Nonetheless, for the generality of the simulation such technicalities are necessary. Without adding an additional component program N_i for every original component program Q_i the simulation would in general not work when two component programs shared literals, e.g. $Q_1 : a$ and $Q_2 : a$. Furthermore, we need to introduce fresh literals as otherwise the simulation would in general not work when we had true negation in the original program, e.g. $Q : \neg a$. We now give the definition of the simulation which works in the general case.

Definition 11

Let $\mathcal{P} = \{Q_1, ..., Q_n\}$ be a communicating normal program. The communicating simple program $\mathcal{P}' = \{Q'_1, ..., Q'_n, N_1, ..., N_n\}$ with $1 \leq i, j \leq n$ that simulates $\mathcal{P}$ is defined by

$$\begin{aligned}
Q'_i &= \left\{l \leftarrow \alpha'_{\text{pos}} \cup \left\{N_j : \neg k^\dagger \mid Q_j : k \in \alpha_{\text{neg}}\right\} \mid (l \leftarrow \alpha) \in Q_i\right\} && (3.1)\\
&\cup \left\{\neg b^\dagger \leftarrow N_i : \neg b^\dagger \mid Q_i : b \in \mathcal{E}_{\text{neg}}\right\} && (3.2)\\
N_i &= \left\{\neg b^\dagger \leftarrow Q'_i : \neg b^\dagger \mid Q_i : b \in \mathcal{E}_{\text{neg}}\right\} && (3.3)\\
&\cup \left\{b^\dagger \leftarrow Q'_i : b \mid Q_i : b \in \mathcal{E}_{\text{neg}}\right\} && (3.4)
\end{aligned}$$

with $\alpha' = \left\{Q'_j : l \mid Q_j : l \in \alpha\right\}$, $\mathcal{E}_{\text{neg}} = \bigcup_{i=1}^{n} \left(\bigcup_{(a \leftarrow \alpha) \in Q_i} \alpha_{\text{neg}}\right)$ and with α_{pos} and α_{neg} as defined before.

Note how this is a polynomial transformation with at most $3 \cdot |\mathcal{E}_{\text{neg}}|$ additional rules. This is important when later we use the NP-completeness results from normal programs to show that communicating simple programs are NP-complete as well. Recall that both $\neg b^\dagger$ and $b^\dagger$ are fresh literals that intuitively correspond to $\neg b$ and b. We use $Q'_i{+}$ to denote the set of rules in Q'_i defined by (3.1) and $Q'_i{-}$ to denote the set of rules in Q'_i defined by (3.2).

The intuition of the simulation in Definition 11 is as follows. The simulation uses the property of mutual influence to mimic the choice induced by negation-as-failure. This is obtained from the interplay between rules (3.2) and (3.3). As such, we can use the new literal '$\neg b^\dagger$' instead of the original extended (situated) literal '$not\ b$', allowing us to rewrite the rules as we do in (3.1). In order to ensure that the simulation works even when the program we want to simulate already contains classical negation, we need to specify some additional bookkeeping (3.4).

As will become clear from Proposition 7 and Proposition 8, the above transformation preserves the semantics of the original program. Since we can thus rewrite any normal program as a communicating simple program, the importance is twofold. On one hand, we reveal that communicating normal programs do not have any additional expressive power over communicating simple programs. On the other hand, it follows that communicating simple programs allow us to solve NP-complete problems. Before we show the correctness of the simulation in Definition 11, we introduce a lemma.

Lemma 1

Let $\mathcal{P} = \{Q_1, ..., Q_n\}$ and let $\mathcal{P}' = \{Q'_1, ..., Q'_n, N_1, ..., N_n\}$ with $\mathcal{P}$ a communicating normal program and $\mathcal{P}'$ the communicating simple program that simulates $\mathcal{P}$ defined in Definition 11. Let M be an answer set of $\mathcal{P}$ and let the interpretation M' be defined as:

$$\begin{aligned} M' = &\{Q'_i : a \mid Q_i : a \in M\} \\ &\cup \left\{Q'_i : \neg b^\dagger \mid Q_i : b \notin M\right\} \\ &\cup \left\{N_i : \neg b^\dagger \mid Q_i : b \notin M\right\} \\ &\cup \left\{N_i : a^\dagger \mid Q_i : a \in M\right\}. \end{aligned} \tag{3.5}$$

For each $i \in \{1, ..., n\}$ it holds that $(Q'_i+)^{M'} = \left\{l \leftarrow \alpha' \mid l \leftarrow \alpha \in Q_i^M\right\}$ with Q'_i+ the set of rules defined in (3.1) with $\alpha' = \{Q'_i : b \mid Q_i : b \in \alpha\}$.

Using this lemma, we can prove that M' as defined in Lemma 1 is indeed an answer set of the communicating simple program that simulates the communicating normal program $\mathcal{P}$ when M is an answer set of $\mathcal{P}$.

Proposition 7

Let $\mathcal{P} = \{Q_1, ..., Q_n\}$ and let $\mathcal{P}' = \{Q'_1, ..., Q'_n, N_1, ..., N_n\}$ with $\mathcal{P}$ a communicating normal program and $\mathcal{P}'$ the communicating simple program that simulates $\mathcal{P}$ as defined in Definition 11. If M is an answer set of $\mathcal{P}$, then M' is an answer set of $\mathcal{P}'$ with M' defined as in Lemma 1.

Next we introduce Lemma 2, which is similar to Lemma 1 in approach but which states the converse.

Lemma 2

Let $\mathcal{P} = \{Q_1, ..., Q_n\}$ and let $\mathcal{P}' = \{Q'_1, ..., Q'_n, N_1, ..., N_n\}$ with $\mathcal{P}$ a communicating normal program and $\mathcal{P}'$ the communicating simple program that simulates $\mathcal{P}$. Assume that M' is an answer set of $\mathcal{P}'$ and that $(M')_{N_i}$ is total w.r.t. $\mathcal{B}_{N_i}$ for all $i \in \{1, ..., n\}$. Let M be defined as

$$M = \left\{Q_i : b \mid Q'_i : b \in \left(\left(Q'_i+\right)^{M'}\right)^\star\right\} \tag{3.6}$$

For each $i \in \{1, ..., n\}$, it holds that $(Q'_i+)^{M'} = \left\{l \leftarrow \alpha' \mid l \leftarrow \alpha \in Q_i^M\right\}$ with $\alpha' = \{Q'_i : b \mid Q_i : b \in \alpha\}$.

Proposition 8

Let $\mathcal{P} = \{Q_1, ..., Q_n\}$ and let $\mathcal{P}' = \{Q'_1, ..., Q'_n, N_1, ..., N_n\}$ with $\mathcal{P}$ a communicating normal program and $\mathcal{P}'$ the communicating simple program that simulates $\mathcal{P}$. Assume that M' is an answer set of $\mathcal{P}'$ and that $(M')_{N_i}$ is total w.r.t. $\mathcal{B}_{N_i}$ for all $i \in \{1, ..., n\}$. Then the interpretation M defined in Lemma 2 is an answer set of $\mathcal{P}$.

It is important to note that Lemma 2 and, by consequence, Proposition 8 require (part of) the answer set M' to be total. This is a necessary requirement, as demonstrated by the following example.

Example 24

Consider the normal program $R = \{a \leftarrow \mathit{not}\ a\}$ which has no answer sets. The corresponding communicating simple program $\mathcal{P}' = \{Q', N\}$ has the following rules:

$$
\begin{aligned}
Q' : a &\leftarrow N : \neg a^\dagger & \qquad N : \neg a^\dagger &\leftarrow Q' : \neg a^\dagger \\
Q' : \neg a^\dagger &\leftarrow N : \neg a^\dagger & \qquad N : a^\dagger &\leftarrow Q' : a.
\end{aligned}
$$

It is easy to see that $I = \emptyset$ is an answer set of $\mathcal{P}'$ since we have ${Q'}^I = N^I = \emptyset$. Notice that I does not correspond with an answer set of R, which is due to $I_N = \emptyset$ not being total and hence we cannot apply Proposition 8.

Regardless, it is easy to see that the requirement for the answer set to be total can be built into the simulation program. Indeed, it suffices to introduce additional rules to every N_i with $1 \leq i \leq n$ in the simulation defined in Definition 11. These rules are

$$
\begin{aligned}
&\left\{N_i : a \leftarrow N_i : a^\dagger, N_i : a \leftarrow N_i : \neg a^\dagger \mid a^\dagger \in \mathcal{B}_{N_i}\right\} \\
&\cup \{N_i : \mathit{total} \leftarrow \beta\} \text{ with } \beta = \left\{N_i : a \mid a^\dagger \in \mathcal{B}_{N_i}\right\}.
\end{aligned}
$$

Thus the requirement that (part of) the answer set must be total can be replaced by the requirement that the situated literals $N_i : \mathit{total}$ must be true in the answer set. Hence, if we want to check whether a literal $Q : l$ is true in at least one answer set of a (communicating) normal program, it suffices to check whether $Q : l$ and $N_i : \mathit{total}$ can be derived in the communicating simple program that simulates it. Clearly we find that brave reasoning for communicating simple programs is NP-hard.

What we have done so far is important for two reasons. First, we have shown that the complexity of brave reasoning for communicating normal programs is not harder than brave reasoning for communicating simple programs. Indeed, the problem of brave reasoning for communicating normal programs can be reduced in polynomial time to the problem of brave reasoning for communicating simple programs. Second, since normal programs are a special case of communicating normal programs and since we know that brave reasoning for normal programs is an NP-complete problem, we have successfully shown that brave reasoning for communicating simple programs is NP-hard. In order to show that brave reasoning for communicating simple programs is NP-complete, we need to additionally show that this is a problem in NP. To this end, consider the following algorithm to find the answer sets of a communicating simple program $\mathcal{P}$:

```
guess an interpretation $I \subseteq (\mathcal{B}_\mathcal{P} \cup \neg\mathcal{B}_\mathcal{P})$
verify that this interpretation is an answer set as follows:
    calculate the reduct $Q^I$ of each component program $Q$
    calculate the fixpoint of each simple component program $Q^I$
    verify that $Q : I_Q = (Q^I)^\star$ for each component program $Q$
```

The first step of the algorithm requires a choice, hence the algorithm is non-deterministic. Next we determine whether this guess is indeed a communicating answer set, which involves taking the reduct, computing the fixpoint and verifying whether this fixpoint coincides with our guess. Clearly, verifying whether the interpretation is an answer set can be done in polynomial time and thus the algorithm to compute the answer sets of a communicating simple program is in NP, and thus NP-complete, regardless of the number of component programs. These same results hold for communicating normal programs since the reduct also removes all occurrences of negation-as-failure.

For communicating disjunctive programs it is easy to see that the Σ_2^P-completeness of classical disjunctive ASP carries over to communicating disjunctive programs. Cautious reasoning is then coNP and coΣ_2^P for communicating normal programs and communicating disjunctive programs, respectively, since this decision problem is the complement of brave reasoning. Finally, the problem of answer set existence is carried over from normal programs and disjunctive programs [Baral 2003] and is NP-hard and coΣ_2^P-hard, respectively. Most of these complexity results correspond with classical ASP, with the results from communicating simple programs being notable exceptions; indeed, for communicating simple programs the communication aspect clearly has an influence on the complexity. Table 3.1 summarises the main complexity results.

Table 3.1: Completeness results for the main reasoning tasks for a communicating program $\mathcal{P} = \{Q_1, ..., Q_n\}$

reasoning task $\rightarrow$ component programs $\downarrow$	answer set existence	brave reasoning	cautious reasoning
simple program	P	NP	coNP
normal program	NP	NP	coNP
disjunctive program	Σ_2^P	Σ_2^P	coΣ_2^P

3.4 Multi-Focused Answer Sets

Answer set semantics are intuitively based on the idea of minimal models. When dealing with agents that can communicate, it becomes unclear how we should interpret the notion of minimality. One option is to assume *global minimality*, i.e. we minimise over the conclusions of all the agents in the network. This is the approach that was taken in Section 3.3. Another option is to assume minimality on the level of a single agent, i.e. *local minimality*. Since it is not always possible to find a model that is minimal for all individual agents, the order in which we minimise over the agents matters, as the next example illustrates.

Example 25

An employee ('E') needs a new printer ('P'). She has a few choices (loud or silent, stylish or dull), preferring silent and stylish. Her manager ('M') insists that it is a silent printer. Her boss ('B') does not want an expensive printer, i.e. one that is both silent and stylish. We can consider the communicating normal program $\mathcal{P} = \{P, E, M, B\}$ with:

$$P:stylish \leftarrow not\ P:dull \qquad P:dull \leftarrow not\ P:stylish \tag{3.7}$$

$$P:silent \leftarrow not\ P:loud \qquad P:loud \leftarrow not\ P:silent \tag{3.8}$$

$$E:undesired \leftarrow P:dull \qquad E:undesired \leftarrow P:loud \tag{3.9}$$

$$M:undesired \leftarrow P:loud \tag{3.10}$$

$$B:expensive \leftarrow P:stylish, P:silent. \tag{3.11}$$

The rules in (3.7) and (3.8) encode the four possible printers and the rules in (3.9), (3.10) and (3.11) encode the inclinations of the employee, manager and boss, respectively. The answer sets of this program, i.e. those with global minimality, are

$$\begin{aligned}
M_1 &= \{P\!:\!stylish, P\!:\!silent, B\!:\!expensive\}\\
M_2 &= \{P\!:\!stylish, P\!:\!loud, E\!:\!undesired, M\!:\!undesired\}\\
M_3 &= \{P\!:\!dull, P\!:\!loud, E\!:\!undesired, M\!:\!undesired\}\\
M_4 &= \{P\!:\!dull, P\!:\!silent, E\!:\!undesired\}
\end{aligned}$$

The answer sets that are minimal for agent B are M_2, M_3 and M_4, i.e. the answer sets that do not contain $B\!:\!expensive$. The only answer set that is minimal for agent E is M_1, i.e. the one that does not contain $E\!:\!undesired$. Hence when we determine local minimality for CASP, the order in which we consider the agents is important as it induces a priority over them, i.e. it makes some agents more important than others. In this example, if the boss comes first, the employee no longer has the choice to pick M_1. This leaves her with the choice of either a dull or a loud printer, among which she has no preferences. Since the manager prefers a silent printer, when we first minimise over 'B' and then minimise over 'M' and 'E' (we may as well minimise over 'E' and then 'M', as 'E' and 'M' have no conflicting preferences) we end up with the unique answer set M_4.

In this section, we formalise such a communication mechanism. We extend the semantics of communicating programs in such a way that it becomes possible to focus on a sequence of component programs. As such, we can indicate that we are only interested in those answer sets that are successively minimal with respect to each respective component program. The underlying intuition is that of leaders and followers, where the decisions that an agent can make are limited by what its leaders have previously decided.

Definition 12

Let $\mathcal{P}$ be a communicating normal program and $\{Q_1, ..., Q_n\} \subseteq \mathcal{P}$ a set of component programs. A $(Q_1, ..., Q_n)$-*focused answer set* of $\mathcal{P}$ is defined recursively as follows:

- M is a $(Q_1, ..., Q_n)$-focused answer set of $\mathcal{P}$ and there are no $(Q_1, ..., Q_{n-1})$-focused answer sets M' of $\mathcal{P}$ such that $M'_{Q_n} \subset M_{Q_n}$;
- a ()-focused answer set of $\mathcal{P}$ is any answer set of $\mathcal{P}$.

In other words, we say that M is a $(Q_1, ..., Q_n)$-focused answer set of $\mathcal{P}$ if and only if M is minimal among all $(Q_1, ..., Q_{n-1})$-focused answer sets w.r.t. the projection on Q_n.

Example 26

Consider the communicating normal program $\mathcal{P} = \{Q, R, S\}$ with the rules

$$\begin{array}{lll}
Q\!:\!a \leftarrow & R\!:\!b \leftarrow S\!:\!c & S\!:\!a \leftarrow \\
Q\!:\!b \leftarrow not\ S\!:\!d & R\!:\!a \leftarrow S\!:\!c & S\!:\!c \leftarrow not\ S\!:\!d, not\ R\!:\!c \\
Q\!:\!c \leftarrow R\!:\!c & R\!:\!a \leftarrow S\!:\!d & S\!:\!d \leftarrow not\ S\!:\!c, not\ R\!:\!c \\
 & R\!:\!c \leftarrow not\ R\!:\!a &
\end{array}$$

The communicating normal program $\mathcal{P}$ has three answer sets, namely

$$\begin{aligned}
M_1 &= Q\!:\!\{a, b, c\} \cup R\!:\!\{c\} \cup S\!:\!\{a\} \\
M_2 &= Q\!:\!\{a, b\} \cup R\!:\!\{a, b\} \cup S\!:\!\{a, c\} \\
M_3 &= Q\!:\!\{a\} \cup R\!:\!\{a\} \cup S\!:\!\{a, d\}.
\end{aligned}$$

The only (R, S)-focused answer set of $\mathcal{P}_{26}$ is M_1. Indeed, since $\{a\} = (M_3)_R \subset (M_2)_R = \{a, b\}$ we find that M_2 is not a (R)-focused answer set. Furthermore $\{a\} = (M_1)_S \subset (M_3)_S = \{a, d\}$, hence M_3 is not an (R, S)-focused answer set.

Proposition 9

Let $\mathcal{P}$ be a communicating simple program. We then have:

- there always exists at least one $(Q_1, ..., Q_n)$-focused answer set of $\mathcal{P}$;
- we can compute this $(Q_1, ..., Q_n)$-focused answer set in polynomial time.

To investigate the computational complexity of multi-focused answer sets we now show how the validity of QBF can be checked using multi-focused answer sets of communicating ASP programs.

Definition 13

Let $\phi = \exists X_1 \forall X_2 \cdots \Theta X_n \cdot p(X_1, X_2, \cdots X_n)$ be a QBF where $\Theta = \forall$ if n is even and $\Theta = \exists$ otherwise, and $p(X_1, X_2, \cdots X_n)$ is a formula of the form $\theta_1 \vee ... \vee \theta_m$ in disjunctive normal form over $X_1 \cup ... \cup X_n$ with X_i, $1 \leq i \leq n$, sets of variables and where each θ_t is a conjunction of propositional literals. We define Q_0 as follows:

$$\begin{aligned} Q_0 = &\{x \leftarrow not\ \neg x, \neg x \leftarrow not\ x \mid x \in X_1 \cup ... \cup X_n\} && (3.12)\\ &\cup \{sat \leftarrow Q_0 \!:\! \theta_t \mid 1 \leq t \leq m\} && (3.13)\\ &\cup \{\neg sat \leftarrow not\ sat\}\,. && (3.14)\end{aligned}$$

For $1 \leq j \leq n-1$ we define Q_j as follows:

$$\begin{aligned} Q_j = &\{x \leftarrow Q_0 \!:\! x, \neg x \leftarrow Q_0 \!:\! \neg x \mid x \in (X_1 \cup ... \cup X_{n-j})\} && (3.15)\\ &\cup \begin{cases} \{\neg sat \leftarrow Q_0 \!:\! \neg sat\} & \text{if } (n-j) \text{ is even}\\ \{sat \leftarrow Q_0 \!:\! sat\} & \text{if } (n-j) \text{ is odd.}\end{cases} && (3.16)\end{aligned}$$

The communicating normal program corresponding with ϕ is $\mathcal{P}_\phi = \{Q_0, ..., Q_{n-1}\}$.

For a QBF of the form $\phi = \forall X_1 \exists X_2 \cdots \Theta X_n \cdot p(X_1, X_2, \cdots X_n)$ where $\Theta = \exists$ if n is even and $\Theta = \forall$ otherwise and $p(X_1, X_2, \cdots X_n)$ once again a formula in disjunctive normal form, the simulation only changes slightly. Indeed, only the conditions in (3.16) are swapped.

Example 27

Given the QBF $\phi = \exists x \forall y \exists z \cdot (x \wedge y) \vee (\neg x \wedge y \wedge z) \vee (\neg x \wedge \neg y \wedge \neg z)$, the communicating normal program $\mathcal{P}$ corresponding with the QBF ϕ is defined as follows:

$$\begin{array}{ccc} Q_0 \!:\! x \leftarrow not\ \neg x & Q_0 \!:\! y \leftarrow not\ \neg y & Q_0 \!:\! z \leftarrow not\ \neg z\\ Q_0 \!:\! \neg x \leftarrow not\ x & Q_0 \!:\! \neg y \leftarrow not\ y & Q_0 \!:\! \neg z \leftarrow not\ z\\ Q_0 \!:\! sat \leftarrow x, y & Q_0 \!:\! sat \leftarrow \neg x, y, z & Q_0 \!:\! sat \leftarrow \neg x, \neg y, \neg z\\ Q_0 \!:\! \neg sat \leftarrow not\ sat & & \\ & & \\ Q_1 \!:\! x \leftarrow Q_0 \!:\! x & Q_1 \!:\! y \leftarrow Q_0 \!:\! y & \\ Q_1 \!:\! \neg x \leftarrow Q_0 \!:\! \neg x & Q_1 \!:\! \neg y \leftarrow Q_0 \!:\! \neg y & Q_1 \!:\! \neg sat \leftarrow Q_0 \!:\! \neg sat\\ & & \\ Q_2 \!:\! x \leftarrow Q_0 \!:\! x & Q_2 \!:\! \neg x \leftarrow Q_0 \!:\! \neg x & Q_2 \!:\! sat \leftarrow Q_0 \!:\! sat \end{array}$$

The communicating normal program in Example 27 can be used to determine whether the QBF ϕ is valid. First, note that the rules in (3.12) generate all possible truth assignments of the variables, i.e. all possible propositional interpretations. The rules in (3.13) ensure that 'sat' is true exactly for those interpretations that satisfy the formula $p(X_1, X_2, ..., X_n)$.

Intuitively, the component programs $\{Q_1, ..., Q_{n-1}\}$ successively bind fewer and fewer variables. In particular, focussing on $Q_1, ..., Q_{n-1}$ allows us to consider the binding of the variables in $X_{n-1}, ..., X_1$, respectively. Depending on the rules from (3.16), focussing on Q_i allows us to verify that either some or all of the assignments of the variables in X_{n-j} make the formula $p(X_1, ..., X_n)$ satisfied, given the bindings that have already been determined by the preceding components. We now prove that the QBF ϕ is satisfiable iff $Q_0\!:\!sat$ is true in some $(Q_1, ..., Q_{n-1})$-focused answer set of the corresponding program.

Proposition 10

Let ϕ and $\mathcal{P}$ be as in Definition 13. We have that a QBF ϕ of the form $\phi = \exists X_1 \forall X_2 \cdots \Theta X_n \cdot p(X_1, X_2, \cdots X_n)$ is satisfiable if and only if $Q_0\!:\!sat$ is true in some $(Q_1, ..., Q_{n-1})$-focused answer set of $\mathcal{P}$. Furthermore, we have that a QBF ϕ of the form $\phi = \forall X_1 \exists X_2 \cdots \Theta X_n \cdot p(X_1, X_2, \cdots X_n)$ is satisfiable if and only if $Q_0\!:\!sat$ is true in all $(Q_1, ..., Q_{n-1})$-focused answer sets of $\mathcal{P}$.

Corollary 1

Let $\mathcal{P}$ be a communicating normal program with $Q_i \in \mathcal{P}$. The problem of deciding whether there exists a $(Q_1, ..., Q_n)$-focused answer set M of $\mathcal{P}$ such that $Q_i\!:\!l \in M$ (brave reasoning) is $\Sigma^{\mathsf{P}}_{n+1}$-hard.

Corollary 2

Let $\mathcal{P}$ be a communicating normal program with $Q_i \in \mathcal{P}$. The problem of deciding whether all $(Q_1, ..., Q_n)$-focused answer sets contain $Q_i\!:\!l$ (cautious reasoning) is Π^{P}_{n+1}-hard.

In addition to these hardness results, we can also establish the corresponding membership results.

Proposition 11

Let $\mathcal{P}$ be a communicating normal program with $Q_i \in \mathcal{P}$. The problem of deciding whether there exists a $(Q_1, ..., Q_n)$-focused answer set M of $\mathcal{P}$ such that $Q_i\!:\!l \in M$ (brave reasoning) is in $\Sigma^{\mathsf{P}}_{n+1}$.

Since cautious reasoning is the complementary problem of brave reasoning it readily follows that cautious reasoning is in co$\Sigma^{\mathsf{P}}_{n+1}$. Now that we have both hardness and membership results, we readily obtain the following corollary.

Corollary 3

Let $\mathcal{P}$ be a communicating normal program with $Q_i \in \mathcal{P}$. The problem of deciding whether $Q_i\!:\!l \in M$ with M a $(Q_1, ..., Q_n)$-focused answer set of $\mathcal{P}$ is $\Sigma^{\mathsf{P}}_{n+1}$-complete.

The next proposition shows that the complexity remains the same when going from normal component programs to simple component programs.

Proposition 12

Let $\mathcal{P}$ be a communicating simple program with $Q_i \in \mathcal{P}$. The problem of deciding whether there exists a $(Q_1, ..., Q_n)$-focused answer set M of $\mathcal{P}$ such that $Q_i\!:\!l \in M$ (brave reasoning) is in $\Sigma^{\mathsf{P}}_{n+1}$.

Finally, we also have a result for communicating disjunctive programs instead of communicating normal programs.

Proposition 13

Let $\mathcal{P}$ be a communicating disjunctive program with $Q_i \in \mathcal{P}$. The problem of deciding whether $Q_i\!:\!l \in M$ with M a $(Q_1, ..., Q_n)$-focused answer set of $\mathcal{P}$ is in $\Sigma^{\mathsf{P}}_{n+2}$.

Table 3.2 summarises the membership results for brave reasoning that were discussed in this section.

Table 3.2: Membership results for brave reasoning with (multi-focused) answer sets of the communicating program $\mathcal{P} = \{Q_1, ..., Q_n\}$

form of communication $\rightarrow$	none	situated literals	multi-focused
type of component program $\downarrow$			
simple program	P	NP	$\Sigma^{\mathsf{P}}_{n+1}$
normal program	NP	NP	$\Sigma^{\mathsf{P}}_{n+1}$
disjunctive program	Σ^{P}_2	Σ^{P}_2	$\Sigma^{\mathsf{P}}_{n+2}$

3.5 Case Study: subset-minimal abductive diagnosis

In this section we work out an example that highlights the usefulness of multi-focused answer sets. Although a lot of interesting problems are indeed in P, NP or Σ_2^{P}, there are still some important problems that are located higher up in the polynomial hierarchy. One such a problem is a special form of abductive diagnostics. An abductive diagnostic problem is encoded as a triple $\langle H, T, O \rangle$ [Eiter et al. 1997], where H is a set of atoms referred to as hypotheses, T is an ASP program referred to as the theory and O is a set of literals referred to as observations. Intuitively, the theory T describes the dynamics of the system, the observations O describe the observed state of the system and the hypotheses H try to explain these observations within the theory. The goal in subset-minimal abductive diagnosis is to find the minimal set of hypotheses that explain the observation. That is, we want to find the minimal set of hypotheses such that $O \subseteq M$ with M an answer set of $T \cup H$. Subset-minimal abductive diagnostics over a theory consisting of a disjunctive program is a problem in Σ_3^{P} and hence we cannot directly rely on classical ASP to find the solutions to this problem. However, as we will see in the next example, we can easily solve this problem using multi-focused answer sets.

Example 28: Adapted from [Eiter et al. 1999]

Consider an electronic circuit, as in Figure 3.2, where we have a power source, a control lamp, three hot-plates wired in parallel and a fuse to protect each hot-plate. It is known that some of the fuses are sensitive to high current and may consequently blow, but it is not known which fuses. Furthermore, plate A sits near a source of water (e.g. a tap). If water comes into contact with plate A, this causes a short circuit which blows the nearest fuse, i.e. fuse A, to prevent any damage.

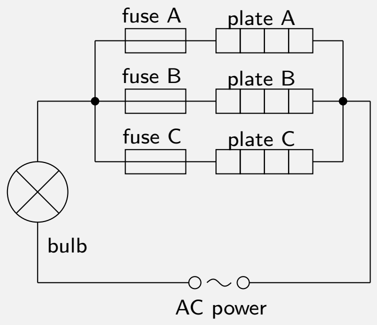


Figure 3.2: Schematics of the electronic circuit we want to diagnose.

Upon inspection, we find that the control lamp is on and that plate A feels cold to the touch. We want to find the subset minimal diagnoses that would explain the problem, i.e. we want to find the minimal causes that can explain this situation.

First we need to describe the theory, i.e. the schematics. The theory describes the dynamics of the system and thus also how the system may fail. We can describe the theory as follows. For starters, a melted fuse can be caused by a high current, or, for fuse A, due to a hazardous water leak:

$$\begin{aligned} Q\!:\!melted\_A; Q\!:\!melted\_B; Q\!:\!melted\_C &\leftarrow Q\!:\!high \\ Q\!:\!melted\_A &\leftarrow Q\!:\!leak. \end{aligned}$$

Furthermore, under a number of conditions the control light will be off:

$$\begin{aligned} Q\!:\!light\_off &\leftarrow Q\!:\!power\_off \\ Q\!:\!light\_off &\leftarrow Q\!:\!broken\_bulb \\ Q\!:\!light\_off &\leftarrow Q\!:\!melted\_A, Q\!:\!melted\_B, Q\!:\!melted\_C. \end{aligned}$$

Then we describe under what conditions each plate will be hot:

$$\begin{aligned} Q\!:\!hot\_plateA &\leftarrow not\ Q\!:\!melted\_A, not\ Q\!:\!power\_off \\ Q\!:\!hot\_plateB &\leftarrow not\ Q\!:\!melted\_B, not\ Q\!:\!power\_off \\ Q\!:\!hot\_plateC &\leftarrow not\ Q\!:\!melted\_C, not\ Q\!:\!power\_off. \end{aligned}$$

We now encode the hypotheses. We have a number of causes, each of which may by itself or in conjunction with other causes explain our observation. In total, we have four causes. The power can be off ($power\_off$), the light bulb might be broken ($broken\_bulb$), there may have been a high current ($high$) and/or a water leak may have occurred ($leak$). We describe all these hypotheses as follows:

$$\begin{aligned} Q\!:\!power\_off &\leftarrow not\ Q\!:\!no\_power\_off \\ Q\!:\!no\_power\_off &\leftarrow not\ Q\!:\!power\_off \\ Q\!:\!broken\_bulb &\leftarrow not\ Q\!:\!no\_broken\_bulb \\ Q\!:\!no\_broken\_bulb &\leftarrow not\ Q\!:\!broken\_bulb \\ Q\!:\!high &\leftarrow not\ Q\!:\!no\_high \\ Q\!:\!no\_high &\leftarrow not\ Q\!:\!high \\ Q\!:\!leak &\leftarrow not\ Q\!:\!no\_leak \\ Q\!:\!no\_leak &\leftarrow not\ Q\!:\!leak. \end{aligned}$$

It is easy to see that these rules in Q encode all possible subsets of hypotheses that may have occurred. We then add rules to a separate component program H which merely relays the information on the set of hypotheses that we chose. The reason for this separate component program H is that we can now minimise over the set of hypotheses that is assumed, simply by focussing on H.

$$\begin{aligned} H\!:\!\mathit{power\_off} &\leftarrow Q\!:\!\mathit{power\_off} \\ H\!:\!\mathit{broken\_bulb} &\leftarrow Q\!:\!\mathit{broken\_bulb} \\ H\!:\!\mathit{high\_current} &\leftarrow Q\!:\!\mathit{high} \\ H\!:\!\mathit{water\_leak} &\leftarrow Q\!:\!\mathit{leak} \end{aligned}$$

Finally, we model the observation. We observe that the control light is on and that plate A is cold. In other words, we obtain the rules (which encode constraints):

$$\begin{aligned} Q\!:\!\mathit{contradiction} &\leftarrow \mathit{not}\ Q\!:\!\mathit{contradiction}, Q\!:\!\mathit{light\_off} \\ Q\!:\!\mathit{contradiction} &\leftarrow \mathit{not}\ Q\!:\!\mathit{contradiction}, Q\!:\!\mathit{hot\_plateA} \end{aligned}$$

which intuitively tell us that we cannot have that the light is off, nor can we have that plate A is hot.

The (H)-focused answer sets give us the subset minimal abductive diagnoses. It is easy to see that the focus on H is needed to minimise over the hypotheses. The program $\mathcal{P} = \{Q, H\}$ has two (H)-focused answer sets M_1 and M_2, both containing $M_{\text{shared}} = \{Q\!:\!\mathit{no\_power\_off}, Q\!:\!\mathit{no\_broken\_bulb}, Q\!:\!\mathit{hot\_plateB}, Q\!:\!\mathit{hot\_plateC}\}$:

$$\begin{aligned} M_1 &= M_{\text{shared}} \cup \{Q\!:\!\mathit{melted\_A}, Q\!:\!\mathit{no\_leak}, Q\!:\!\mathit{high}, H\!:\!\mathit{high\_current}\} \\ M_2 &= M_{\text{shared}} \cup \{Q\!:\!\mathit{melted\_A}, Q\!:\!\mathit{leak}, Q\!:\!\mathit{no\_high}, H\!:\!\mathit{water\_leak}\}\,. \end{aligned}$$

Hence the minimal sets of hypotheses that support our observation, i.e. M_H with M an (H)-focused answer set, are that either there was a high current (which melted fuse A) or there was a water leak (which also melted fuse A).

3.6 Work Related to CASP

Important work has been done in the domain of MCSs and multi-agent ASP to enable collaboration between different contexts/ASP programs. We discuss some of the more prominent work in these areas in this section. The work of [Roelofsen and Serafini 2005] proposes an extension of MCSs [Giunchiglia and Serafini 1994] that allows MCSs to reason about absent information, i.e. they introduce non-monotonicity in the context of MCSs.

The idea of a MCS, as we have seen in the introduction of this chapter, is that we have a number of contexts that each have access to only a subset of the available information. Each context has a local model and reasoning capabilities, but there is also an information flow defined by the system between the different contexts. It is this idea that was later adopted in the ASP community and in this chapter in particular.

Our work has a comparable syntax as [Roelofsen and Serafini 2005] but rather different semantics. The semantics in [Roelofsen and Serafini 2005] are closely related to the well-founded semantics [Gelder et al. 1991], while our semantics are closer in spirit to the stable model semantics [Gelfond and Lifschitz 1988]. Another point where our semantics differ is that we allow a restricted circular explanation of why a literal is true, if that circular explanation is due to our reliance on other component programs. This particular form of circular reasoning has been identified in [Buccafurri et al. 2008] as a requirement in the representation of social reasoning.

The work in [Brewka et al. 2007] extends upon the work in [Roelofsen and Serafini 2005] and addresses a number of problems and deficiencies. The paper is, to the best of our knowledge, the first to offer a syntactical rather than semantical description of communication in multi-context systems, making it easier to implement an actual algorithm. A number of interesting applications of contextual frameworks, including information fusion, game theory and social choice theory are highlighted in the paper. Lastly, the paper identifies that the complexity of the main reasoning task is on the second level of the polynomial hierarchy.

Along similar lines the work in [Brewka and Eiter 2007] combines the non-monotonicity from [Roelofsen and Serafini 2005] with the heterogeneous approach which was presented in [Giunchiglia and Serafini 1994] into a single framework for heterogeneous non-monotonic multi-context reasoning. The work in [Brewka and Eiter 2007] introduces several notions of equilibria, including minimal and grounded equilibria. In our approach, local reasoning is captured by grounded equilibria (which does not allow circular explanations) while communicating with other component programs is captured by the weaker minimal equilibria. The work in [Brewka and Eiter 2007] offers various membership results on deciding the existence of an equilibrium and is one of the first to note that multi-context systems, due to the nature of the bridge rules/situated literals, can be non-monotonic even if all the logics in the component programs themselves are monotonic.

An initial implementation of a distributed solver for heterogeneous multi-context systems was first discussed in [Dao-Tran et al. 2010]. While solvers exist to compute multi-context systems locally, this is the first work to consider an algorithm which is both distributed (i.e. no shared memory) and modular (i.e. computation starting from partial models). When the context under consideration uses e.g. ASP, loop formulas can be devised which allow bridge rules to be compiled into local classical theories. It is then possible to use

SAT solvers to compute the grounded equilibria of the heterogeneous multi-context system. Later work in [Drescher et al. 2011] improved on the idea by offering a mechanism to identify and break symmetries (i.e. permutations of belief which result in identical knowledge). As such, the solver need never visit two points in the search space that are symmetric, thus potentially offering a considerable speedup. Experimental results show that the solution space can indeed be (significantly) compressed. A similar idea might be used to compute answer sets of a communicating ASP program in a distributed fashion. Indeed, such answer sets are closely related to the idea of minimal equilibria from [Brewka and Eiter 2007]. A few modifications should nonetheless be made. For example, the Herbrand base needs to be redefined in a way that is safe in such a distributed setting, e.g. by only taking situated literals into account that occur in a given component program. Optimizations to the distributed algorithm also seem likely to be applicable to the setting of CASP. On the other hand, it does not seem to be straightforward to extend these ideas to compute multi-focused answer sets in a distributed fashion.

One of the most recent extensions to multi-context systems are managed multi-context system (mMCS) [Brewka et al. 2011]. Normally, bridge rules can only be used to pass along information which allows for e.g. selection and abstraction of information between contexts. In an mMCS, however, additional operations on knowledge bases can be freely defined. For example, operations may be defined that remove or revise information. Such operations are performed by the context itself, i.e. by the legacy system that is used such as ASP, but mMCS allow to cope with this additional functionality in a principled way. As one would expect, adding such complex operations increases the expressiveness of the resulting system considerably. Our work, on the other hand, only allows for information to be passed along. By varying the way that the communication works, we achieved a comparable expressiveness.

We now direct our attention to work done within the ASP community. The ideas presented in this chapter are related to HEX programs [Eiter et al. 2005] in which ASP is extended with higher-order predicates and external atoms. These external atoms allow to exchange knowledge in a declarative way with external sources that may implement functionality which is inconvenient or impossible to encode using current answer set programming paradigms. Application-wise, HEX is mainly proposed as a tool for non-monotonic semantic web reasoning under the answer set semantics. Hence HEX is not primarily targeted at increasing the expressiveness, but foremost at extending the applicability and ease of use of ASP.

In [De Vos et al. 2005] a multi-agent framework called LAIMA is developed, similar as in [Van Nieuwenborgh et al. 2007], in which multiple agents/component programs can communicate with each other. It allows to represent and model knowledge obtained from different contexts. Each of these contexts is represented as an OCLP or OCLP or

Ordered Choice Logic Program [Brain and De Vos 2003], which is an expressive variant of classical ASP. IN OCLP, negation-as-failure is not modelled explicitly but with preferences. Contrary to [Van Nieuwenborgh et al. 2007], agents can communicate with whoever they want and circular communication is allowed (where agent A tells something to agent B which tells something to A ...). However, only positive information can be shared and the authors do not examine the expressiveness of the LAIMA framework.

We also mention [Dao-Tran et al. 2009] where recursive modular non-monotonic logic programs (MLP) under the ASP semantics are considered. The main difference between MLP and our work is that our communication mechanism is parameter-less, i.e. the truth of a situated literal is not dependent on parameters passed by the situated literal to the target component program. Our approach is clearly different and we cannot readily mimic the behaviour of the networks presented in [Dao-Tran et al. 2009]. Our expressiveness results therefore do not directly apply to MLPs.

Finally, there is an interesting resemblance between multi-focused answer sets and the work on multi-level integer programming [Jeroslow 1985]. In multi-level integer programming, different agents control different variables that are outside of the control of the other agents, yet are linked by means of linear inequalities (constraints). The agents have to fix the values of the variables they can control in a predefined order, such that their own linear objective function is optimized. Similarly, in CASP, literals belong to different component programs (agents), and their values are linked through constraints, which in this case take the form of rules. Again the agents act in a predefined order, but now they try to minimise the set of literals they have to accept as being true, rather than a linear objective function. Although there is an intuitive link, further research is required to make this link between multi-focused answer sets and the work on multi-level integer programming explicit.

3.7 Summary

In Chapter 1 we emphasized how we wanted to look at extensions of ASP for epistemic reasoning. ASP can, by itself, be used to model the knowledge of a single agent. However, ASP lacks the means to allow for communication, i.e. ASP lacks the means to model the knowledge of a network of interacting agents. To this end, we introduced CASP, which combines ASP with mechanisms for communication in a way that is similar to the approach taken in a MCS.

Combining the expressive power of multiple ASP programs to solve a complex problem is not a new idea, but current approaches start from expressive and non-standard forms of ASP or consider involved communication mechanisms. As such, while often demonstrating a higher overall expressivity, these approaches do not fully examine the necessary contributions needed to arrive at such a higher expressivity.

In this chapter we have systematically studied the effect of adding communication to ASP in terms of expressiveness and computational complexity. We started from simple programs, i.e. definite programs extended with true negation. Determining whether a literal belongs to an answer set of a simple program is a problem in P. We extended these simple programs by means of a straightforward construct that allows one program to ask questions to another program. A network of these simple programs, which we called communicating simple programs, is expressive enough to simulate normal programs as we have shown in Proposition 8. In other words, determining whether a literal belongs to an answer set of a communicating simple program is NP-hard. Importantly, this added expressiveness is directly related to the use of classical negation, which forms an essential part in our simulation. Furthermore, communicating simple programs can also simulate communicating normal programs provided that the resulting answer sets are partially complete, thus showing that adding negation-as-failure to communicating simple programs does not further increase the expressiveness. Nevertheless, the addition of negation-as-failure is desirable, as it can often provide an easy and intuitive way to model complex problems. We furthermore show in Section 3.3 how it is possible to simulate CASP with ASP. This will provide us in Chapter 6 with an easy and performant implementation of CASP.

In addition, we introduced multi-focused answer sets for communicating programs in Section 3.4. The underlying intuition is that of leaders and followers, where the choices available to the followers are limited by what the leaders have previously decided. On a technical level, the problem translates to establishing local minimality for some of the component programs in the communicating program, rather than global minimality on the level of a communicating program. In general, however, it is not possible to ensure local minimality for all component programs. Thus an order must be defined among component programs on which to focus. The resulting framework demonstrates an increase in expressiveness, where the problem of deciding whether $Q_i : l \in M$ with M a $(Q_1, ..., Q_n)$-focused answer set of a communicating normal program $\mathcal{P}$ is $\Sigma^{\mathsf{P}}_{n+1}$-complete, as shown in Proposition 10. We showed in Section 3.5 how this framework can be used to model complex problems (that cannot be expressed in a classical disjunctive ASP program) in an intuitive way.

Throughout this chapter we purposefully only considered a simple form of communication, i.e. a mechanism where programs can ask questions to each other. Most of the approaches that combine ASP with communication mechanisms use such a simple form of communication, making it easier to generalise the complexity results obtained in this chapter to these other approaches. Nevertheless, it will be interesting to analyse how more complex communication mechanisms affect the overall complexity. For example, in a blackboard architecture [Erman et al. 1980, Pang 1991] a shared data region is available which can be iteratively updated by the agents involved in the computation. Each of these agents can update the blackboard whenever the information available on the blackboard allows them to derive new conclusions. In addition, a control mechanism is defined that mediates between the different agents and determines when an agent is allowed to modify the information on the blackboard. The resulting architecture makes it possible for agents to collaborate to solve complex problems. However, new problems arise with the use of more complex communication mechanisms. For example, in the blackboard architecture the problem arises of how to deal with inconsistent information on the shared data region. Much work has been done on inconsistency handling, and data fusion in general. Possibility theory, as described in Section 1.3, can be used for data merging. For example, in e.g. [Benferhat et al. 2000], an approach is presented for merging prioritized knowledge bases, where the priorities are represented using possibilistic logic. Paraconsistent logics [Bremer 2005] are another family of logic systems that are able to handle inconsistent information and that can be used for data fusion. Merging methods based on ASP have also been proposed in the literature. For example, in[Delgrande et al. 2009] two different merging techniques are proposed based on ASP, where the merging operators do not affect the complexity of the base formalism. When dealing with inaccurate information rather than completely incorrect information, approaches such as [Schockaert and Prade 2011] can be used. In this approach statements are interpreted in a more flexible way, rather than merely ignoring the information, in an attempt to resolve the inconsistencies. A more elaborate overview of various techniques for data merging can be found in e.g. [Konieczny and Pérez 2011]. Clearly, however, extending the CASP framework to more elaborate forms of communication is the subject of future work.

To conclude, we have considered an epistemic extension of ASP where, by adding means of communication, we are able to use ASP to model a network of contexts where the contexts can share information with one another. We find that the choice of the communication mechanism is paramount w.r.t. the expressiveness of the overall system, in addition to the expressiveness of the individual agents. An overview of our results is presented in Table 3.2, which highlights the membership results for brave reasoning obtained in Section 3.4.

4 | Characterizing and extending ASP using possibility theory

4.1 Introduction

In the previous chapter we looked at an extension of ASP that allows for communication between a network of ASP programs. We discussed how different mechanisms of communication affect the complexity in different ways. We also illustrated how CASP, and multi-focused answer sets in particular, can be used to solve problems that cannot be modelled in classical ASP. However, we did not look at how an individual program can reason about uncertain information. Still, when we want to model what an agent knows or what an agent believes, uncertainty plays an important role.

The Possibilistic Answer Set Programming (PASP) extension of ASP, which allows us to deal with uncertainty, already exists and was discussed in Section 2.5 of Chapter 2. We can consider PASP to be a family of approaches that share a common syntax and that have semantics based on possibility distributions. Syntactically, a weight λ is associated with a rule $(head \leftarrow body)$. In PASP$_\mathsf{G}$ we can derive the $head$ with certainty $\min(\lambda, N(body))$ for possibilistic simple rules, i.e. the certainty of $head$ is restricted by the least certain piece of information in the derivation chain. The semantics for PASP which we present in this chapter, which we refer to as PASP$_\text{Ł}$, treat the weight in the same way when considering possibilistic simple rules. However, there will be a notable difference in how negation-as-failure is treated. Recall that, when faced with negation-as-failure, in PASP$_G$ this means that the weights associated with the rules are initially ignored, the classical reduct is determined and the weights are then reassociated with the corresponding rules in the reduct. Given this particular treatment of negation-as-failure, the underlying intuition of '$not\ l$' is "*l cannot be derived with a strictly positive certainty*". Indeed, as soon as 'l' can be derived with a certainty $\lambda > 0$, 'l' is treated as true when determining the reduct. However, this particular understanding of negation-as-failure is not always the most intuitive one.

Consider the following example. You want to go to the airport, but you notice that your passport will expire in less than three months. Some countries require that the passport is at least valid for an additional three months on the date of entry. As such, you have some certainty that your passport might be invalid ($invalid$). When you are not entirely certain that your passport is invalid, it is still useful to go the airport ($airport$) and check-in. Indeed, since you are not absolutely certain that you will not be allowed to board, you might still get lucky. We have the possibilistic program:

$$\begin{aligned} \mathbf{0.1}\colon & invalid \leftarrow \\ \mathbf{1}\colon & airport \leftarrow not\ invalid \end{aligned}$$

where 0.1 and 1 are the weights associated with the rules ($invalid \leftarrow$) and $airport \leftarrow invalid$, respectively. Clearly, what we would like to be able to conclude with a high certainty is that you need to go to the airport to check-in. However, as the semantics from [Nicolas et al. 2006] adhere to a different intuition of negation-as-failure, the conclusion is that you need to go to the airport with a necessity of 0. Or, in other words, you should not go to the airport at all.

As a first contribution in this chapter, we present new semantics for PASP by interpreting possibilistic rules as constraints on possibility distributions. We will refer to these semantics as PASP$_Ł$. The answers sets from PASP$_Ł$ do not, in general, correspond with the semantics from PASP$_G$ when considering programs with negation-as-failure. Specifically, the semantics presented in this chapter can be used in settings in which the possibilistic answer sets according to PASP$_G$ do not correspond with the intuitively acceptable results. For the example mentioned above, the conclusion under the new semantics will be that you need to go to the airport with a necessity of 0.9.

In addition, the new semantics that we present in this chapter allow us to uncover a new characterization of classical ASP in terms of possibility theory. Over the years, many equivalent approaches have been proposed to define the notion of an answer set. One of the most popular characterizations is in terms of a reduct [Gelfond and Lifschitz 1988] in which an answer set is guessed and verified to be stable. This characterization is used in PASP$_\mathsf{G}$. Alternatively, the answer set semantics of normal programs can be defined in terms of autoepistemic logic [Marek and Truszczyński 1991], a well-known non-monotonic modal logic which we briefly mentioned in Section 1.2. An important advantage of the latter approach is that autoepistemic logic enjoys more syntactic freedom, which opens the door to more expressive forms of logic programming. However, as has been shown early on in [Lifschitz and Schwarz 1993], the characterization in terms of autoepistemic logic does not allow us to treat classical negation or disjunctive rules in a natural way, which weakens its position as a candidate for generalizing ASP from normal programs to e.g. disjunctive programs. Equilibrium logic [Pearce 1997] offers yet another way for characterizing and extending ASP, but does not feature modalities which limits its potential for epistemic reasoning as it does not allow us to reason over the established knowledge of an agent. The new characterization of ASP, as presented in this chapter, is a characterization in terms of necessary and contingent truths, where possibility theory is used to express our certainty in logical propositions. Such a characterization is unearthed by looking at ASP as a special case of PASP in which the rules are certain. It highlights the intuition of ASP that the head of a rule is certain when the information encoded in its body is certain. Furthermore, this characterization stays close to the intuition of the Gelfond-Lifschitz reduct, while sharing the explicit reference to modalities with autoepistemic logic.

As a second contribution, we show in this chapter how this new characterization of ASP in terms of possibility theory can be used to uncover a new form of disjunction in both ASP and PASP. As indicated, we have that the new semantics offer us an explicit reference to modalities, i.e. operators with which we can qualify a statement. Epistemic logic is an example of a modal logic in which we use the modal operator K to reason about knowledge, where K is intuitively understood as "*we know that*", as mentioned in Section 1.2. A statement such as $a \vee b \vee c$ can then be treated in two distinct ways. On the one hand, we can interpret this statement as $Ka \vee Kb \vee Kc$, which makes it explicit that we know which one of the disjuncts is true. This treatment corresponds with the understanding of disjunction in disjunctive ASP and will be referred to as *strong disjunction*. Alternatively, we can interpret $a \vee b \vee c$ as $K(a \vee b \vee c)$ which only states that we know that the disjunction is true, i.e. we do not know which of the disjuncts is true. We will refer to this form of disjunction as *weak disjunction*. This is the new form of disjunction that we will discuss in Section 4.3.2, as it allows us to reason in settings where a choice cannot or should not be made. Still, such a framework allows for non-trivial forms of reasoning, as we will see in e.g. Example 36. This is also apparent when we study the complexity of weak disjunction. In particular, we show that while most complexity results coincide with the strong disjunctive semantics, the complexity of brave reasoning (deciding whether a literal 'l' is entailed by a consistent answer set of program P) in absence of negation-as-failure is lower for weak disjunction. Still, the expressiveness is higher than for normal programs. The complexity results are summarized in Table 4.1 in Section 4.4.

The structure of this chapter is as follows. In Section 4.2 we introduce new semantics for PASP, referred to as PASP$_Ł$, based on constraints on possibility distributions. These new semantics agree with PASP$_G$ for simple programs. However, they adhere to a different intuition for negation-as-failure. Specifically, in PASP$_Ł$ we have that '$not\ l$' is understood as "*the degree to which '$\neg l$' is possible*". Classical ASP can furthermore be seen as a special case of PASP$_Ł$. Indeed, in classical ASP we consider rules that are absolutely certain and we do not allow for uncertainty in the answer sets. We show that under these conditions PASP$_Ł$ can be used to characterize ASP.

In Section 4.3 we highlight that the treatment of a rule as a constraint on possibility distributions allows for two ways to treat disjunction. One of these can be used to characterizing disjunctive ASP programs. The other form of disjunction, called weak disjunction, does not enforce a choice and is treated propositionally. This allows us to reason in settings where a choice cannot or should not be made. In Section 4.4 we investigate the complexity of PASP$_Ł$ and, specifically, of weak disjunction. We find that for possibilistic normal programs and possibilistic disjunctive programs the complexity coincides with the classical case. For weak disjunction, we find that most complexity results coincide with strong disjunction. However, interestingly, for the problem of brave reasoning without negation-as-failure and, crucially, with classical negation, the complexity results are between normal programs and programs with strong disjunction. These results show that weak disjunction is non-trivial, as it is more expressive than normal programs while being less complex than disjunctive programs under the strong semantics.

4.2 Characterizing (P)ASP

ASP lends itself well to being characterized in terms of modalities. For instance, ASP can be characterized in autoepistemic logic by interpreting '$not\ a$' as the epistemic formula $\neg \mathrm{L}a$ ("a is not believed") [Gelfond 1987]. In this chapter, as an alternative, we show how ASP can be characterized within possibility theory. To arrive at this characterization, we first note that ASP is essentially a special case of PASP in which every rule is certain. As such, we will show how PASP can be characterized within possibility theory. We refer to these new semantics for PASP as PASP$_Ł$. A characterization of ASP is then obtained from these new semantics by considering the special case in which all rules are entirely certain.

This characterization of ASP, while still in terms of modalities, stays close in spirit to the Gelfond-Lifschitz reduct. In contrast to the characterization in terms of autoepistemic logic it does not require a special translation of literals to deal with classical negation and disjunction. The core idea of our characterization is to encode the meaning of each rule as a constraint on possibility distributions. Particular minimally specific possibility distributions that satisfy all the constraints imposed by the rules of a program will then correspond to the answer sets of that program.

In this section, we first limit our scope to possibilistic simple programs (Section 4.2.1). Afterwards we will broaden the scope and also consider possibilistic normal programs (Section 4.2.2). The most general case, in which we also consider possibilistic disjunctive programs, will be discussed in Section 4.3.

4.2.1 Characterizing Possibilistic Simple Programs

When considering a fact, i.e. a rule of the form $r = (l_0 \leftarrow \top)$, we know by definition that this rule encodes that the literal in the head is necessarily true, i.e. $N(l_0) = 1$. If we attach a weight to a fact, then this expresses the knowledge that we are not entirely certain of the conclusion in the head, i.e. for a possibilistic rule $p = (r, \lambda)$ we have that $N(l_0) \geq \min(N(\top), \lambda)$. Note that the constraint uses $\geq$, as there may be other rules in the program that allow us to deduce l_0 with a greater certainty.

In a similar fashion we can characterize a rule of the form $(l_0 \leftarrow l_1, ..., l_m)$ as the constraint $N(l_0) \geq N(l_1 \wedge ... \wedge l_m)$ which is equivalent to the constraint $N(l_0) \geq \min(N(l_1), ..., N(l_m))$ due to the min-decomposability property of the necessity measure. Indeed, the intuition of such a rule is that the head is only necessarily true when every part of the body is true. When associating a weight with a rule, we obtain the constraint $N(l_0) \geq \min(N(l_1), ..., N(l_m), \lambda)$ for a possibilistic rule $p = (r, \lambda)$ with $r = (l_0 \leftarrow l_1, ..., l_m)$. Similarly, to characterize a constraint rule, i.e. a rule of the form $r = (\bot \leftarrow l_1, ..., l_m)$, we use the constraint $N(\bot) \geq \min(N(l_1), ..., N(l_m))$, or, in the possibilistic case with $p = (r, \lambda)$, the constraint $N(\bot) \geq \min(N(l_1), ..., N(l_m), \lambda)$.

Definition 14

Let P be a possibilistic simple program and $\pi : \Omega \rightarrow [0, 1]$ a possibility distribution. For every $p \in P$, the constraint $\gamma(p)$ imposed by $p = (r, \lambda)$ with $\lambda \in]0, 1]$, $r = (l_0 \leftarrow l_1, ..., l_m)$ and $m \geq 0$ is given by

$$N(l_0) \geq \min(N(l_1), ..., N(l_m), \lambda). \tag{4.1}$$

$C_P = \{\gamma(p) \mid p \in P\}$ is the set of constraints imposed by program P. If π satisfies the constraints in C_P, π is said to be a possibilistic model of C_P, written $\pi \models C_P$.

A possibilistic model of C_P will also be called a possibilistic model of P. We write S_P for the set of all minimally specific possibilistic models of P.

Definition 15

Let P be a possibilistic simple program. Let π be a minimally specific model of P, i.e. $\pi \in S_P$. Then $V = \left\{l^{N(l)} \mid l \in Lit_P\right\}$ is called a *possibilistic answer set* of P.

Example 29

Consider the possibilistic simple program P with the rules:

$$\begin{aligned} &\mathbf{0.8}\!: a \leftarrow && \mathbf{0.6}\!: \neg b \leftarrow a \\ &\mathbf{0.7}\!: c \leftarrow a, \neg b && \mathbf{0.9}\!: d \leftarrow d. \end{aligned}$$

The set C_P consists of the constraints:

$$\begin{aligned} N(a) &\geq 0.8 & N(\neg b) &\geq \min(N(a), 0.6) \\ N(c) &\geq \min(N(a), N(\neg b), 0.7) & N(d) &\geq \min(N(d), 0.9). \end{aligned}$$

It is easy to see that the last constraint is trivial and can be omitted and that the other constraints can be simplified to $\Pi(\neg a) \leq 0.2$, $\Pi(b) \leq 0.4$ and $\Pi(\neg c) \leq 0.4$. The least specific possibility distribution that satisfies these constraints is given by:

$$\begin{aligned} \pi(\{a,b,c,d\}) &= 0.4 & \pi(\{a,c,d\}) &= 1 & \pi(\{b,c,d\}) &= 0.2 & \pi(\{c,d\}) &= 0.2 \\ \pi(\{a,b,c\}) &= 0.4 & \pi(\{a,c\}) &= 1 & \pi(\{b,c\}) &= 0.2 & \pi(\{c\}) &= 0.2 \\ \pi(\{a,b,d\}) &= 0.4 & \pi(\{a,d\}) &= 0.4 & \pi(\{b,d\}) &= 0.2 & \pi(\{d\}) &= 0.2 \\ \pi(\{a,b\}) &= 0.4 & \pi(\{a\}) &= 0.4 & \pi(\{b\}) &= 0.2 & \pi(\{\}) &= 0.2. \end{aligned}$$

By definition, since the possibility distribution π satisfies the given constraints, it is a possibilistic model. Furthermore, it is easy to see that π is the unique minimally specific possibilistic model (due to least specificity). We can verify that $N(\neg a) = N(b) = N(\neg c) = N(\neg d) = 0$ since we have that $\pi(\{a, c, d\}) = 1$ and that $N(d) = 0$ since $\pi(\{a, c\}) = 1$. Furthermore it is easy to verify that $N(a) = 0.8$, $N(\neg b) = 0.6$ and $N(c) = 0.6$. Hence we find that $V = \left\{a^{0.8}, \neg b^{0.6}, c^{0.6}\right\}$ is a possibilistic answer set of P.

In particular, when we consider all the rules to be entirely certain, i.e. $\lambda = 1$, the results are compatible with the semantics of classical ASP.

Example 30

Consider the program $P = \{(b \leftarrow a), (\neg a \leftarrow)\}$. The set of constraints C_P is given by $N(b) \geq N(a)$ and $N(\neg a) \geq N(\top)$. The first constraint can be rewritten as $1 - \Pi(\neg b) \geq 1 - \Pi(\neg a)$, i.e. as $\Pi(\neg a) \geq \Pi(\neg b)$. The last constraint can be rewritten as $1 - \Pi(a) \geq 1$, i.e. as $\Pi(a) = \max\{\pi(\omega) \mid \omega \models a\} = 0$. Given these two constraints, we find that S_P contains exactly one element, which is defined by

$$\begin{aligned} \pi(\{a,b\}) &= 0 & \qquad \pi(\{a\}) &= 0 \\ \pi(\{b\}) &= 1 & \pi(\{\}) &= 1. \end{aligned}$$

Notice how the first constraint turned out to be of no relevance for this particular example. Indeed, due to the principle of minimal specificity and since there is nothing that prevents $\Pi(\neg a) = 1$, we find that $N(a) = 1 - \Pi(\neg a) = 0$. Therefore the first constraint simplifies to $N(b) \geq 0$. Once more, due to the principle of minimal specificity we thus find that $N(b) = 0$ as there is no information that prevents $\Pi(\neg b) = 1$. To find out whether a, b, $\neg a$ and $\neg b$ are necessarily true w.r.t. the least specific possibility distribution $\pi \in S_P$ arising from the program, we verify whether $N(a) = 1$, $N(b) = 1$, $N(\neg a) = 1$ and $N(\neg b) = 1$, respectively, with N the necessity measure induced by the unique least specific possibility distribution $\pi \in S_P$. As desired, we find that $N(\neg a) = 1 - \Pi(a) = 1$ whereas $N(a) = N(b) = N(\neg b) = 0$. The unique possibilistic answer set is therefore $\{\neg a^1\}$. As we will see, it then follows from Proposition 14 that the unique classical answer set of P is $\{\neg a\}$.

In Propositions 14 and 15, below, we prove that this is indeed a correct characterization of simple programs. First, we present a technical lemma.

Lemma 3

Let L be a set of literals, $M \subseteq L$ a consistent set of literals and let the possibility distribution π be defined as $\pi(\omega) = 1$ if $\omega \models M$ and $\pi(\omega) = 0$ otherwise. Then $M = \{l \mid N(l) = 1, l \in L\}$.

Proposition 14

Let P be a simple program. If $\pi \in S_P$ then either the unique consistent answer set of P is given by $M = \{l \mid N(l) = 1, l \in Lit_P\}$ or π is the vacuous distribution, in which case P does not have any consistent answer sets.

Proposition 15

Let P be a simple program. If M is an answer set of P then the possibility distribution π defined by $\pi(\omega) = 1$ iff $\omega \models M$ and $\pi(\omega) = 0$ otherwise belongs to S_P.

4.2.2 Characterizing Possibilistic Normal Programs

To deal with negation-as-failure, we rely on a reduct-style approach in which a valuation is guessed and it is verified whether this guess is indeed stable. The approach taken in [Gelfond and Lifschitz 1988] to deal with negation-as-failure is to guess an interpretation and verify whether this guess is stable. We propose to treat a rule of the form $r = (l_0 \leftarrow l_1, ..., l_m, not\ l_{m+1}, ..., not\ l_n)$ as the constraint

$$N(l_0) \geq \min\left(N(l_1), ..., N(l_m), 1 - V(l_{m+1}), ..., 1 - V(l_n)\right)$$

where V is the guess for the valuation and where we assume $\min(\{\}) = 1$. Or, when we consider a possibilistic rule $p = (r, \lambda)$, we treat it as the constraint

$$N(l_0) \geq \min\left(N(l_1), ..., N(l_m), 1 - V(l_{m+1}), ..., 1 - V(l_n), \lambda\right).$$

We like to make it clear to the reader that the characterization of normal programs in terms of constraints on possibility distributions in its basic form is little more than a reformulation of the Gelfond-Lifschitz approach. The key difference is that this characterization can be used to guess the certainty with which we can derive particular literals from the available rules, rather than guessing what may or may not be derived from it. Nevertheless, this difference plays a crucial role when dealing with uncertain rules. In particular, this characterization of PASP does not coincide with the semantics of [Nicolas et al. 2006] and adheres to a different intuition for negation-as-failure.

Definition 16

Let P be a possibilistic normal program and let V be a valuation. For every $p \in P$, the constraint $\gamma_V(p)$ induced by $p = (r, \lambda)$ with $\lambda \in]0,1]$, $r = (l_0 \leftarrow l_1, ..., l_m, not\ l_{m+1}, ..., not\ l_n)$ and V is given by

$$N(l_0) \geq \min\left(N(l_1), ..., N(l_m), 1 - V(l_{m+1}), ..., 1 - V(l_n), \lambda\right). \tag{4.2}$$

$C_{(P,V)} = \{\gamma_V(p) \mid p \in P\}$ is the set of constraints imposed by program P and valuation V, and $S_{(P,V)}$ is the set of all minimally specific possibilistic models of $C_{(P,V)}$.

Definition 17

Let P be a possibilistic normal program and let V be a valuation. Let $\pi \in S_{(P,V)}$ be such that

$$\forall l \in Lit_P \cdot N(l) = V(l)$$

then $V = \left\{l^{N(l)} \mid l \in Lit_P\right\}$ is called a possibilistic answer set of P.

Example 31

Consider the possibilistic normal program P from Section 4.1. The constraints C_P induced by P are:

$$N(invalid) \geq 0.1$$
$$N(airport) \geq \min(1 - V(invalid), 1)$$

From the first constraint it readily follows that we need to choose $V(invalid) = 0.1$ to comply with the principle of minimal specificity. The other constraint can then readily be simplified to:

$$N(airport) \geq 0.9$$

Hence it follows that $V = \left\{invalid^{0.1}, airport^{0.9}\right\}$ is the unique possibilistic answer set of P.

It is easy to see that the proposed semantics remain closer to the intuition of the possibilistic normal program discussed in the introduction. Indeed, we conclude with a high certainty that we need to go to the airport.

Still, it is interesting to further investigate the particular relationship between the semantics for PASP as proposed in [Nicolas et al. 2006] and the semantics presented in this section. Let the possibilistic rule r be of the form:

$$\boldsymbol{\lambda} : l_0; ...; l_k \leftarrow l_{k+1}, ..., l_m, not\ l_{m+1}, ..., not\ l_n.$$

When we determine the reduct w.r.t. a valuation V of the possibilistic program containing r, then the certainty of the rule in the reduct that corresponds with r can be verified to be:

$$\min(F_N(V(l_{m+1})), ..., F_N(V(l_n)), \lambda)$$

with F_N a fuzzy negator, i.e. where F_N is a decreasing function with $F_N(0) = 1$ and $F_N(1) = 0$. In particular, for the semantics of [Nicolas et al. 2006] we have that F_N is the Gödel negator F_{G}, defined as $F_{\text{G}}(0) = 1$ and $F_{\text{G}}(c) = 0$ with $0 < c \leq 1$. In the semantics for PASP presented in this section, F_N is the Łukasiewicz negator $F_{\text{Ł}}(c) = 1 - c$ with $0 \leq c \leq 1$. Thus, for a rule such as:

$$\mathbf{0.9} : b \leftarrow not\ a$$

and a valuation $V = \{a^{0.2}\}$ we obtain under the approach from [Nicolas et al. 2006] the reduct $(\mathbf{0} : b \leftarrow)$, whereas under our approach we obtain the constraint $N(b) \geq \min(1 - 0.2, 0.9)$, which can be encoded by the rule $(\mathbf{0.8} : b \leftarrow)$. Essentially, the difference between both semantics can thus be reduced to a difference in the choice of negator. However, even though the semantics share similarities, there is a notable difference in the underlying intuition of both approaches. Specifically, in the semantics presented in this chapter, we have that '$not\ l$' is understood as "*the degree to which '$\neg l$' is possible*", or, equivalently, "*the degree to which it is not the case that we can derive 'l' with certainty*". This contrasts with the intuition of '$not\ l$' in [Nicolas et al. 2006] as a Boolean condition and understood as "*we cannot derive 'l' with a strictly positive certainty*".

Interestingly, we find that the complexity of the main reasoning tasks for possibilistic normal programs remains at the same level of the polynomial hierarchy as the corresponding normal ASP programs.

While we will see in Section 4.4 that the complexity of possibilistic normal programs remains unchanged compared to classical normal programs, it is important to note that under the semantics proposed in this section there is no longer a 1-on-1 mapping between the classical answer sets of a normal program and the possibilistic answer sets. Indeed, if we consider a possibilistic normal program constructed from a classical normal program where we attach certainty $\lambda = 1$ to each rule, then we can sometimes obtain additional intermediary answer sets. Consider the next example:

Example 32

Consider the normal program with the single rule $a \leftarrow not\ a$. This program has no classical answer sets. Now consider the possibilistic normal program P with the rule

$$\mathbf{1}\colon a \leftarrow not\ a.$$

The set of constraints $C_{(P,V)}$ is given by

$$N(a) \geq \min(1 - V(a), 1).$$

This constraint can be rewritten as

$$\begin{aligned} & N(a) \geq \min(1 - V(a), 1) \\ & \equiv N(a) \geq 1 - V(a) \\ & \equiv 1 - \Pi(\neg a) \geq 1 - V(a) \\ & \equiv \Pi(\neg a) \leq V(a). \end{aligned}$$

We thus find that the set $S_{(P,V)}$ is a singleton with $\pi \in S_{(P,V)}$ defined by $\pi(\{a\}) = 1$ and $\pi(\{\}) = V(a)$. We can now establish for which choices of $V(a)$ it holds that $V(a) = N(a)$:

$$\begin{aligned} V(a) &= N(a) \\ \Pi(\neg a) &= 1 - \Pi(\neg a) \\ 2 \cdot \Pi(\neg a) &= 1 \end{aligned}$$

and thus, since $\Pi(\neg a) \leq V(a)$, we have $\pi(\{\}) = 0.5$. The unique possibilistic answer set of P is therefore $\{a^{0.5}\}$. In the same way, one may verify that the program

$$\mathbf{1}\colon a \leftarrow not\ b \qquad\qquad \mathbf{1}\colon b \leftarrow not\ a$$

has an infinite number of possibilistic answer sets, i.e. $\{a^c, b^{1-c}\}$ for every $c \in [0, 1]$.

For practical purposes, however, this behaviour has a limited impact as we only need to consider a finite number of certainty levels to perform brave/cautious reasoning. Indeed, we only need to consider the certainties used in the program, their complement to account for negation-as-failure and $\frac{1}{2}$ to account for the intermediary value as in Example 32. Thus, for the main reasoning tasks it suffices to limit our attention to the certainties from the set $cert^+(P)$.

We now show that when we consider rules with an absolute certainty, i.e. classical normal programs, we obtain a correct characterization of classical ASP, provided that we restrict ourselves to absolutely certain conclusions, i.e. valuations V for which it holds that $\forall l \cdot V(l) \in \{0,1\}$.

Example 33

Consider the program P with the rules

$$a \leftarrow \qquad\qquad b \leftarrow b \qquad\qquad c \leftarrow a, not\ b.$$

The set of constraints $C_{(P,V)}$ is then given by

$$N(a) \geq 1 \qquad\qquad N(b) \geq N(b) \qquad\qquad N(c) \geq \min\left(N(a), 1 - V(b)\right).$$

We can rewrite the first constraint as $1 - \Pi(\neg a) \geq 1$ and thus $\Pi(\neg a) = 0$. The second constraint is trivially satisfied and, since it does not entail any new information, can be dropped. The last constraint can be rewritten as $\Pi(\neg c) \leq 1 - \min(1 - \Pi(\neg a), 1 - V(b))$, which imposes an upper bound on the value that $\Pi(\neg c)$ can assume. Since we already know that $\Pi(\neg a) = 0$ we can further simplify this inequality to $\Pi(\neg c) \leq 1 - \min(1-0, 1-V(b)) = 1 - (1 - V(b)) = V(b)$. In conclusion, the program imposes the constraints

$$\Pi(\neg a) = 0 \qquad\qquad\qquad \Pi(\neg c) \leq V(b).$$

The set $S_{(P,V)}$ then contains exactly one element, which is defined by

$$\begin{aligned} \pi(\{a,b,c\}) &= 1 & \pi(\{b,c\}) &= 0 \\ \pi(\{a,b\}) &= V(b) & \pi(\{b\}) &= 0 \\ \pi(\{a,c\}) &= 1 & \pi(\{c\}) &= 0 \\ \pi(\{a\}) &= V(b) & \pi(\{\}) &= 0. \end{aligned}$$

Note that this possibility distribution is independent of the choice for $V(a)$ and $V(c)$ since there are no occurrences of '$not\ a$' and '$not\ c$' in P. It remains then to determine for which choices of $V(b)$ it holds that $V(b) = N(b)$, i.e. for which the guess $V(b)$ is stable. We have:

$$V(b) = N(b) = 1 - \Pi(\neg b) = 1 - \max\left\{\pi(\omega) \mid \omega \models \neg b\right\} = 0$$

and thus we find that $\pi(\{a,b\}) = \pi(\{a\}) = 0$. We have $N(a) = 1 - \Pi(\neg a) = 1$, $N(c) = 1 - \Pi(\neg c) = 1$ and $N(b) = 1 - \Pi(\neg b) = 0$. As we will see in the next propositions, the unique answer set of P is therefore $\{a, c\}$.

Proposition 16

Let P be a normal program and V a valuation. Let $\pi \in S_{(P,V)}$ be such that

$$\forall l \in Lit_P \cdot V(l) = N(l) \text{ ; and} \tag{4.3}$$

$$\forall l \in Lit_P \cdot N(l) \in \{0,1\} \tag{4.4}$$

then $M = \{l \mid N(l) = 1, l \in Lit_P\}$ is an answer set of the normal program P.

Proof. This proposition is a special case of Proposition 18 presented below. □

Note that the requirement stated in (4.4) cannot be omitted. Let us consider Example 32, in which we considered the normal program $P = \{a \leftarrow not\ a\}$. This normal program P has no classical answer sets. The constraint that corresponds with the rule $(a \leftarrow not\ a)$ is $N(a) \geq 1 - V(a)$. For a choice of $V = \{a^{0.5}\}$, however, we would find that $V(a) = N(a)$ and thus that V is an answer set of P if we were to omit this requirement.

Proposition 17

Let P be a normal program. If M is an answer set of P, there is a valuation V, defined by $V(l) = 1$ if $l \in M$ and $V(l) = 0$ otherwise, and a possibility distribution $\pi \in S_{(P,V)}$ such that for every $l \in Lit_P$ we have $V(l) = N(l)$ (i.e. $N(l) = 1$ if $l \in M$ and $N(l) = 0$ otherwise).

Proof. This proposition is a special case of Proposition 19 presented below. □

We like to point out to the reader that we could try to encode the information in a rule in such a way that we interpret '$not\ a$' as $\Pi(\neg a)$, which closely corresponds to the intuition of negation-as-failure. Indeed, when it is completely possible to assume that '$\neg a$' is true, then surely '$not\ a$' is true. Under this encoding, however, we run into a significant problem. Consider the rules $(b \leftarrow not\ c)$ and $(c \leftarrow not\ b)$. These rules would then correspond with the constraints $N(b) \geq \Pi(\neg c)$ and $N(c) \geq \Pi(\neg b)$, respectively. Notice though that both constraints can be rewritten as the constraint $1 - \Pi(\neg b) \geq \Pi(\neg c)$. This would imply that both rules are semantically equivalent in ASP, which is clearly not the case. Hence we cannot directly encode '$not\ a$' as $\Pi(\neg a)$ and guessing a valuation is indeed necessary since without the guess V we would not be able to obtain a unique set of constraints. As we have shown this only affects literals preceded by negation-as-failure and we can continue to interpret a literal 'b' as $N(b)$.

4.3 Possibilistic Semantics of Disjunctive ASP Programs

We now turn our attention to how we can characterize disjunctive rules. We found in Section 4.2 that we can characterize a rule of the form $r = (head \leftarrow body)$ as the constraint $N(head) \geq N(body)$, or, similarly, that we can characterize a possibilistic rule $p = (r, \lambda)$ as the constraint $N(head) \geq \min(N(body), \lambda)$. Such a characterization works particularly well due the min-decomposability w.r.t. conjunction. Indeed, since the body of e.g. a simple rule $r = (l_0 \leftarrow l_1, ..., l_m)$ is a conjunction of literals we can write $body = l_1 \wedge ... \wedge l_m$. Then $N(body)$ can be rewritten as $\min(N(l_1), ..., N(l_m))$, which allows for a straightforward simplification. In a similar fashion, for a positive disjunctive rule $r = (l_0; ...; l_k \leftarrow l_{k+1}, ..., l_m)$ we can readily write $N(body)$ as $\min(N(l_{k+1}), ..., N(l_m))$. We would furthermore like to simplify $N(head)$ with $head = l_0 \vee ... \vee l_k$. However, we do not have that $N(head) = \max(N(l_0), ..., N(l_k))$. Indeed, in general we only have that $N(head) \geq \max(N(l_0), ..., N(l_k))$. This means that we can either choose to interpret the head as $\max(N(l_0), ..., N(l_k))$ or $N(l_0 \vee ... \vee l_k)$. In particular, a *possibilistic disjunctive rule* $p = (r, \lambda)$ with

$$r = (l_0; ...; l_k \leftarrow l_{k+1}, ..., l_m, not\ l_{m+1}, ..., not\ l_n)$$

can either be interpreted as the constraint

$$\max(N(l_0), ..., N(l_k)) \geq \min(N(l_{k+1}), ..., N(l_m), 1 - V(l_{m+1}), ..., 1 - V(l_n), \lambda) \quad (4.5)$$

which we will call the *strong* interpretation of disjunction, or as the constraint

$$N(l_0 \vee ... \vee l_k) \geq \min(N(l_{k+1}), ..., N(l_m), 1 - V(l_{m+1}), ..., 1 - V(l_n), \lambda) \quad (4.6)$$

which we will call the *weak* interpretation of disjunction.

The choice of how to treat disjunction is an important one that crucially impacts the nature of the resulting answer sets. For example, the non-deterministic nature of strong disjunction provides a useful way to generate different (candidate) solutions, whereas weak disjunction is oftentimes better suited when we are interested in modelling the epistemic state of an agent since it amounts to accepting the disjunction as being true rather than making a choice of which disjunct to accept. In this section we consider both characterizations; the characterization of disjunction as (4.5) is discussed in Section 4.3.1 and in

Section 4.3.2 we discuss the characterization as (4.6). In particular we will show that the first characterization of disjunction corresponds to the semantics of disjunction found in ASP whereas the Boolean counterpart of the second characterization has, to the best of our knowledge, not yet been studied in the literature.

4.3.1 Strong Possibilistic Semantics of Disjunctive Rules

We first consider the characterization of disjunction in which we treat a disjunction of the form '$l_0; ...; l_k$' as $\max(N(l_0), \ldots, N(l_k))$. As it turns out, under these strong possibilistic semantics the disjunction behaves as in classical ASP.

Definition 18

Let P be a possibilistic disjunctive program and let V be a valuation. For every possibilistic disjunctive rule $p = (r, \lambda)$ with $\lambda \in \,]0, 1]$ and r a rule of the form $r = (l_0; ...; l_k \leftarrow l_{k+1}, ..., l_m, not\ l_{m+1}, ..., not\ l_n)$ the constraint $\gamma^{\mathrm{s}}_V(p)$ induced by p and V is given by

$$\max(N(l_0), ..., N(l_k)) \geq \min(N(l_{k+1}), ..., N(l_m), 1 - V(l_{m+1}), ..., 1 - V(l_n), \lambda) \tag{4.7}$$

$C^{\mathrm{s}}_{(P,V)} = \left\{\gamma^{\mathrm{s}}_V(p) \mid p \in P\right\}$ is the set of constraints imposed by program P and V, and $S^{\mathrm{s}}_{(P,V)}$ is the set of all minimally specific possibilistic models of $C^{\mathrm{s}}_{(P,V)}$.[1]

Whenever P is a positive disjunctive program, i.e. whenever P is a disjunctive program without negation-as-failure, (4.7) is independent of V and we simplify the notation to $\gamma^{\mathrm{s}}, C^{\mathrm{s}}_P$ and S^{s}_P.

Notice that, unlike in possibilistic logic where a unique least specific possibility distribution exists because of the specific form of the considered constraints, the constraint of the form (4.7) can give rise to multiple minimally specific possibility distributions of which some will correspond with answer sets. Indeed, the program $P = \{a; b \leftarrow\}$ induces the constraint $\max(N(a), N(b)) \geq 1$, which has two minimally specific possibility distributions, yet no least specific possibility distribution. Indeed, we have the minimally specific possibility distributions π_1, π_2 defined by

$$\begin{array}{rrrr} \pi_1(\{a,b\}) = 1 & \pi_1(\{b\}) = 0 & \pi_2(\{a,b\}) = 1 & \pi_2(\{b\}) = 1 \\ \pi_1(\{a\}) = 1 & \pi_1(\{\}) = 0 & \pi_2(\{a\}) = 0 & \pi_2(\{\}) = 0 \end{array}$$

[1]We use the superscript 's' to highlight that we employ the semantics of *s*trong disjunction.

Definition 19

Let P be a possibilistic disjunctive program and let V be a valuation. Let $\pi \in S^{\text{s}}_{(P,V)}$ be such that

$$\forall l \in Lit_P \cdot N(l) = V(l)$$

then $V = \left\{ l^{N(l)} \mid l \in Lit_P \right\}$ is called a possibilistic answer set of P.

We now further illustrate the semantics and the underlying intuition by considering a possibilistic disjunctive program in detail.

Example 34

Consider the possibilistic (positive) disjunctive program P with the following rules:

$$\begin{aligned} \mathbf{0.8}\!: a; b &\leftarrow \\ \mathbf{0.6}\!: c &\leftarrow a \\ \mathbf{0.4}\!: c &\leftarrow b. \end{aligned}$$

The constraints C^{s}_P induced by this program are:

$$\begin{aligned} \max(N(a), N(b)) &\geq 0.8 \\ N(c) &\geq \min(N(a), 0.6) \\ N(c) &\geq \min(N(b), 0.4). \end{aligned}$$

From the first constraint it follows that we either need to choose $V(a) = 0.8$ or $V(b) = 0.8$, in accordance with the principal of minimal specificity. Hence, we either obtain $V(c) = 0.6$ or $V(c) = 0.4$. As such we find that the two unique possibilistic answer sets of P are $\left\{a^{0.8}, c^{0.6}\right\}$ and $\left\{b^{0.8}, c^{0.4}\right\}$.

As before, if we restrict ourselves to rules that are entirely certain we obtain a characterization of disjunctive programs in classical ASP.

Example 35

Consider the program P with the rules

$$a; b \leftarrow \qquad\qquad a \leftarrow b$$

The set of constraints C_P^{s} is given by

$$\max(N(a), N(b)) \geq N(\top) = 1 \qquad\qquad N(a) \geq N(b).$$

Intuitively, the first constraint induces a choice. To satisfy this constraint, we need to take either $N(a) = 1$ or $N(b) = 1$. Depending on our choice, we can consider two possibility distributions. The possibility distribution π_1 is the least specific possibility distribution that satisfies the constraints $N(a) = 1$ and $N(a) \geq N(b)$, whereas π_2 is the least specific possibility distribution satisfying the constraints $N(b) = 1$ and $N(a) \geq N(b)$:

$$\pi_1(\{a,b\}) = 1 \qquad\qquad \pi_1(\{b\}) = 0$$
$$\pi_1(\{a\}) = 1 \qquad\qquad \pi_1(\{\}) = 0$$

and

$$\pi_2(\{a,b\}) = 1 \qquad\qquad \pi_2(\{b\}) = 0$$
$$\pi_2(\{a\}) = 0 \qquad\qquad \pi_2(\{\}) = 0.$$

It is clear that the possibility distribution π_2 cannot be minimally specific w.r.t. the constraints $\max(N(a), N(b)) = 1$ and $N(a) \geq N(b)$ since $\pi_1(\{a\}) > \pi_2(\{a\})$ and $\pi_1(\omega) \geq \pi_2(\omega)$ for all other interpretations ω. We thus have that S_P^{s} only contains a single element, namely π_1. With N the necessity measure induced by π_1 we obtain $N(a) = 1$ and $N(b) = 0$. As will follow from Proposition 18 and 19 the unique answer set of P is therefore $\{a\}$.
Let us now add the rule $(b \leftarrow \mathit{not}\ b)$ to P. Notice that in classical ASP this extended program has no answer sets. The set of constraints $C_{(P,V)}^{\mathrm{s}}$ is given by:

$$C_P^{\mathrm{s}} \cup \{N(b) \geq 1 - V(b)\}.$$

This new constraint, intuitively, tells us that 'b' must necessarily be true, since we force it to be true whenever it is not true. Note, however, that the act of making 'b' true effectively removes the motivation for making it true in the first place. As expected, we cannot find any minimally specific possibilistic model that agrees with the constraints imposed by P and V such that $\forall l \in \mathit{Lit}_P \cdot N(l) \in \{0,1\}$. The problem has to do with our choice of $V(b)$. If we take $V(b) = 1$ then the constraint imposed by the first rule still forces us to choose either $N(a) = 1$ or $N(b) = N(a) = 1$ due to the interplay with the constraint imposed by the second rule. However, $S_{(P,V)}^{\mathrm{s}}$ contains only one minimally specific possibility distribution, namely the one with $N(a) = 1$. Hence $N(b) = 0 \neq V(b)$. If we take $V(b) = 0$ then the last rule forces $N(b) = 1$. Hence $V(b) = 0 \neq 1 = N(b)$.

Now that we have clarified the intuition, we can formalize the connection between the strong possibilistic semantics and classical disjunctive ASP.

Proposition 18

Let P be a disjunctive program, V a valuation and let $\pi \in S^{\mathrm{s}}_{(P,V)}$ be such that

$$\forall l \in Lit_P \cdot V(l) = N(l) \text{ ; and} \tag{4.8}$$
$$\forall l \in Lit_P \cdot N(l) \in \{0,1\} \tag{4.9}$$

then $M = \{l \mid N(l) = 1, l \in Lit_P\}$ is an answer set of the disjunctive program P.

Proposition 19

Let P be a disjunctive program. If M is an answer set of P, there is a valuation V, defined as $V(l) = 1$ if $l \in M$ and $V(l) = 0$ otherwise, and a possibility distribution π, defined as $\pi(\omega) = 1$ if $\omega \models M$ and $\pi(\omega) = 0$ otherwise, such that $\pi \in S^{\mathrm{s}}_{(P,V)}$ and for every $l \in Lit_P$ we have $V(l) = N(l)$.

4.3.2 Weak Possibilistic Semantics of Disjunctive Rules

Under the strong possibilistic semantics of disjunction we consider all the disjuncts of a satisfied rule separately. Under this non-deterministic view the rule $(a; b \leftarrow)$ means that 'a' is believed to be true or 'b' is believed to be true. When looking at answer sets as epistemic states it becomes apparent that there is also another choice in how we can treat disjunction in the head. Indeed, we can look at the disjunction as a whole to hold, without making any explicit choices as to which of the disjuncts holds. When trying to reason about one's knowledge there are indeed situations in which we do not want, or simply cannot make, a choice as to which of the disjuncts is true. This implies that we need to look at an answer set as a set of clauses, rather than a set of literals. In the remainder of this chapter, we syntactically differentiate between both approaches by using the notation $l_0; ...; l_k$ and $l_0 \vee ... \vee l_k$ to denote strong disjunction and weak disjunction, respectively.

Example 36

Consider the following example. A SCADA (supervisory control and data acquisition) system is used to monitor the brewing of beer in an industrialised setting. To control the fermentation, the system regularly verifies an air-lock for the presence of bubbles. An absence of bubbles may be due to a number of possible causes. On the one hand there may be a production problem such as a low yeast count or low temperature. Adding yeast when the temperature is low results in a beer with a strong yeast flavour, which should be avoided. Raising the temperature when there is too little yeast present will kill off the remaining yeast and will ruin the entire batch. On the other hand, there may be technical problems. There may be a malfunction in the SCADA system, which can be verified by running a diagnostic. The operator runs a diagnostic ($diagnostic$), which reports back that there is no malfunction ($\neg malfunction$). Or, alternatively, the air-lock may not be sealed correctly ($noseal$). The operator furthermore checks the temperature because he suspects that the temperature is the problem ($verifytemp$), but the defective temperature sensor returns no temperature when checked ($notemp$). These three technical problems require physical maintenance and the operator should send someone out to fix them. Technical problems do not affect the brewing. As such, the brewing process should not be interrupted for such problems as this will ruin the current batch. If there is a production problem, however, the brewing process needs to be interrupted as soon as possible (in addition, evidently, to interrupting the brewing process when the brewing is done). This prevents the current batch from being ruined due to over-brewing but also allows the interaction with the contents of the kettle. In particular, when the problem is diagnosed to be low yeast the solution is to add a new batch of yeast and restart the process. Similarly, low temperature can be solved by raising the kettle temperature and restarting the fermentation process. Obviously, the goal is to avoid ruining the current batch. An employer radios in that the seal is okay. We have the following program:

$$\begin{aligned}
lowyeast \lor lowtemp \lor noseal \lor malfunction &\leftarrow not\ bubbles\\
diagnostic &\leftarrow\\
\neg malfunction &\leftarrow diagnostic\\
verifytemp &\leftarrow\\
notemp &\leftarrow verifytemp\\
maintenance &\leftarrow noseal \lor malfunction \lor notemp\\
brew &\leftarrow not\ (lowyeast \lor lowtemp \lor done)
\end{aligned}$$

$$\begin{aligned} addyeast &\leftarrow lowyeast \\ raisetemp &\leftarrow lowtemp \\ ruin &\leftarrow raisetemp, not\ lowtemp \\ ruin &\leftarrow addyeast, not\ lowyeast \\ ruin &\leftarrow not\ brew, not\ (lowtemp \vee lowyeast) \\ &\leftarrow ruin \\ \neg noseal &\leftarrow \end{aligned}$$

The program above does not use the standard ASP syntax since we allow for disjunction in the body. Furthermore, the disjunction used in the head and the body is weak disjunction. The only information that we can therefore deduce from e.g. the first rule is $(lowyeast \vee lowtemp \vee noseal \vee malfunction)$. At first, this new form of disjunction may indeed appear weaker than strong disjunction since it does not induce a choice. Still, even without inducing a choice, conclusions obtained from other rules may allow us to refine our knowledge. In particular, note that from $lowyeast \vee lowtemp \vee noseal \vee malfunction$ together with $\neg malfunction$ and $\neg noseal$ we can entail $lowyeast \vee lowtemp$. Similarly, conclusions can also have prerequisites that are disjunctions. For example, we can no longer deduce $brew$ since $lowyeast \vee lowtemp$ entails $lowyeast \vee lowtemp \vee done$. From $maintenance \leftarrow noseal \vee malfunction \vee notemp$ and $notemp$ we can deduce that we should call maintenance. However, we do not yet have enough information to diagnose whether yeast should be added or whether the temperature should be raised. The unique answer set of this program, according to the semantics of weak disjunction which we present in this section, is given by

$$\{lowyeast \vee lowtemp, maintenance,$$
$$diagnostic, \neg malfunction, verifytemp, notemp, \neg noseal\}.$$

To conclude this example, note that e.g. $(noseal \vee malfunction \vee notemp)$ cannot simply be replaced by a single atom. An atom would be needed for every subset of disjuncts and there may be an exponential number of such subsets. This approach would therefore not be efficient in examples with a large number of disjuncts.

In the remainder of this section we extend the PASP semantics with the notion of clauses, rather than literals, and define an applicable immediate consequence operator for programs composed of clauses. We then prove some important properties, such as the monotonicity of the immediate consequence operator. For the classical case (i.e. when omitting weights), we furthermore characterize the complexity of clausal programs, both with and without negation-as-failure in Section 4.4. In particular, we show how the complexity is critically determined by whether we restrict ourselves to atoms and highlight, as shown by the higher complexity of some of the reasoning tasks, that weak disjunction is a non-trivial extension of ASP.

We start by giving the definition of possibilistic clausal programs, i.e. possibilistic programs with a syntax that allows for disjunction in the body. We then define the weak possibilistic semantics of such clausal programs in terms of constraints on possibility distributions. We also introduce an equivalent characterization based on an immediate consequence operator and a reduct, which is more in line with the usual treatment of ASP programs. When all the rules are entirely certain we obtain the classical counterpart, which we name clausal programs.

Semantical Characterization

We rely on the notion of a *clause*, i.e. a finite disjunction of literals. Consistency and entailment for sets of clauses are defined as in propositional logic. As such, we can derive from the information '$a \vee b \vee c$' and '$\neg b$' that '$a \vee c$' is true.

Definition 20

A *clausal rule* is an expression of the form $(e_0 \leftarrow e_1, ..., e_m, not\ e_{m+1}, ..., not\ e_n)$ with e_i a clause for every $0 \leq i \leq n$. A *positive clausal rule* is an expression of the form $(e_0 \leftarrow e_1, ..., e_m)$, i.e. a clausal rule without negation-as-failure. A *(positive) clausal program* is a finite set of (positive) clausal rules.

For a clausal rule, which is of the form $r = (e_0 \leftarrow e_1, ..., e_m, not\ e_{m+1}, ..., not\ e_n)$, we say that e_0 is the *head* and that $e_1, ..., e_m, not\ e_{m+1}, ..., not\ e_n$ is the *body* of the clausal rule. We use the notation $head(r)$ and $body(r)$ to denote the clause in the head, resp. the set of clauses in the body. The Herbrand base $\mathcal{B}_P$ of a clausal program P is still defined as the set of atoms appearing in P. As such, possibility distributions are defined in the usual way as $\pi : 2^{\mathcal{B}_P} \rightarrow [0,1]$ mappings.

Until now, we were able to define the possibility distributions that satisfied the constraints imposed by the rules in a program in terms of a valuation V, i.e. a $V : Lit_P \rightarrow [0,1]$ mapping. This need no longer be the case. Specifically, note that we will now impose constraints of the form $N(l_0 \vee ... \vee l_k) \geq \lambda$. Assume that we have a possibility distribution π defined as

$$\pi(\{a,b,c\}) = 0 \qquad \pi(\{a,b\}) = 0 \qquad \pi(\{a,c\}) = 1 \qquad \pi(\{a\}) = 1$$
$$\pi(\{b,c\}) = 0 \qquad \pi(\{b\}) = 0 \qquad \pi(\{c\}) = 1 \qquad \pi(\{\}) = 0.$$

This possibility distribution is the least specific possibility distribution that satisfies the constraints $N(a \vee b \vee c) = 1$ and $N(\neg b) = 1$. However, it can be verified that this possibility distribution cannot be defined in terms of a mapping $V : Lit_P \to [0,1]$.

Instead, we define the set of clauses appearing in the head of the rules of a clausal program P as $Clause_P = \{head(r) \mid r \in P\}$. Given a clausal program, it is clear that the only information that can be derived from the program are those clauses that are in the head of a rule. To compactly describe a possibility distribution imposed by clausal programs we therefore consider sets of weighted clauses, where a set of weighted clauses E corresponds with the set of constraints $\{N(e) \geq \lambda \mid e^\lambda \in E\}$. We use the notations E^λ and $E^{\underline{\lambda}}$ to denote the sets $\left\{e^{\lambda'} \mid e^{\lambda'} \in E, \lambda' \geq \lambda\right\}$ and $\left\{e^{\lambda'} \mid e^{\lambda'} \in E, \lambda' > \lambda\right\}$, respectively. Entailment for sets of weighted clauses is defined as in possibilistic logic, i.e. if we consider the least specific possibility distribution π_E satisfying the constraints $\{N_E(e) \geq \lambda \mid e^\lambda \in E\}$ then $E \models p^\lambda$ with 'p' a proposition iff $N_E(p) \geq \lambda$. In particular, recall from possibilistic logic the inference rules (GMP) or graded modus ponens, i.e. we can infer from $N(\alpha) \geq \lambda$ and $N(\alpha \to \beta) \geq \lambda'$ that $N(\beta) \geq \min(\lambda, \lambda')$. In addition recall the inference rule (S), i.e. we can infer from $N(\alpha) \geq \lambda$ that $N(\alpha) \geq \lambda'$ with $\lambda \geq \lambda'$. Consistency for sets of weighted clauses is also defined as in possibilistic logic.

Definition 21

A *possibilistic (positive) clausal program* is a set of possibilistic (positive) clausal rules, which are pairs $p = (r, \lambda)$ with r a (positive) clausal rule and $\lambda \in \,]0,1]$ a certainty associated with r.

We define P^* and the λ-cut P_λ as usual.

We are now almost able to define the semantics of weak disjunction. In the previous sections we guessed a valuation and used this valuation to deal with negation-as-failure. However, for clausal programs, a new problem arises. Note that the least specific possibility distribution that satisfies the constraints $N(a \vee b \vee c) = 1$ and $N(\neg b) = 1$ is also the least specific possibility distribution that satisfies the constraints $N(a \vee c)$ and $N(\neg b)$. As such, if $Clause_P = \{(a \vee b \vee c), (\neg b), (a \vee c)\}$, there would not be a unique set of weighted clauses that can be used to define this least specific possibility distribution. Indeed, a set of weighted clauses uniquely defines a possibility distribution, but not vice versa. To avoid such ambiguity, we will instead immediately guess a possibility distribution π_E and use this possibility distribution to deal with negation-as-failure in a clausal program.

Definition 22

Let P be a possibilistic clausal program and let π_E be a possibility distribution. For every $p \in P$, the constraint $\gamma^{\mathrm{w}}_{\pi_E}(p)$ induced by $p = (r, \lambda)$ with $\lambda \in]0, 1]$, $r = (e_0 \leftarrow e_1, ..., e_m, not\ e_{m+1}, ..., not\ e_n)$ and π_E under the weak possibilistic semantics is given by

$$N(e_0) \geq \min(N(e_1), ..., N(e_m), 1 - N_E(e_{m+1}), ..., 1 - N_E(e_n), \lambda). \tag{4.10}$$

$C^{\mathrm{w}}_{(P,\pi_E)} = \left\{\gamma^{\mathrm{w}}_{\pi_E}(p) \mid p \in P\right\}$ is the set of constraints imposed by program P and π_E, and $S^{\mathrm{w}}_{(P,\pi_E)}$ is the set of all minimally specific possibilistic models of $C^{\mathrm{w}}_{(P,\pi_E)}$.

Whenever P is a possibilistic (positive) clausal program, i.e. whenever P is a possibilistic clausal program without negation-as-failure, (4.10) is independent of π_E and we simplify the notation to $\gamma^{\mathrm{w}}, C^{\mathrm{w}}_P$ and S^{w}_P.

Definition 23

Let P be a possibilistic clausal program. Let π_E be a possibility distribution such that $\pi_E \in S^{\mathrm{w}}_{(P,\pi_E)}$. We then say that π_E is a possibilistic answer set of P.

As already indicated we can also use a set of weighted clauses E to concisely describe π_E. For compactness, we slightly abuse the terminology. Specifically, when we say that E is a possibilistic answer set of the clausal program P we are stating that the possibility distribution induced by E is a possibilistic answer set of the clausal program P. Finally, a (possibilistic) answer set E of the clausal program P is said to be consistent when the set of weighted clauses E is consistent.

Lemma 4

Let P be a possibilistic positive clausal program. Then $S^{\mathrm{w}}_{(P,\pi_E)}$ is a singleton, i.e. $\pi \in S^{\mathrm{w}}_{(P,\pi_E)}$ is a least specific possibility distribution.

Proof. This readily follows from the form of the constraints imposed by the rules $p \in P$ and since a possibilistic positive clausal program is free of negation-as-failure. □

Example 37

Consider the possibilistic clausal program P with the rules:

$$\begin{aligned}\mathbf{1}\colon & a \vee c \vee d \leftarrow \\ \mathbf{0.4}\colon & \neg d \leftarrow \\ \mathbf{0.8}\colon & e \leftarrow \mathit{not}\ (a \vee b \vee c).\end{aligned}$$

We have that $C^{\mathrm{w}}_{(P,\pi_E)}$ is the set of constraints:

$$\begin{aligned}N(a \vee c \vee d) &\geq 1 \\ N(\neg d) &\geq 0.4 \\ N(e) &\geq \min(1 - N_E(a \vee b \vee c), 0.8).\end{aligned}$$

We can rewrite the first constraint as $N(\neg d \rightarrow a \vee c) \geq 1$. Given the second constraint $N(\neg d) \geq 0.4$ we can apply the inference rule (GMP) to conclude that $N(a \vee c) \geq 0.4$. From propositional logic we know that $(a \vee c) \rightarrow (a \vee b \vee c)$, i.e. we also have $N(a \vee b \vee c) \geq 0.4$.

For π_E to be an answer set of P we know from Definition 23 that we must have that $\pi \in S^{\mathrm{w}}_{(P,\pi_E)}$ with $\pi = \pi_E$. In other words, we must have that $N_E(a \vee b \vee c) = N(a \vee b \vee c) \geq 0.4$. Due to the principle of least specificity, which implies that $N(a \vee b \vee c) = 0.4$, the last constraint can be simplified to $N(e) \geq \min(1 - 0.4, 0.8)$ or $N(e) \geq 0.6$. As such, the least specific possibility distribution defined by the constraints $N(e) \geq 0.6$, $N(a \vee c \vee d) \geq 1$ and $N(\neg d) \geq 0.4$ is a possibilistic answer set of P.

Notice that we implicitly defined the possibilistic answer set of the previous example as a set of weighted clauses, i.e. in terms of clauses that appear in the head. Alternatively we could thus write that $E = \left\{e^{0.6}, a \vee b \vee d^{1}, \neg b^{0.4}\right\}$ defines the possibilistic answer set of P. This idea will be further developed in Section 4.3.2 to avoid the need to explicitly define a possibility distribution (which would require an exponential amount of space) and instead rely on an encoding of a possibility distribution by a (polynomial) set of weighted clauses.

For the crisp case, we only want clauses that are either entirely certain or completely uncertain, i.e. true or false. To this end, we add the constraint (4.11), which is similar to (4.4) from Proposition 16.

Definition 24

Let P be a clausal program and $\pi_E \in S^{\mathrm{w}}_{(P,\pi_E)}$ a possibility distribution such that

$$\forall \omega \in \Omega \cdot \pi_E(\omega) \in \{0,1\} \tag{4.11}$$

then π_E is called an answer set of P.

Syntactic Characterization

We now introduce a syntactic counterpart of the semantics for weak disjunction by defining an immediate consequence and reduct operator. As such, it is more in line with the classical Gelfond-Lifschitz approach. In addition, the syntactic approach only needs a polynomial amount of space (as we will only consider clauses appearing in the head of the clausal rules). Indeed, what we will do is formalise the idea of using a set of weighted clauses to determine the possibilistic answer sets of a clausal program, rather than relying on an exponential possibility distribution.

Definition 25

Let P be a possibilistic positive clausal program. We define the immediate consequence operator T^{w}_P as:

$$T^{\mathrm{w}}_P(E)(e_0) = \max\big\{\lambda \in [0,1] \mid (e_0 \leftarrow e_1, ..., e_m) \in P_\lambda \\ \text{and } \forall i \in \{1, ..., m\} \cdot E^\lambda \models e_i\big\}.$$

We use $P^\star_{\mathrm{w}}$ to denote the fixpoint which is obtained by repeatedly applying T^{w}_P starting from the empty set $E = \emptyset$, i.e. the least fixpoint of T^{w}_P w.r.t. set inclusion. When P is a positive clausal program we take $\lambda \in \{0,1\}$.

Example 38

Consider the clausal program P with the clausal rules

$$\begin{aligned} \mathbf{1}\colon & a \vee b \vee c \leftarrow \\ \mathbf{0.4}\colon & \neg b \leftarrow \\ \mathbf{0.8}\colon & e \leftarrow (a \vee c \vee d). \end{aligned}$$

We can easily verify that, starting from $E = \emptyset$, we obtain

$$T_P^{\mathrm{w}}(E)(a \vee b \vee c) = 1 \text{ and}$$
$$T_P^{\mathrm{w}}(E)(\neg b) = 0.4.$$

In the next iteration we furthermore find that

$$T_P^{\mathrm{w}}(T_P^{\mathrm{w}}(E))(e) = 0.4$$

since $(\mathbf{0.8}\!: e \leftarrow (a \vee c \vee d)) \in P_{0.4}$ and since $(T_P^{\mathrm{w}}(E))^{0.4} \models a \vee c \vee d$. In addition, this is the least fixpoint, i.e. we have $P_{\mathrm{w}}^{\star} = \left\{(a \vee b \vee c)^1, \neg b^{0.4}, e^{0.4}\right\}$.

Notice that this definition of the immediate consequence operator is a generalization of the immediate consequence operator for possibilistic simple programs (see Definition 9). Indeed, for a possibilistic positive clausal program where all clauses contain only a single literal, i.e. a possibilistic simple program, we have that $P^{\star} = P_{\mathrm{w}}^{\star}$. In addition, when all clauses contain only a single literal, we can simplify the immediate consequence operator and simply write $e_i \in E^{\lambda}$ instead of $E^{\lambda} \models e_i$.
We now show that the fixpoint obtained from the immediate consequence operator T_P^{w} is indeed the answer set of P.

Proposition 20

Let P be a possibilistic positive clausal program without possibilistic constraint rules. Then $P_{\mathrm{w}}^{\star}$ is a possibilistic answer set of P.

Thus far, we only considered possibilistic positive clausal programs. If we allow for negation-as-failure, we will also need to generalize the notion of a reduct. As usual, in the classical case we want that an expression of the form '$not\ e$' is true when 'e' cannot be entailed. Furthermore, since we are working in the possibilistic case, we want to take the degrees into account when determining the reduct.

Definition 26

Given a possibilistic clausal program P and a set of weighted clauses E, the reduct P^E of P w.r.t. E is defined as:

$$\begin{aligned} P^E = \{ \ &((e_0 \leftarrow e_1, ..., e_m), \min(\lambda_{rule}, \lambda_{body})) \ \mid \ \min(\lambda_{rule}, \lambda_{body}) > 0 \\ &\wedge \lambda_{body} = \max \left\{ \lambda \mid \forall i \in \{m+1, ..., n\} \cdot \underline{E^{1-\lambda}} \not\models e_i, \lambda \in [0,1] \right\} \\ &\wedge ((e_0 \leftarrow e_1, ..., e_m, not\ e_{m+1}, ..., not\ e_n), \lambda_{rule}) \in P \} \end{aligned}$$

This definition corresponds with the Gelfond-Lifschitz reduct when we consider crisp clausal programs where each clause consists of exactly one literal. Indeed, if we consider clauses with exactly one literal, we could simplify $\forall i \in \{m+1, ..., n\} \cdot \underline{E^{1-\lambda}} \not\models e_i$ to $\{e_{m+1}, ..., e_n\} \cap \underline{E^{1-\lambda}} = \emptyset$. This new reduct generalises the Gelfond-Lifschitz reduct in two ways. Firstly, we now have clauses, i.e. we now need to verify whether the negative body is not entailed by our guess. Secondly, we need to take the weights attached to the rules, which we interpret as certainties, into account. In particular, the certainty of the reduct of a rule is limited by the certainty of the negative body of the rule and the certainty of the rule itself. In the crisp case these certainty degrees would become trivial.

Proposition 21

A set of weighted clauses E is a possibilistic answer set of the possibilistic clausal program P without possibilistic constraint rules iff E is a possibilistic answer set of P^E.

Before we discuss the complexity results, we look at an example to further uncover the intuition of clausal programs.

Example 39

Consider the possibilistic clausal program P with the following rules:

$$\mathbf{0.7}\colon a \vee b \vee c \leftarrow \qquad \mathbf{0.2}\colon \neg b \leftarrow \qquad \mathbf{1}\colon d \leftarrow not\ (a \vee c \vee f) \qquad \mathbf{1}\colon e \leftarrow not\ c.$$

The reduct P^E with $E = \left\{ (a \vee b \vee c)^{0.7}, (\neg b)^{0.2}, d^{0.8}, e^1 \right\}$ is then:

$$\mathbf{0.7}\colon a \vee b \vee c \leftarrow \qquad \mathbf{0.2}\colon \neg b \leftarrow \qquad \mathbf{0.8}\colon d \leftarrow \qquad \mathbf{1}\colon e \leftarrow$$

since $E^{1-0.8} \models a \vee c$ but $E^{\underline{1-0.8}} \not\models a \vee c$ and $E^{\underline{1-1}} \not\models c$. We then have that $(P^E)^\star_{\mathrm{w}} = \left\{(a \vee b \vee c)^{0.7}, (\neg b)^{0.2}, d^{0.8}, e^1\right\}$, hence E is indeed an answer set of P.

4.4 Complexity Results of PASP

Before we discuss the complexity results of the weak possibilistic semantics for disjunctive rules (Section 4.3.2), we first look at the complexity results of both possibilistic normal programs (Section 4.2.2) and the strong possibilistic semantics for disjunctive rules (Section 4.3.1). We find that for possibilistic normal programs the addition of weights does not affect the complexity compared to classical normal programs.

Proposition 22: possibilistic normal program; brave reasoning

Let P be a possibilistic normal program. The problem of deciding whether there exists a possibilistic answer set V of P such that $V(l) \geq \lambda$ is NP-complete.

Proposition 23: possibilistic normal program; cautious reasoning

Let P be a possibilistic normal program. The problem of deciding whether for all possibilistic answer sets V of P we have that $V(l) \geq \lambda$ is coNP-complete.

Similarly, we find for possibilistic disjunctive programs under the strong disjunctive semantics that the addition of weights does not affect the complexity compared to classical disjunctive programs.

Proposition 24: possibilistic disjunctive program; brave reasoning

Let P be a possibilistic disjunctive program. The problem of deciding whether there is a possibilistic answer set V such that $V(l) \geq \lambda$ is a Σ_2^{P}-complete problem.

Proposition 25: possibilistic disjunctive program; cautious reasoning

Let P be a possibilistic disjunctive program. The problem of deciding whether for all possibilistic answer sets V we have that $V(l) \geq \lambda$ is a Π_2^{P}-complete problem.

We now look at the complexity of the weak possibilistic semantics for disjunctive rules for a variety of decision problems and under a variety of restrictions. In particular, throughout this section we look at the complexity of weak disjunction in the crisp case that allows us to compare these results against the complexity of the related decision problems in classical ASP and other epistemic extensions of ASP, e.g. [Truszczyński 2011, Vlaeminck et al. 2012]. As we will see, for certain classes of clausal programs, decision problems exist where weak disjunction is computationally less complex than disjunctive programs while remaining more complex than normal programs.

An overview of the complexity results available in the literature for disjunctive programs as well as the new results for weak disjunction (in the crisp case) which we discuss in the remainder of this section can be found in Table 4.1.

Table 4.1: Completeness results for the main reasoning tasks with references

no NAF, no $\neg$	existence	brave reasoning	cautious reasoning
strong disjunction	NP [(1)]	Σ_2^P [(1)]	coNP [(1)]
weak disjunction	P [(6)]	P [(6)]	P [(6)]
no NAF, $\neg$	**existence**	**brave reasoning**	**cautious reasoning**
strong disjunction	NP [(1)]	Σ_2^P [(1)]	coNP [(1)]
weak disjunction	NP [(4)]	BH_2 [(3)]	coNP [(5)]
NAF, $\neg$	**existence**	**brave reasoning**	**cautious reasoning**
strong disjunction	Σ_2^P [(2)]	Σ_2^P [(2)]	Π_2^P [(2)]
weak disjunction	Σ_2^P [(8)]	Σ_2^P [(7)]	Π_2^P [(9)]

“no NAF” (resp. “no $\neg$”) indicates results for programs without negation-as-failure (resp. classical negation)

(1) [Eiter and Gottlob 1993]
(2) [Baral 2003]
(3) Proposition 26 and 27
(4) Corollary 5
(5) Corollary 6
(6) Proposition 28
(7) Proposition 29 and 30
(8) Corollary 8
(9) Corollary 31

Proposition 26: weak disjunction, positive clausal program; brave reasoning

Let P be a positive clausal program. The problem of deciding whether a clause ‘e’ is entailed by a consistent answer set E of P is BH_2-hard.

Proposition 27: weak disjunction, positive clausal program; brave reasoning

Let P be a positive clausal program. The problem of deciding whether a clause 'e' is entailed by a consistent answer set E of P is in BH_2.

Corollary 4

Let P be a positive clausal program. The problem of deciding whether a clause 'e' is entailed by a consistent answer set E of P is BH_2-complete.

Corollary 5: weak disjunction, positive clausal program; answer set existence

Determining whether a positive clausal program P has a consistent answer set is an NP-complete problem.

Corollary 6: weak disjunction, positive clausal program; cautious reasoning

Cautious reasoning, i.e. determining whether a clause 'e' is entailed by every answer set E of a positive clausal program P is coNP-complete.

Surprisingly, the expressivity of positive clausal programs under the weak interpretation of disjunction is directly tied to the ability to use classical negation in clauses. If we limit ourselves to positive clausal programs without classical negation we find that the expressiveness is restricted to P.

In order to see this, let us take a closer look at the immediate consequence operator for clausal programs as defined in Definition 25. When there are no occurrences of classical negation we can simplify this immediate consequence operator to

$$T_P^{\mathrm{w}}(E) = \{e_0 \mid e_0 \leftarrow e_1, ..., e_m \in P \wedge \forall i \in \{1, ..., m\} \cdot \exists e \in E \cdot e \subseteq e_i\}$$

where $e \subseteq e_i$ is defined as the subset relation where we interpret e and e_i as sets of literals, i.e. $e = (l_1 \vee ... \vee l_n)$ is interpreted as $\{l_1, ..., l_n\}$.

Proposition 28

Let P be a positive clausal program without classical negation. We can find the unique answer set of P in polynomial time.

We now examine the complexity of general clausal programs. We will do this by showing that the problem of determining the satisfiability of a QBF of the form $\phi = \exists X_1 \forall X_2 \cdot p(X_1, X_2)$ with $p(X_1, X_2)$ in DNF can be reduced to the problem of determining whether a clause 'e' is entailed by a consistent answer set M of the clausal program P. We start with the definition of our reduction.

Definition 27

Let $\phi = \exists X_1 \forall X_2 \cdot p(X_1, X_2)$ be a QBF with $p(X_1, X_2) = \theta_1 \vee ... \vee \theta_n$ a formula in disjunctive normal form with X_i sets of variables. We define the clausal program P_ϕ corresponding to ϕ as

$$\begin{aligned} P_\phi = &\{x \leftarrow not\ \neg x \mid x \in X_1\} \cup \{\neg x \leftarrow not\ x \mid x \in X_1\} && (4.12)\\ &\cup \{\neg\theta_t \vee sat \leftarrow \mid 1 \leq t \leq n\} && (4.13)\\ &\cup \{\leftarrow not\ sat\} && (4.14)\end{aligned}$$

with $\neg\theta_t$ the clausal representation of the negation of the formula θ_t, e.g. when $\theta_t = x_1 \wedge \neg x_2 \wedge ... \wedge \neg x_k$ then $\neg\theta_t = \neg x_1 \vee x_2 \vee ... \vee x_k$.

Example 40

Given the QBF $\phi = \exists p_1, p_2 \forall q_1, q_2 \cdot (p_1 \wedge q_1) \vee (p_2 \wedge q_2) \vee (\neg q_1 \wedge \neg q_2)$ the clausal program P_ϕ is

$$\begin{aligned} p_1 &\leftarrow not\ \neg p_1\\ \neg p_1 &\leftarrow not\ p_1\\ p_2 &\leftarrow not\ \neg p_2\\ \neg p_2 &\leftarrow not\ p_2\\ \neg p_1 \vee \neg q_1 \vee sat &\leftarrow\\ \neg p_2 \vee \neg q_2 \vee sat &\leftarrow\\ q_1 \vee q_2 \vee sat &\leftarrow\\ &\leftarrow not\ sat.\end{aligned}$$

Notice how $M = \{p_1, p_2, \neg p_1 \vee \neg q_1 \vee sat, \neg p_2 \vee \neg q_2 \vee sat, q_1 \vee q_2 \vee sat\}$ is an answer set of P_ϕ and that $M \models sat$. Accordingly we find that the QBF is satisfied.

If we take the QBF $\phi' = \exists p_1, p_2 \forall q_1, q_2 \cdot (p_1 \wedge q_1) \vee (p_2 \wedge q_2)$ then the clausal program $P_{\phi'}$ corresponding to ϕ' is the program P_ϕ in which the penultimate rule has been removed. Notice how $P_{\phi'}$ has no answer sets, because we are not able to entail 'sat' from any of the models of $P_{\phi'}$. Indeed, the QBF ϕ' is not satisfiable.

Proposition 29: weak disjunction; brave reasoning

Let P be a clausal program. The problem of deciding whether a clause 'e' is entailed by a consistent answer set E of P is Σ_2^{P}-hard.

Proposition 30: weak disjunction; brave reasoning

Let P be a clausal program. The problem of deciding whether a clause 'e' is entailed by a consistent answer set E of P is in Σ_2^{P}.

Corollary 7

Let P be a clausal program. The problem of deciding whether a clause 'e' is entailed by a consistent answer set E of P is Σ_2^{P}-complete.

Corollary 8: weak disjunction; answer set existence

Determining whether a clausal program P has a consistent answer set is an Σ_2^{P}-complete problem.

Proposition 31: weak disjunction; cautious reasoning

Cautious reasoning, i.e. determining whether a clause 'e' is entailed by every answer set E of a clausal program P, is Π_2^{P}-complete.

4.5 Work Related to PASP$_Ł$

The work presented in this chapter touches on various topics that have been the subject of previous research. Since we will continue our work on PASP in Chapter 5, and since a lot of work is related to both this chapter and Chapter 5, we postpone some of our discussions w.r.t. work related to PASP$_Ł$, and PASP in general, to Section 5.4. In this section we focus our discussion along two main lines which are mainly relevant to PASP$_Ł$. In Section 4.5.1

previous work on the semantics of disjunctive programs is discussed. In Section 4.5.2 we look at prior work on characterizing rules with possibility theory and fuzzy logic.

4.5.1 Semantics of Disjunctive Programs

Several authors have already proposed alternatives and extensions to the semantics of disjunctive programs. Ordered disjunction [Brewka 2002] falls in the latter category and allows to use the head of the rule to formulate alternative solutions in their preferred order. For example, a rule such as $l_1 \times ... \times l_k \leftarrow$ represents the knowledge that l_1 is preferred over l_2 which is preferred over l_3 ..., but that at the very least we want l_k to be true. As such it allows for an easy way to express context dependent preferences. The semantics of ordered disjunction allow certain non-minimal models to be answer sets, hence, unlike the work in this chapter, it does not adhere to the standard semantics of disjunctive rules in ASP.

Annotated disjunctions are another example of a framework that changes the semantics of disjunctive programs [Vennekens et al. 2004]. It is based on the idea that every disjunct in the head of a rule is annotated with a probability. Interestingly, both ordered and annotated disjunction rely on split programs, as found in the possible model semantics [Sakama and Inoue 1994]. These semantics provide an alternative to the minimal model semantics. The idea is to split a disjunctive program into a number of normal programs, one for each possible choice of disjuncts in the head, of which the minimal Herbrand models are then the possible models of the disjunctive programs. Intuitively this means that a possible model represents a set of atoms for which a possible justification is present in the program. In line with our results for weak disjunction, using the possible model semantics also leads to a lower computational complexity.

Not all existing extensions of disjunction allow non-minimal models. For example, in [Buccafurri et al. 2002] an extension of disjunctive logic programs is presented which adds the idea of inheritance. Conflicts between rules are resolved in favour of more specific rules. Such an approach allows for an intuitive way to deal with default reasoning and exceptions. In particular, the semantics allow for rules to be marked as being defeasible and allows to specify an order or inheritance tree among (sets of) rules. Interestingly, the complexity of the resulting system is not affected and coincides with the complexity of ordinary disjunctive programs.

4.5.2 Characterization Rules using Multi-Valued Logics

A large collection of research has focused on how possibility distributions can be used to assign a meaning to rules. For example, possibility theory has been used to model default rules [Benferhat et al. 1992, Benferhat et al. 1997]. Specifically, a default rule "if a then b" is interpreted as $\Pi(a \wedge b) > \Pi(a \wedge \neg b)$, which captures the intuition that when a is known to hold, b is more plausible than $\neg b$, if all that is known is that a holds. In this approach entailment is defined by looking at the least specific possibility distributions which is similar in spirit to our approach for characterizing ASP rules (although the notion of least specific possibility distribution is defined, in this context, w.r.t. the plausibility ordering on interpretations induced by the possibility degrees).

The work on possibilistic logic [Dubois et al. 1994], as discussed in Section 2.4, forms the basis of possibilistic logic programming [Dubois et al. 1991]. The idea of possibilistic logic programming is to start from a necessity-valued knowledge base, which is a finite set of pairs $(\phi\ \alpha)$, called necessity-valued formulas, with ϕ a closed first-order formula and $\alpha \in [0, 1]$. Semantically, a necessity-valued formula expresses a constraint of the form $N(\phi) \geq \alpha$ on the set of possibility distributions. A possibilistic logic program is then a set of necessity-valued implications. As rules are essentially modelled using material implication, however, the stable model semantics cannot straightforwardly be characterized using possibilistic logic programming. For example, the knowledge base $\{(a \to b\ \ 1), (\neg b\ \ 1)\}$, which represents the program $\{b \leftarrow a, \neg b \leftarrow\}$, induces that $N(\neg a) = 1$. Indeed, the semantics of this knowledge base indicate that $\Pi(a \wedge \neg b) = 0$ and $\Pi(b) = 0$, i.e. we find that $\Pi(a) = 0$. In other words: a direct encoding using possibilistic logic programming allows for contraposition, which is not in accordance with the stable model semantics.

Rules in logic can also be interpreted as statements of conditional probability [Jaynes 2003]. In the possibilistic setting this notion has been adapted to the notion of conditional necessity measures. Rules can then also be modelled in terms of conditional necessity measures [Benferhat et al. 1997, Dubois and Prade 1997, Benferhat et al. 2002]. The conditional possibility measure $\Pi(\psi \mid \phi)$ is defined as the greatest solution to the equation $\Pi(\phi \wedge \psi) = \min(\Pi(\psi \mid \phi), \Pi(\phi))$ in accordance with the principle of least specificity. It can be derived mathematically that $\Pi(\psi \mid \phi) = 1$ if $\Pi(\psi \wedge \phi) = \Pi(\phi)$ and $\Pi(\psi \mid \phi) = \Pi(\psi \wedge \phi)$ otherwise whenever $\Pi(\phi) > 0$. When $\Pi(\phi) = 0$, then by convention $\Pi(\psi \mid \phi) = 1$ for every $\psi \neq \bot$ and $\Pi(\bot \mid \phi) = 0$, otherwise. The conditional necessity measure is defined as $N(\psi \mid \phi) = 1 - \Pi(\neg\psi \mid \phi)$. However, there does not seem to be a straightforward way to capture the stable model semantics using conditional necessity measures, especially when classical negation is allowed. For example, consider the program $\{b \leftarrow a, \neg a \leftarrow\}$, which has the corresponding constraints $N(b \mid a) \geq 1$ and $N(\neg a \mid \top) \geq 1$. Using the definition of the conditional necessity measure, the first constraint is equivalent to $1 - \Pi(\neg b \mid a) \geq 1$, i.e. $\Pi(\neg b \mid a) = 0$. The second constraint simplifies to $\Pi(a) = 0$, which, using the convention stated above gives rise to $\Pi(\neg b \mid a) = 1$, clearly a contradictory result to the earlier conclusion that $\Pi(\neg b \mid a) = 0$. The semantics presented in this chapter avoid both these contradictions and the problem of contraposition, yet still characterize ASP using the machinery of possibilistic logic. They thus benefit from the capabilities of possibilistic logic to reason about uncertain information, while being in accordance with the stable model semantics.

Possibility theory, which can e.g. be used for belief revision, has a strong epistemic notion and shares a lot of commonalities with epistemic entrenchments [Dubois and Prade 1991]. Furthermore, in [Dubois et al. 2012] a generalization of possibilistic logic is studied, which corresponds to a weighted version of a fragment of the modal logic KD. In this logic, epistemic states are represented as possibility distributions, and logical formulas are used to express constraints on possible epistemic states. In this chapter, we similarly interpret rules in ASP as constraints on possibility distributions, which furthermore allows us to unearth the semantics of weak disjunction.

Possibility theory has also been used to define various semantics of certain versions of fuzzy if-then rules [Zadeh 1992]. Rather than working with literals, fuzzy if-then rules can consider fuzzy predicates which each have their own universe of discourse. To draw conclusions from a set of fuzzy if-then rules, mechanisms are needed that can produce an (intuitively acceptable) conclusion from a set of such rules.

4.6 Summary

In this chapter we defined new semantics for PASP, a framework that combines possibility theory and ASP to allow for reasoning under (qualitative) uncertainty. These semantics, called $\text{PASP}_{Ł}$, are based on the interpretation of possibilistic rules as constraints on possibility distributions. We showed how $\text{PASP}_{Ł}$ differs from the existing semantics for PASP, referred to as PASP_{G}. Specifically, $\text{PASP}_{Ł}$ adheres to a different intuition for negation-as-failure. As such, these semantics can be used to arrive at acceptable results for problems where the possibilistic answer sets according to PASP_{G} do not necessarily agree with our intuition of the problem. In addition, we showed how $\text{PASP}_{Ł}$ allowed for a new characterization of ASP. When looking at ASP as a special case of PASP, we naturally recover the intuition that the head of a rule is certain whenever we are certain that the body holds. The resulting characterization stays close to the intuition of the stable model semantics, yet also shares the explicit reference to modalities with autoepistemic logic. We showed that this characterization not only naturally characterizes normal programs, i.e. programs with negation-a-failure, but can also naturally characterize disjunctive programs and programs with classical negation.

Due to our explicit reference to modalities in the semantics, we are furthermore able to characterize an alternative semantics for disjunction in the head of a rule that has a more epistemic flavour than the standard treatment of disjunction in ASP, i.e. given a rule of the form $(a \vee b \leftarrow)$ we do not obtain two answer sets, but rather we have

'$a \vee b$' as-is in the answer set. While such a characterization might seem weak, we showed that the interplay with literals significantly affects the expressiveness. Indeed, we found that the problem of brave reasoning/cautious reasoning under these weak semantics for disjunction for a program without negation-as-failure, but with classical negation, is BH_2-complete and coNP-complete, respectively. This highlights that weak disjunction is not merely syntactic sugar, i.e. it cannot simply be simulated in normal ASP without causing an exponential blow-up. For strong disjunction, on the other hand, we have obtained that brave and cautious reasoning without negation-as-failure are Σ_2^{P}-complete and coNP-complete, respectively. As such, the weak semantics for disjunction detailed in this chapter allow us to work with disjunction in a less complex way that still remains non-trivial. If, however, we restrict ourselves to atoms, then brave reasoning under the weak semantics for disjunction is P-complete.

5 | Possibilistic ASP: uncertain rules versus rules with uncertain conclusions

5.1 Introduction

In the previous chapter we introduced $\text{PASP}_{Ł}$, new semantics for PASP which are based on interpreting possibilistic rules as constraints on possibility distributions. We showed how these new semantics are different from PASP_{G} in how they view negation-as-failure. Furthermore, we highlighted how $\text{PASP}_{Ł}$ is firmly rooted in ASP. Indeed, we showed how a characterization of ASP can be obtained by looking at ASP as a special case of $\text{PASP}_{Ł}$ in which all rules are certain and no uncertainty is allowed in the answer sets.

While the intuition of PASP_{G} and $\text{PASP}_{Ł}$ is clearly different, both semantics coincide for simple programs. This is because both semantics interpret the weight as an assessment of the certainty that the conclusion of the rule holds, given that the body is known to hold. This, however, is not the only way in which such a weight can be interpreted. In fact, two natural motivations can be envisaged for introducing degrees of uncertainty in the setting of ASP. On the one hand, we may wish to model weighted epistemic states, in which an agent can be completely certain about the truth of some literals, while believing in the truth of other literals without complete certainty. This is the approach taken in PASP_G and $\text{PASP}_\text{Ł}$. On the other hand, we may wish to keep epistemic states Boolean, but rather express that the rules which constrain the possible epistemic states are not fully certain. In the latter case, an ASP program should correspond to a weighted set of classical answer sets.

In this chapter, we develop semantics for PASP based on the notion that the weight represents the uncertainty of the rule. Such an interpretation of uncertainty is quite natural. Indeed, we are often uncertain whether the rules we encode are actually reliable, e.g. when they are coming from possibly unreliable sources. We will refer to these semantics as PASP_r, where the 'r' signifies that we are considering uncertain rules rather than uncertain conclusions. In this chapter, we furthermore compare and contrast both motivations, i.e. we evaluate PASP_r against PASP_G and $\text{PASP}_\text{Ł}$.

As an example, and to further clarify the difference in intuition, let us consider the following program:

$$\begin{aligned}
\mathbf{0.7}&: paper\_title(title) \leftarrow \\
\mathbf{0.9}&: author(John\_Doe) \leftarrow paper\_title(title) \\
\mathbf{0.2}&: author(Jane\_Roe) \leftarrow paper\_title(title) \\
\mathbf{1}&: \leftarrow author(John\_Doe), author(Jane\_Roe).
\end{aligned}$$

This program encodes that, during a conference, a colleague shares the title of an interesting paper with us. We are quite certain that we recall the name of the title correctly and we would like to find out who the principal author of the paper is. We consult the university website, which in the past has given reliable answers. However, a quick search on the internet results in a different principal author for the same paper. Evidently, they cannot both be the principal author of the paper.

The uncertainty attached to each rule now expresses how certain we are that the information encoded in the rule is indeed valid. In particular, any world in which both John Doe and Jane Roe are the principal author of the paper can immediately be discarded due to the absolute certainty of the last rule. We do, however, acknowledge that neither of the candidate authors may be correct because we do not have absolute certainty as to whether we correctly recall the title of the paper. Of the two remaining rules, we have the most confidence in the rule that identifies John Doe as the author. Thus, we expect that the conclusion that the actual principal author of the paper is John Doe can be deduced with a higher certainty than the conclusion that the principal author is Jane Roe. Semantics that agree with this intuition, i.e. that agree with the idea of uncertain rules rather than uncertain conclusions, are discussed in Section 5.2. Notice, furthermore, that both PASP$_G$ and PASP$_Ł$ are inadequate to solve the aforementioned problem. Indeed, In neither PASP$_G$ nor PASP$_Ł$ does the above program have any answer set since we deduce both $author(John\_Doe)$ and $author(Jane\_Roe)$ with some certainty, which conflicts with the constraint rule $(\mathbf{1}\colon \leftarrow author(John\_Doe), author(Jane\_Roe))$.

The remainder of this chapter is organized as follows. In Section 5.2 we present semantics for PASP in which we treat weights as an expression of confidence in the validity of a rule. When such a PASP program has no possibilistic answer sets, we might intuitively want to exclude the least reliable rules in order to still reach a conclusion. Indeed, if we have e.g. $(\mathbf{0.8}\colon a \leftarrow)$ and $(\mathbf{0.2}\colon \neg a \leftarrow)$ then it seems more intuitive to conclude that 'a' is true (to some degree) rather than that '$\neg a$' is true. This is the approach taken in possibilistic logic. Due to the non-monotonicity of ASP, however, we cannot simply choose to omit the least certain rules to derive the most certain conclusions. Indeed, as we will see, both including invalid rules and excluding valid rules may lead to errors. For example, if we have the additional rule $(\mathbf{1}\colon b \leftarrow not\ \neg a)$ then the exclusion of $(\mathbf{0.2}\colon \neg a \leftarrow)$ would allow us to deduce 'b', where previously we were not able to conclude 'b'. As such, we need to consider all situations in which some, none, or all of the least certain rules are omitted. This naturally leads to the idea of an ASP program with optional rules. Given such a program with optional rules, we want to determine whether particular conclusions hold irrespective of the inclusion of the optional rules. In addition, we show how the new semantics require us to define new main reasoning tasks. In Section 5.3 we show how some of these reasoning tasks can be applied to the much wider problem range of programs with optional rules. In particular, we show how two interesting problems in AI, namely cautious abductive reasoning and conformant planning, can be expressed in terms of programs with optional rules. In Chapter 6, we will provide simulations using ASP for PASP$_r$. As such, we will obtain an implementation for these AI problems.

5.2 Semantics & Complexity Results of Uncertain Rules

In PASP$_G$ and PASP$_Ł$ the weight associated with a rule is interpreted as the certainty with which we can deduce the head when the body is known to hold. As such, we obtain semantics based on weighted epistemic states, where we relate exactly one possibility distribution with each possibilistic answer set of the program. We can, however, look at these certainties in another way. Rather than considering weighted epistemic states, we can use Boolean epistemic states and use the certainties associated with the rules to express that some epistemic states are more plausible than others. We thus look at the weight associated with a rule as expressing our uncertainty as to whether the rule is valid. Indeed, the information encoded in the various rules may e.g. come from different sources and this may affect the degree to which we believe the rule to be valid.

Example 41

Consider the program P with the rules:

$$\begin{aligned}\mathbf{0.2}\colon & \mathit{raining} \leftarrow \\ \mathbf{0.9}\colon & \mathit{slippery} \leftarrow \mathit{raining} \\ \mathbf{0.7}\colon & \mathit{safe} \leftarrow \mathit{not\ slippery}.\end{aligned}$$

We will use this program to clarify the semantics proposed in this chapter. Intuitively, the program encodes the knowledge that it is raining, that raining makes the floor slippery and that a floor which is not slippery is safe to walk on without risk of injury.

Clearly, if we incorrectly consider a rule to be valid we may draw incorrect conclusions. The usual strategy, which is adopted in e.g. possibilistic logic, is to discard the least reliable pieces of knowledge when we want to ensure that what we derive is reliable. However, such a strategy would not work in ASP due to its non-monotonic nature. Indeed, while failing to discard incorrect rules may lead to erroneous conclusions, the same may happen when we incorrectly discard a valid rule. For example, we may choose to omit the rule that encodes the information that it is raining because we believe that this rule is insufficiently reliable. However, as a result, we are not able to conclude that the floor is slippery. This, in turn, enables us to derive that it is safe to walk on the floor, i.e. the omission of a rule allowed us to derive additional information. Hence, to assess the certainty with which a literal can be derived, we need to consider all the subprograms of a given program, including the complete program itself. Some of these subprograms are more likely than others to correspond with an accurate representation of the considered problem. An answer set is then said to be necessary to the extent that it is an answer set of all the plausible subprograms. Similarly, we say that an answer set is possible to the extent that it is an answer set of some plausible subprogram.

Each subprogram corresponds with the assumption that some particular rules are wrong, namely those rules from the program that are missing, while the rules in the subprogram are assumed to be correct (in particular, we can also assume all rules to be correct or all rules to be wrong). Ideally, we thus want to associate a possibility degree with each subprogram, i.e. the degree to which we believe that this subprogram consists exactly of the valid rules. In practice, however, it is not feasible to list all subprograms and associate a possibility with each individual subprogram. Instead, we encode a possibility distribution over subprograms by associating a certainty with each rule in our possibilistic program P. The possibility of each subset is then determined by looking at the certainties of the rules that are omitted from the program. If we omit a rule with a certainty of 1, i.e. a rule of which we are certain that it is valid, then the rules in the subprogram can never be the set of all rules that are valid. Thus, our possibility that the rules in the subprogram correspond with the set of valid rules is 0. Conversely, if we only omit rules with a low certainty, then we retain a high possibility that the rules in our subprogram are exactly those rules that are valid. In particular, in the above example, omitting only the first rule would result in a subprogram with a high possibility of 0.8. However, omitting the second rule with a certainty of 0.9 would result in a subprogram which contains the valid rules with a possibility of 0.1.

Following this line of reasoning, we conceptually need a possibility distribution over subprograms of P. Specifically, we interpret a possibilistic rule (r, c) as the constraint $N(r) \geq c$, where $N(r)$ stands for $N(\{P' \mid P' \subseteq P$ and P' contains the rule $r\})$. The possibility distribution π_P is then the least specific possibility distribution that satisfies these constraints. Thus, a subprogram is considered possible to the extent that it contains all of the certain rules.

Definition 28

Let P be a PASP program. We define the possibility distribution π_P over the subsets $P' \subseteq P$ as

$$\pi_P(P') = \begin{cases} 1 - \max\{c \mid (r,c) \in P \setminus P'\} & \text{when } P'^{*} \text{ consistent} \\ 0 & \text{otherwise} \end{cases}$$

Intuitively, this definition states that the less certain the rules are that we omit from the subprogram P', the more possible it is that P' is the correct program. It is not hard to see that this definition corresponds with the least specific possibility distribution that satisfies the constraints $N(r) \geq c$ for every $(r, c) \in P$, with the additional constraint that inconsistent programs are impossible. Indeed, we have that:

$$\begin{aligned}
&\forall (r,c) \in P \cdot N(r) \geq c \\
&\equiv \forall (r,c) \in P \cdot N(\{P' \mid P' \subseteq P \text{ and } (r,c) \in P'\}) \geq c \\
&\equiv \forall (r,c) \in P \cdot \Pi(\{P' \mid P' \subseteq P \text{ and } (r,c) \in (P \setminus P')\}) \leq 1 - c \\
&\equiv \forall (r,c) \in P \cdot \max\{\pi(P') \mid P' \subseteq P, (r,c) \in (P \setminus P')\} \leq 1 - c \\
&\equiv \forall (r,c) \in P \cdot \forall P' \subseteq P, (r,c) \in (P \setminus P') \cdot \pi(P') \leq 1 - c \\
&\equiv \forall P' \subseteq P \cdot \pi(P') \leq \min\{1 - c \mid (r,c) \in (P \setminus P')\} \\
&\equiv \forall P' \subseteq P \cdot \pi(P') \leq 1 - \max\{c \mid (r,c) \in (P \setminus P')\} \\
&\equiv \forall P' \subseteq P \cdot \pi(P') \leq \pi_P(P')
\end{aligned}$$

Notice furthermore that π_P is a normalized possibility distribution whenever P^* is consistent since then $\pi_P(P) = 1$.

Example 42

Consider the program P from Example 41. For compactness, we name the rules r_1, r_2 and r_3 from top to bottom. The possibility distribution π over the subprograms of P is defined as:

$$
\begin{aligned}
\pi(\{r_1, r_2, r_3\}) &= 1 & \pi(\{r_2, r_3\}) &= 0.8 \\
\pi(\{r_1, r_3\}) &= 0.1 & \pi(\{r_3\}) &= 0.1 \\
\pi(\{r_1, r_2\}) &= 0.3 & \pi(\{r_2\}) &= 0.3 \\
\pi(\{r_1\}) &= 0.1 & \pi(\{\}) &= 0.1
\end{aligned}
$$

We now define the main reasoning tasks for these new semantics.

Definition 29

Let P be a PASP program. Let π_P be as in Definition 28. The degree to which it is possible that 'l' is a brave/cautious consequence of P is defined as:

$$
\begin{aligned}
\Pi\left(P \models^{\mathrm{b}} l\right) &= \max\left\{\pi_P(P') \mid P' \subseteq P \text{ and } P'^* \models^{\mathrm{b}} l\right\} \\
\Pi\left(P \models^{\mathrm{c}} l\right) &= \max\left\{\pi_P(P') \mid P' \subseteq P \text{ and } P'^* \models^{\mathrm{c}} l\right\}
\end{aligned}
$$

i.e. this is the degree to which some program $P' \subseteq P$ is possible which has 'l' as a brave/cautious consequence. The degree to which it is necessary that P has 'l' as a brave/cautious consequence is defined as:

$$
\begin{aligned}
N\left(P \models^{\mathrm{b}} l\right) &= 1 - \max\left\{\pi_P(P') \mid P' \subseteq P \text{ and } P'^* \not\models^{\mathrm{b}} l\right\} \\
N\left(P \models^{\mathrm{c}} l\right) &= 1 - \max\left\{\pi_P(P') \mid P' \subseteq P \text{ and } P'^* \not\models^{\mathrm{c}} l\right\}.
\end{aligned}
$$

Note that this is the degree to which all programs $P' \subseteq P$ that do not have 'l' as a brave/cautious consequence are impossible.

In the remainder of this chapter, we will also write $P \models^{\mathrm{b}}_{\Pi} l^{\lambda}$ to denote that $\Pi\left(P \models^{\mathrm{b}} l\right) \geq \lambda$, and similar for the notations $P \models^{\mathrm{c}}_{\Pi} l^{\lambda}$, $P \models^{\mathrm{b}}_{N} l^{\lambda}$ and $P \models^{\mathrm{c}}_{N} l^{\lambda}$.
The differences between these types of inference are shown in the next example.

Example 43

Consider the PASP program P with the rules:

$$\mathbf{0.8}\colon b \leftarrow \textit{not } c \qquad \mathbf{0.3}\colon c \leftarrow d, \textit{not } b \qquad \mathbf{0.9}\colon d \leftarrow .$$

We have that

$$\begin{array}{lr}
\pi_P(P) = 1 & \{b,d\}, \{c,d\} \\
\pi_P(\mathbf{0.8}\colon b \leftarrow \textit{not } c; \mathbf{0.9}\colon d \leftarrow) = 0.7 & \{b,d\} \\
\pi_P(\mathbf{0.3}\colon c \leftarrow d, \textit{not } b; \mathbf{0.9}\colon d \leftarrow) = 0.2 & \{c,d\} \\
\pi_P(\mathbf{0.9}\colon d \leftarrow) = 0.2 & \{d\} \\
\pi_P(\mathbf{0.8}\colon b \leftarrow \textit{not } c; \mathbf{0.3}\colon c \leftarrow d, \textit{not } b) = 0.1 & \{b\} \\
\pi_P(\mathbf{0.8}\colon b \leftarrow \textit{not } c) = 0.1 & \{b\} \\
\pi_P(\mathbf{0.3}\colon c \leftarrow d, \textit{not } b) = 0.1 & \{\} \\
\pi_P(\{\}) = 0.1 & \{\}.
\end{array}$$

where the possibility associated with each subprogram is shown on the left and the classical answer set(s) of each subprogram is shown on the right. We obtain the following conclusions:

$$\begin{array}{ll}
P \models^{\mathrm{b}}_{N} \{b^{0.8}, c^{0.3}, d^{0.9}\} & P \models^{\mathrm{c}}_{N} \{b^{0}, c^{0}, d^{0.9}\} \\
P \models^{\mathrm{b}}_{\Pi} \{b^{1}, c^{1}, d^{1}\} & P \models^{\mathrm{c}}_{\Pi} \{b^{0.7}, c^{0.2}, d^{1}\}.
\end{array}$$

When the particular understanding of negation-as-failure in PASP$_{\mathrm{G}}$ prevents us from obtaining intuitive results, both the approach presented in this section as well as PASP$_{\text{Ł}}$ can be used to obtain more satisfactory results. We illustrate this in the next example.

Example 44

Consider the following possibilistic normal program P, which describes the operations of a computer system:

$$\begin{aligned}
&\mathbf{0.1}\colon \textit{normal} \leftarrow \\
&\mathbf{1.0}\colon \textit{abnormal} \leftarrow \textit{not normal} \\
&\mathbf{0.8}\colon \textit{problematic} \leftarrow \textit{abnormal}.
\end{aligned}$$

The intuition of this program is different depending on the semantics that are used. In both PASP$_\text{G}$ and PASP$_\text{Ł}$ the intuition of the first and last rule is the same. Indeed, the first rule describes that we have a low certainty that the system is operating normally. The last rule encodes that we are fairly certain that the system is operating problematically whenever the system is operating abnormally. However, the intuition of the second rule does differ significantly. Indeed, in PASP$_\text{G}$ the second rule would read *"the system is operating abnormally, unless there is some certainty that the system is operating normally"*. Under PASP$_\text{Ł}$, however, the second rule would be understood as *"the system is operating abnormally to the degree that the system is not working normally"*.

Under the new semantics presented in this chapter, which we refer to as PASP$_\text{r}$, the intuition of the rules changes slightly. For example, we are certain with a fairly high degree that the last rule, which describes that an abnormal system is behaving problematically, is indeed valid. In PASP$_\text{Ł}$, one would expect a conclusion in which we deduce with a high certainty that the system will give problematic errors. Indeed, the second rule states that we will assume that the system is working abnormally, unless we are very certain that the system is working normally, i.e. we act cautiously. Using the last rule, we then obtain with a high certainty that the system will cause problematic behaviour. The results obtained by the different semantics are:

semantics from PASP$_\text{G}$	$\{normal^{0.1}\}$
semantics from PASP$_\text{Ł}$	$\{normal^{0.1}, abnormal^{0.9}, problematic^{0.8}\}$
semantics from PASP$_\text{r}$	$P \models^{\text{c}}_{N} \{normal^{0.1}, abnormal^{0}, problematic^{0}\}$
	$P \models^{\text{b}}_{N} \{normal^{0.1}, abnormal^{0}, problematic^{0}\}$
	$P \models^{\text{c}}_{\Pi} \{normal^{1}, abnormal^{0.9}, problematic^{0.9}\}$
	$P \models^{\text{b}}_{\Pi} \{normal^{1}, abnormal^{0.9}, problematic^{0.9}\}$

Note that in PASP$_\text{G}$, because the certainty is ignored when determining the reduct, '$normal$' is assumed to be true without doubt. Hence, the second rule is removed from the reduct and we have no way of concluding that the system is performing abnormally. In PASP$_\text{Ł}$, however, the certainty is taken into consideration and we conclude, with a fairly high certainty, that '$abnormal$' is true. The semantics proposed in this chapter, for this given example, agree with PASP$_\text{G}$ when we are interested in the inference based on cautious necessity. On the other hand, the conclusions are closer to those of PASP$_\text{Ł}$ when we look at the inference based on brave possibility.

Furthermore, note that since each subprogram has a unique answer set both brave and cautious reasoning coincide.

In general though, neither of the semantics for PASP need to agree with each other, as can be seen in the next example:

Example 45

Consider the PASP program P with the rules:

$$\begin{aligned} \mathbf{1} &: lost \leftarrow not\ visible \\ \mathbf{1} &: visible \leftarrow not\ hidden \\ \mathbf{0.5} &: hidden \leftarrow \end{aligned}$$

For easy reference, we name these rules from top to bottom r_1, r_2 and r_3. Intuitively, this example describes a simple game where an agent loses when he cannot see an object. However, there is uncertainty as to whether or not the object itself is hidden. We have that:

$$\begin{aligned} &\pi_P(P) = 1 && \{hidden, lost\} \\ &\pi_P(r_1, r_2) = 0.5 && \{visible\} \end{aligned}$$

whereas the possibility of all the other subprograms $P' \subseteq P$ is 0. Since each subprogram has a unique answer set, brave and cautious reasoning once again coincide. The results obtained by the different semantics are:

semantics from PASP$_\text{G}$	$\{hidden^{0.5}, lost^{1}\}$
semantics from PASP$_\text{Ł}$	$\{hidden^{0.5}, visible^{0.5}, lost^{0.5}\}$
semantics from PASP$_\text{r}$	$P \models_N^{c} \{hidden^{0.5}, visible^{0}, lost^{0.5}\}$
	$P \models_\Pi^{b} \{hidden^{1}, visible^{0.5}, lost^{1}\}$

Neither of these conclusions agree. Nevertheless, the semantics proposed in this chapter provide an intuitively satisfiable answer to the outcome of the game that the agent plays. Indeed, we can conclude that it is entirely possible that the agent has lost (since $P \models_\Pi^{b} lost^{1}$), while at the same time we know that this is not necessarily so (since $P \models_N^{c} lost^{0.5}$).

Still, some interesting links exist between the semantics for PASP that we thus far discussed in this thesis.

Proposition 32

Let P be a simple PASP program. For each literal 'l' we have that $N\left(P \models^{\mathrm{c}} l\right) \geq \lambda$ iff $M(l) \geq \lambda$ with M the possibilistic answer set of P under the semantics from $\mathsf{PASP}_{\text{Ł}}$, which in turn coincides with the semantics from $\mathsf{PASP}_{\mathsf{G}}$.

The previous result is not surprising because, without negation-as-failure, all semantics for PASP adhere to the semantics of possibilistic logic (interpreting rules in terms of material implication). Furthermore, notice that for simple programs, which have a unique answer set, checking whether $N\left(P \models^{\mathrm{b}} l\right) \geq \lambda$ is equivalent to checking whether $N\left(P \models^{\mathrm{c}} l\right) \geq \lambda$ and, similarly, checking whether $\Pi\left(P \models^{\mathrm{c}} l\right) \geq \lambda$ is equivalent to checking whether $\Pi\left(P \models^{\mathrm{b}} l\right) \geq \lambda$. Thus, also the complexity of these reasoning types coincide. This is not the case for possibilistic normal/disjunctive programs.

Proposition 33

Let P be a possibilistic normal program. Deciding whether

$$\begin{aligned}\Pi\left(P \models^{\mathrm{b}} l\right) \geq \lambda &\text{ is NP-complete;}\\ N\left(P \models^{\mathrm{c}} l\right) \geq \lambda &\text{ is coNP-complete;}\\ \Pi\left(P \models^{\mathrm{c}} l\right) \geq \lambda &\text{ is } \Sigma_2^{\mathsf{P}}\text{-complete;}\\ N\left(P \models^{\mathrm{b}} l\right) \geq \lambda &\text{ is } \Pi_2^{\mathsf{P}}\text{-complete.}\end{aligned}$$

A similar jump in the polynomial hierarchy can be seen for possibilistic disjunctive programs.

Proposition 34

Let P be a possibilistic disjunctive program. Deciding whether

$$\begin{aligned}\Pi\left(P \models^{\mathrm{b}} l\right) \geq \lambda &\text{ is } \Sigma_2^{\mathsf{P}}\text{-complete;}\\ N\left(P \models^{\mathrm{c}} l\right) \geq \lambda &\text{ is } \Pi_2^{\mathsf{P}}\text{-complete;}\\ \Pi\left(P \models^{\mathrm{c}} l\right) \geq \lambda &\text{ is } \Sigma_3^{\mathsf{P}}\text{-complete;}\\ N\left(P \models^{\mathrm{b}} l\right) \geq \lambda &\text{ is } \Pi_3^{\mathsf{P}}\text{-complete.}\end{aligned}$$

Thus far, we have considered a possibility distribution over the subprograms. However, from an application perspective, it often makes more sense to consider the possibility or necessity of an answer set. Clearly, each subprogram $P' \subseteq P$ may have zero or more answer sets. Furthermore we may have that two subprograms $P' \subseteq P$ and $P'' \subseteq P$ have the same answer set, even if $\pi_P(P') \neq \pi_P(P'')$. This leads us to the following definition.

Definition 30

Let P be a PASP program. Let π_P be the possibility distribution over the subsets $P' \subseteq P$. We define the possibility distribution π_A over the interpretations M:

$$\pi_A(M) = \max\left\{\pi_P(P') \mid M \text{ is an answer set of } {P'}^*\right\}$$

Note that this definition implies that $\pi_A(M) = 0$ whenever M is not an answer set of any subprogram $P' \subseteq P$. Let π_A be the possibility distribution over the interpretations M. The possibility that l is a literal in the epistemic state of the agent is given by $\Pi(l) = \max\{\pi_A(M) \mid l \in M\}$. The necessity that l is a literal in the epistemic state of the agent is given by $N(l) = 1 - \max\{\pi_A(M) \mid l \notin M\}$.

Interestingly, we have that $\Pi(l) = \Pi\left(P \models^{\mathrm{b}} l\right)$ and $N(l) = N\left(P \models^{\mathrm{c}} l\right)$. We now show that the new semantics are a proper extension of ASP.

Example 46

Consider the program from Example 43. We can verify that:

$$\begin{array}{lll} \Pi(b) = 1 & \Pi(c) = 1 & \Pi(d) = 1 \\ N(b) = 0 & N(c) = 0 & N(d) = 0.9 \end{array}$$

Since all the associated weights are 1, only the subprogram consisting of all the rules has a non-negative weight. Thus, only the classical answer sets are considered to be possible and the reasoning tasks from Definition 29 reduce to cautious and brave reasoning.

We now provide a more elaborate example, which highlights a complex setting in which humans can fairly easily come to a satisfactory conclusion, but which is not easy to encode using e.g. classical ASP.

Example 47

Triage at an accident site with a large number of casualties is an essential part of medical treatment when resources are limited. With the help of triage, it becomes possible to distinguish which casualties can wait for medical attention at a hospital and which casualties need to be treated on the spot. For brevity of this example, we consider a triage system with three levels. The casualty may have minor injuries ($minor$), which means that the person can wait for treatment at the hospital. The casualty may need to be treated immediately because of life-threatening, yet treatable injuries ($nowait$). The final category is beyond urgency ($beyond$) and encompasses those casualties which are so severely injured that, for the time being, medical attention is better directed towards casualties in the $nowait$ category as the chances of survival of casualties in this latter category are far higher.

A rescue helper is faced with a casualty with extensive external injuries ($extensive$), which indicates that he/she either falls in the $nowait$ or $beyond$ category. The casualty is faintly moaning ($moaning$), which, with a very low certainty, is an indication of the casualty still being conscious ($conscious$). Similarly, the casualty is exhibiting a bleeding nose ($nosebleed$), which might indicate internal bleeding ($internal$). The rescue helper would be a lot more certain that the casualty is experiencing internal bleeding when he/she also had low blood pressure ($lowblood$), but this has not been established. Whenever the casualty does not appear to be conscious, he/she is assumed to be in the $nowait$ or $beyond$ category. When there is no indication of internal bleeding, the casualty is in the $nowait$ category. A classification in one of the categories is never entirely certain since it is not possible, due to time constraints, to perform all the required tests. We have the program with the rules:

$$
\begin{aligned}
\mathbf{1} &: extensive \leftarrow \\
\mathbf{0.9} &: minor \leftarrow not\ extensive \\
\mathbf{1} &: moaning \leftarrow \\
\mathbf{0.1} &: conscious \leftarrow moaning \\
\mathbf{0.9} &: nowait \leftarrow not\ beyond, not\ internal, not\ conscious, extensive \\
\mathbf{0.9} &: beyond \leftarrow not\ nowait, not\ conscious, extensive \\
\mathbf{1} &: nosebleed \leftarrow \\
\mathbf{0.1} &: internal \leftarrow nosebleed \\
\mathbf{0.7} &: internal \leftarrow nosebleed, lowblood \\
\mathbf{1} &: \leftarrow nowait, beyond, extensive \\
\mathbf{1} &: \leftarrow not\ nowait, not\ beyond, extensive
\end{aligned}
$$

The last two rules encode that one and exactly one category needs to be assigned to the casualty (either $nowait$ or $beyond$), a requirement for an efficient triage.

Notice that at least one rule needs to be omitted to make the program consistent. Indeed, if we look at the classical program by ignoring the weights, then it is clear that we have information (with varying degrees of certainty) to support both $nowait$ and $beyond$. In PASP$_\mathsf{G}$, we are unable to take the low certainty of the literal $conscious$ into account when reasoning with negation-as-failure. As such, the literal $conscious$, for the purpose of determining the reduct, is considered as entirely true, which immediately removes the rules with $nowait$ or $beyond$ in the head of the rule from the reduct. In PASP$_\mathsf{Ł}$, on the other hand, we are unable to choose between $nowait$ and $beyond$. Indeed, we obtain an infinite number of answer sets M such that $\left\{nowait^c, beyond^{1-c}\right\} \subset M$ with $c \in [0.1, 0.9]$. Under the semantics proposed in this chapter, however, we obtain that $P \models^{\mathrm{b}}_{\Pi} \left\{beyond^{0.9}, nowait^{0.9}\right\}$, $P \models^{\mathrm{c}}_{\Pi} \left\{beyond^{0.9}, nowait^{0.1}\right\}$, $P \models^{\mathrm{b}}_{N} \left\{beyond^{0.9}, nowait^{0.1}\right\}$ and furthermore that $P \models^{\mathrm{c}}_{N} \left\{beyond^{0.1}, nowait^{0.1}\right\}$. The new semantics are thus capable of arriving at the desired conclusion. Indeed, since the necessity associated with $beyond$ is higher or equal to the necessity associated with $nowait$ for all reasoning tasks, a reasonable classification for the casualty is $beyond$. This corresponds with our intuition, as a number of indications hint towards this worst case scenario (e.g. the bleeding nose). If we added the fact that the casualty has low blood pressure, then even a brave conclusion with possibility measures would indicate that the casualty is beyond urgency, i.e. it would further reaffirm our conclusion.

5.3 Applications

The semantics for PASP proposed in this chapter, and the related concept of optional rules, have a number of interesting applications. In particular, in this section we prove how two interesting AI problems, namely cautious abductive reasoning and conformant planning, can be expressed in terms of programs with optional rules. This allows us to solve these problems with the implementations that will be presented in Chapter 6 and furthermore allows to trivially extend these problems with certainty weights. Notably, we will use the decision problems $N\left(P \models^{\mathrm{b}} l\right) \geq \lambda$ and $\Pi\left(P \models^{\mathrm{c}} l\right) \geq \lambda$, as these are the most expressive. We look at cautious abductive reasoning in Section 5.3.1 and show how cautious abductive reasoning can be used to simulate PASP$_\mathsf{r}$ and how, in turn, PASP$_\mathsf{r}$ can be used to simulate cautious abductive reasoning. In Section 5.3.2 we look at the problem of conformant planning and prove a simulation of this problem with PASP$_\mathsf{r}$.

5.3.1 Cautious abductive reasoning

Recall from Section 3.5 an abductive diagnosis program [Eiter et al. 1997] is encoded as a triple $\langle H, T, O\rangle$ where H is a set of atoms referred to as hypotheses, T is a (normal) ASP program referred to as the theory and O is a set of literals referred to as observations. Contrary to what we have done in Section 3.5, where we used CASP to solve an instance of a brave abductive reasoning problem, we are now interested in how we can solve cautious abductive reasoning problems. Cautious abductive reasoning is concerned with the problem of finding hypotheses that could explain the observations in O. Specifically, we are interested in a set $E \subseteq H$ such that $T \cup E \models^{c} O$, where E is said to be a cautious explanation.

We first show how the decision problems $N\left(P \models^{b} l\right) \geq \lambda$ and $\Pi\left(P \models^{c} l\right) \geq \lambda$ can be expressed in terms of cautious abductive reasoning.

Definition 31

Let P be a possibilistic disjunctive program. We define $P_{\text{elem}}(\lambda)$ as the set of rules:

$$\{r'_i \leftarrow \mid (r_i, c_i) \in P, c_i > 1 - \lambda\} \tag{5.1}$$
$$\cup \{\mathit{head}(r_i) \leftarrow \mathit{body}(r_i) \cup \{r'_i\} \mid (r_i, c_i) \in P\} \tag{5.2}$$

Intuitively, the program $P_{\text{elem}}(\lambda)$ simulates the semantics of PASP$_\text{r}$ using classical ASP. In particular, the rules in (5.1) ensure that all rules with a sufficiently high certainty are considered as valid. These rules, along with the rules that will be guessed by the hypotheses in our abductive cautious reasoning program, are applied using (5.2).

Proposition 35

Let P be a possibilistic disjunctive program. Let $\langle H, T, O\rangle$ be an abductive diagnosis program with $H = \{r'_i \mid (r_i, c_i) \in P, c_i \leq 1 - \lambda\}$, $T = P_{\text{elem}}(\lambda)$ and $O = \{l\}$. We have that $\Pi(P \models^{c} l) \geq \lambda$ iff the abductive diagnosis program $\langle H, T, O\rangle$ has a cautious explanation.

Using the relationship between assertions of the form $N(P \models^{b} l) \geq \lambda$ and assertions of the form $\Pi(P \models^{c} l) \geq \lambda$, we easily obtain the following proposition.

Proposition 36

Let P be a possibilistic disjunctive program. Let $\langle H, T, O\rangle$ be an abductive diagnosis program with $H = \{r'_i \mid (r_i, c_i) \in P, c_i \leq \lambda'\}$, $T = P_{\text{elem}}(1{-}\lambda') \cup \{\mathit{missing} \leftarrow \mathit{not}\ l\}$

and $O = \{missing\}$. Let $\lambda' \in cert^{+}(P)$ be such that $\lambda' < \lambda$ and for which we have that $\nexists \lambda'' \in cert^{+}(P) \cdot \lambda' < \lambda'' < \lambda$. We have that $N(P \models^{\mathrm{b}} l) \geq \lambda$ iff the abductive diagnosis program $\langle H, T, O \rangle$ has no cautious explanations.

Conversely, we can express a cautious abductive reasoning problem in terms of the decision problems $\Pi\,(P \models^{\mathrm{c}} l) \geq \lambda$ by utilizing the notion of optional rules.

Proposition 37

Let P_{abd} be the possibilistic normal program defined for an abductive diagnosis program $\langle H, T, O \rangle$ as

$$\{\mathbf{0.5}\colon block\_h \leftarrow \mid h \in H\} \tag{5.3}$$
$$\cup\,\{\mathbf{1}\colon h \leftarrow not\ block\_h \mid h \in H\} \tag{5.4}$$
$$\cup\,\{\mathbf{1}\colon goal \leftarrow O\} \tag{5.5}$$
$$\cup\,\{\mathbf{1}\colon r \mid r \in T\}\,. \tag{5.6}$$

It holds that $\langle H, T, O \rangle$ has a cautious explanation iff $\Pi\,(P_{\mathrm{abd}} \models^{\mathrm{c}} goal) \geq 0.5$. In particular, E is a cautious explanation iff for $P' = P_{\mathrm{abd}} \setminus \{block\_h \leftarrow \mid h \in E\}$ we have that $(P')^{*} \models^{\mathrm{c}} goal$.

Given that a cautious abductive reasoning problem can be expressed in terms of decision problems of PASP$_{\mathrm{r}}$, it becomes trivial to extend cautious abductive reasoning with weights. This further increases the expressive capabilities of the cautious abductive reasoning.

Example 48

John wants to become rich. He can either choose to invest his money in stocks, or to invest it in bonds (he lacks the money to do both). He can either win or fail with his investment, where he resp. becomes rich or bankrupt. Bonds are safer, but we are less certain that this will make John rich. Whether or not he should invest will also depend on how certain he is that his investment will succeed (very certain) or fail (somewhat certain) and his confidence that investing in bonds will make him rich (somewhat certain):

$$\begin{array}{rlrl}
\mathbf{0.8}\colon & win \leftarrow not\ fail, stocks & \mathbf{0.5}\colon & fail \leftarrow not\ win, stocks \\
\mathbf{1}\colon & rich \leftarrow win & \mathbf{1}\colon & bankrupt \leftarrow fail \\
\mathbf{0.5}\colon & rich \leftarrow bonds & \mathbf{1}\colon & \leftarrow stocks, bonds
\end{array}$$

Given the hypotheses $H = \{stocks, bonds\}$, it is clear that when we ignore the weights only $E = \{bonds\}$ is a cautious abductive explanation for the observation $O = \{rich\}$. If we take the certainties into account, then $E_1 = \{bonds\}$ is only a cautious explanation when we take $\lambda = 0.5$. In other words: we are only somewhat certain that the action $bonds$ will cautiously make us rich. Notice that $E_2 = \{stocks\}$ will only be a cautious explanation for $\lambda = 0.2$. Indeed, we are far less certain that buying stocks will be a guaranteed way to make us rich. Conversely, if we were looking for a brave explanation, then we would sooner be advised to buy stocks as these have a high potential of making us rich.

5.3.2 Conformant planning

Conformant planning is the problem of determining whether a plan (i.e. a series of actions) exists that always leads to the desired goal, regardless of the incompletely known initial state of the agent. Such problems are typically expressed using an action language.

An action language is built from a finite number of *fluent*s $f_1, ..., f_n$. A *state* is a finite set of fluents. The properties of the *initial state* s_0 are described by formulas of the form 'initially f', which are called *value propositions*, with f a *fluent literal*, i.e. a fluent or a fluent preceded by $\neg$. Changes of states are defined using a finite number of *action*s $a_1, ..., a_k$. Formulas of the form 'a causes f if $f_1, ..., f_m$' are called *effect propositions*, with $f, f_1, ..., f_m$ fluent literals and a an action. A *domain* D is a finite set of value and effect propositions. A *proper* domain, to which we limit ourselves in this thesis, is a domain in which we can determine in polynomial time what the successor state is, given the current state and an action. A *plan* is a sequence of actions $[a_1, ..., a_j]$. The *planning problem* is to determine for a given domain D and a fluent literal f whether a plan exists leading from s_0 to a state in which f is true, where we call f the *goal fluent*. To solve a planning problem, the domain is translated to ASP. Particularly, such a translation can be written as $P_{\text{act}} \cup P_{\text{rem}}$ where P_{act} are those rules used to describe the actions, whereas P_{rem} are the remaining rules that among others describe the (incomplete) initial state and rules to ensure inertia. Then, a plan exists when an answer set contains the goal fluent.

However, not all forms of planning problems can be solved in this way. When we say that we have an incomplete domain, this means that the initial values of some fluents are unknown. *Conformant planning* is the problem of determining whether for an incomplete domain and a fluent f a plan exists leading to a state in which f is true, regardless of the initial values of the unknown fluents. Only some action languages, e.g. $\mathcal{K}$ [Eiter et al. 2000], have the expressive power to describe conformant planning problems. For solving such problems, $DLV^{\mathcal{K}}$ relies on a two-step translation to ASP where a plan is generated (that is not necessarily a conformant plan) and verified to be an actual conformant plan, until an actual conformant plan is found. However, these methods are not designed to work with uncertainty and cannot, e.g. compute the most reliable plan when no conformant plan can be found.

Example 49

The $BT(p)$ problem is the basic version of the bomb in the toilet problem, which we discuss in more detail in Section 6.5. We know that there are p packages, exactly one of which is a bomb ($armed$). There is one toilet in which we can dump these packages. If the package with the bomb is not dumped in the toilet, it remains armed. Once the package with the bomb is dumped in the toilet, the bomb deactivates ($-armed$) and we reach a safe state. The goal is to reach this safe state without knowing which package is the bomb. We assume concurrency, i.e. only a single package can be dumped each time step. An encoding in e.g. $DLV^{\mathcal{K}}$ of this problem is:

```
fluents:
          armed(P) requires package(P).
          unsafe.

actions:
          dump(P) requires package(P).

always:
          inertial -armed(P).

          caused -armed(P) after dump(P).
          caused unsafe if not -armed(P).

          executable dump(P).
          nonexecutable dump(P) if dump(Q), P <> Q.

goal: not unsafe?
```

The only action that we can undertake is to $dump$ a package in the toilet. In the $always$ section we describe the inertia of the bomb, the effect of dumping the bomb, the unsafe state, and when the action $dump$ can or cannot be performed. The above encoding can then be translated to an equivalent ASP program [Eiter et al. 2000] (where we still need to provide the parameters on the number of packages to consider and the desired plan length). A solution to this planning problem for 4 packages and a plan length of 4 is e.g. the plan $dump(1), dump(2), dump(3), dump(4)$, i.e. we sequentially dump the packages. Any permutation of this plan is clearly also a plan. The encoding of this problem in ASP is listed in Appendix 1.

We now show how conformant planning can be expressed in terms of a decision problem of the form $N\left(P \models^{\mathrm{b}} l\right) \geq \lambda$. Note that the existence of a conformant plan can also be written as $\exists p\, \forall iv \cdot P(p, iv, pp)$ where $P(p, iv, pp)$ describes that for the planning problem pp and for all initially unknown values iv the plan p leads to the goal fluent.

Proposition 38

Let P_{con} be the possibilistic normal program defined for a conformant planning problem with the atom '$goal$' the desired goal fluent. We express the domain knowledge as a normal ASP program $P_{\text{act}} \cup P_{\text{rem}}$. Then P_{con} is:

$$\{\mathbf{0.5}\colon block\_i \leftarrow \mid r_i \in P_{\text{act}}\} \tag{5.7}$$

$$\cup \{\mathbf{1}\colon head(r_i) \leftarrow body(r_i) \cup \{not\ block\_i\} \mid r_i \in P_{\text{act}}\} \tag{5.8}$$

$$\cup \{\mathbf{1}\colon r \mid r \in P_{\text{rem}}\} \tag{5.9}$$

$$\cup \{\mathbf{1}\colon \leftarrow not\ goal\} \tag{5.10}$$

A conformant plan exists iff $\Pi\left(P_{\text{con}} \models^{\mathrm{c}} goal\right) \geq 0.5$.

An extract of this translation for Example 49 is given in Appendix 2.

Contrary to cautious abductive reasoning, many performant solvers for conformant planning exist, including $DLV^{\mathcal{K}}$. Nevertheless, such solvers cannot take certainties into account. For practical purposes though, when no "*perfect*" conformant plan exists, a plan that is conformant with a "*high certainty*" is preferred over choosing a plan arbitrarily.

Example 50

A firm has a client calling from city C to ask for an appointment the next day. The secretary knows that the sales person is either in city A (inA) or city B (inB). The secretary is almost certain (resp. absolutely certain) that a sales person can get from city A (resp. city B) to city C in one day (toC), assuming there is no road block. There are some rumours of a possible road block on the route from city A to C. Cities A and B also connect to city D, which is definitely reachable in one day from A and B (toD). This problem can be encoded as the following ASP program:

$$\begin{array}{rr}
\mathbf{1}\colon inA \leftarrow not\ inB & \mathbf{0.8}\colon toC \leftarrow inA, gotoC, not\ blockAC \\
\mathbf{1}\colon inB \leftarrow not\ inA & \mathbf{1}\colon toD \leftarrow inA, gotoD, not\ blockAD \\
\mathbf{0.2}\colon blockAC \leftarrow & \mathbf{1}\colon toC \leftarrow inB, gotoC, not\ blockBC \\
 & \mathbf{1}\colon toD \leftarrow inB, gotoD, not\ blockBD
\end{array}$$

The additional rules $P_{act} = \{\mathbf{1} : gotoC \leftarrow, \mathbf{1} : gotoD \leftarrow\}$ describe the actions to go to either city C or city D. If the goal would be to get a sales representative to city D, then a (perfect) cautious plan exists. Indeed, the plan consisting of the single action '$gotoD$' can, regardless of the initial state, take us to city D. However, this example specifically calls for sending a representative to city C. Since there is uncertainty as to whether or not the road is blocked between A and C, no cautious plan exists. Still, depending on how certain we want to be that we reach city C in time, a conformant plan may still exist. Given the low certainty of $(blockAC \leftarrow)$, it seems reasonable to assume that, with a fairly high possibility, a representative will be able to make it to city C from city A. Hence the plan consisting of the single action '$gotoC$' is, with a high certainty, a cautious plan.

5.4 Work Related to PASP$_Ł$ and PASP$_r$

A large collection of previous work in the literature is applicable to the work we presented in both this chapter and Chapter 4. We structure our discussion of these works along 3 main lines. Prior works that have combined logic programming with mechanisms to reason about uncertainty are explored in Section 5.4.1. In Section 5.4.2 we look at how ASP and possibility theory have been used in the literature for epistemic reasoning. Finally, in Section 5.4.3, we look at work on dealing with preferences and inconsistency handling, which are two of the domains for which uncertainty handling can be used.

5.4.1 Reasoning about Uncertainty in Logic Programming

The combination of logic programming and uncertainty handling in a single framework has been an active topic of research during the last decennia. The idea of combining logic programming and possibility theory was pioneered in [Dubois et al. 1991]. However, this approach was limited to classical formulas and as such did not include default negation, i.e. non-monotonic reasoning. In [Wagner 1998], a framework was proposed in which stable models are combined with a semi-possibilistic first-order logic, used as a logic of graded truth, to deal with uncertainty. Specifically, this semi-possibilistic logic is a compositional version of possibilistic logic, in which compositionality is preserved on the basis of a Heyting algebra. As a direct consequence, classical Boolean tautologies are no longer preserved in semi-possibilistic logic [Dubois and Prade 1994]. A more recent approach is the work from [Chesñevar et al. 2004], which combines defeasible logic, a form of non-monotonic reasoning involving both strict and defeasible rules, with possibility theory in a single framework. This allows, among other things, to resolve conflicts between contradictory goals. Possibility theory has also been combined with argumentation frameworks, e.g. in [Amgoud and Prade 2004], where revision rules allow an agent to revise its beliefs and goals.

Many probabilistic extensions of logic programming have also been considered. One of the first generalizations of propositional logic based on probability theory is [Nilsson 1986]. In this work, a logic is defined in terms of probability distributions over possible worlds, where the probability attached to a formula corresponds with the probability that the real world is among those that make the formula true. This idea was later extended to probabilistic logic programming [Lukasiewicz 2002], where maximum entropy takes on a role that is very similar to the role of minimal specificity, as shown in the transformation from [Dubois et al. 1993]. Similar work on combining probability theory with logic programming had already been done in [Ng and Subrahmanian 1991]. Indeed, this is one of the earliest works where, in the setting of probabilistic deductive databases, probability theory is combined with non-monotonic negation. While many works exist that combine probability theory and logic programming in general, only few have tried to combine probability theory and ASP. One of the most notable exceptions is the work from [Baral et al. 2009], where probabilistic atoms are used to encode the probability that an associated random variable will take on a given value.

Other frameworks to deal with uncertainty can also be used. Bayesian Logic Programming [Kersting and De Raedt 2001], where a generalization of Bayesian networks is used to reason over Horn clauses, is one such example. Bayesian networks employ well-understood Bayesian models for representing joint probabilities and offer a good graphical representation of local influences. Other popular approaches for dealing with uncertainty include Markov Logic Networks [Richardson and Domingos 2006], where first-order logic is used to compactly specify a Markov Network and as such allow for uncertain inference. Markov Logic Networks make it easy to specify interactions between random variables, and allow for dealing with cyclic dependencies. However, inference in Markov Logic Networks is often computationally quite complex. Furthermore, the weights attached to the formulas in Markov Logic Networks tend to be counterintuitive in that their influence in the network as a whole is not immediately obvious.

5.4.2 Epistemic Reasoning with ASP and Possibility Theory

In [Gelfond 1991] it was argued that classical ASP, while later proven to have strong epistemic foundations [Loyer and Straccia 2006], is not well-suited for epistemic reasoning. Specifically, ASP lacks mechanisms for introspection and can thus not be used to e.g. reason based on cautiously deducible information. At the same time, however, it was shown that extensions of ASP could be devised that do allow for a natural form of epistemic reasoning. The language ASP^K proposed in [Gelfond 1991] allows for modal atoms, e.g. Ka, where K is a modal operator that can intuitively be read as "*it is known that [a is true]*". These new modal atoms can in turn be used in the body of rules. The semantics of ASP^K were originally based on a three-valued interpretation (to allow for the additional truth value '$uncertain$'), but later, in [Truszczyński 2011], it was shown that this is not essential and that a more classical two-valued possible world structure can also be considered. In addition, further extensions are discussed that allow for epistemic reasoning over arbitrary theories, where it is shown that ASP^K can be encoded within these extensions. The complexity is studied for these extensions and is shown to be brought up one level w.r.t. ASP, e.g. to Σ_3^{P} for disjunctive epistemic programs.

Alternatively, existing extensions of ASP can be used to implement some epistemic reasoning tasks, such as reasoning based on brave/cautious conclusions. This idea is proposed in [Faber and Woltran 2009] to overcome the need for an intermediary step to compute the desired consequences of the ASP program P_1, before being fed into P_2. Rather, they propose a translation to manifold answer set programs, which exploit the concept of weak constraints [Buccafurri et al. 2000] to allow for such programs to access all desired consequences of P_1 within a single answer set. As such, for problems that can be cast into this particular form, only a single ASP program needs to be evaluated and the intermediary step is made obsolete.

As was mentioned in Section 1.1 and Section 4.5.1, the semantics of ASP can also be expressed in terms of autoepistemic logic [Marek and Truszczyński 1991]. These semantics have the benefit of making the modal operator explicit, allowing for an extensions of ASP that incorporates such explicit modalities to better express exactly which form of knowledge is required. However, since autoepistemic logic treats negation-as-failure as a modality, it is quite hard to extend to the uncertain case.

Finally, a formal connection also exists between the approach from Section 4.2 and the work on residuated logic programs [Damásio and Pereira 2001] under the Gödel semantics. Both approaches are different in spirit, however, in the same way that possibilistic logic (which deals with uncertainty or priority) is different from Gödel logic (which deals with graded truth). The formal connection is due to the fact that necessity measures are min-decomposable and disappears as soon as classical negation or disjunction is considered.

5.4.3 Uncertainty, Preferences and Inconsistency Handling

From a practical point of view, being able to deal with uncertainty plays an important (although often implicit) role in economics and in dealing with preferences, handling inconsistencies and dealing with weak constraints. Indeed, the uncertainty of costs [Mills 1959] is an important problem, where factors such as demand, production and actual costs are all pervaded by uncertainty [Sandmo 1971]. Unsurprisingly, this is a very active domain where probability theory plays a significant role [Garvey 2000]. Preferences, on the other hand, are an important topic within the ASP community. For example, ordered logic programs, introduced in [Van Nieuwenborgh and Vermeir 2002], are used to deal with preferences. Ordered logic programs assume a partial order among rules, allowing less important rules to be violated in order to satisfy rules with higher importance. In some sense, the use of such preferences among rules is related to using certainty weights, although the resulting semantics are closer in spirit to the approach from [Nicolas et al. 2006] than to the semantics we have developed throughout the last two chapters. Quite a number of other works also deal with preference handling in non-monotonic reasoning; a thorough overview is given in [Delgrande et al. 2004]. Weak constraints [Buccafurri et al. 1997] are yet another example of a problem that can be seen as a problem of preferences amongst rules. Indeed, weak constraints are constraints that we try to apply, while we are still willing to violate such constraints if applying the constraint would otherwise prevent us from finding an answer set. When dealing with inconsistencies in ASP, a number of different approaches can be taken. Indeed, approaches exist to highlight inconsistencies (e.g. [Gebser et al. 2008]), to resolve inconsistencies (e.g. [Balduccini and Gelfond 2003]) or to reason in inconsistent knowledge bases. Examples of the latter case include pstable models [Osorio et al. 2006], which are a framework characterized by a fusion of ASP and paraconsistent logic and which offer alternative semantics for PASP. The focus of these pstable models is mostly on handling inconsistency. For instance, the program containing the rule $(\mathbf{c}\colon a \leftarrow not\ a)$ has (a, c) as its unique possibilistic pstable model, which is not compatible with a reading of '$not\ a$' as "*it cannot be established that a is certain*". Specifically, such pstable models are closer to the intuition of classical models and possibilistic logic than they are to stable models, as in our approach. Also the semantics introduced in this chapter can be applied to handle inconsistencies. Indeed, we have seen that the semantics from this chapter are able to deal with inconsistencies in settings where PASP$_\mathsf{G}$ and PASP$_\mathsf{Ł}$ are not.

5.5 Summary

In this chapter we contrasted two different semantics for PASP based on the idea that an ASP program can be seen as a means to reason over the epistemic states of an agent, where each answer set is interpreted as an epistemic state. When extending this idea to PASP, the weights attached to the rules can be treated in two dual ways. On the one hand, weighted epistemic states can be considered where the weight attached to each rule reflects the certainty an agent would have in the conclusion of the rule, knowing that the body is satisfied. In other words, weighted rules are interpreted as rules with uncertain conclusions. This was the view proposed in Chapter 4.

Alternatively, we can maintain crisp epistemic states and use the weights associated with rules to express that some epistemic states are more compatible with available meta-knowledge than others. This has been the view developed in this chapter. Given this understanding of a PASP program, we treat the weight attached to each rule as the certainty that the rule is valid. As such, weighted rules are seen as rules whose validity is uncertain. We showed how treating a PASP program like this comes down to an efficient encoding of a possibility distribution over the exponentially many subprograms of an ASP program. This gives rise to four distinct types of inference, for which we have examined the computational complexity. We find that two of these inference types are as complex as the corresponding inference types in classical ASP, while the complexity of the other two inference types goes up one level in the polynomial hierarchy.

We showed the practical significance of our work in Section 5.3, beyond managing uncertainty in ASP, by showing how cautious abductive reasoning and conformant planning can be naturally seen as special cases of the considered problem. In addition, since PASP is a formalism for reasoning about uncertainty, these problems can trivially be extended with certainty degrees to increase their expressive power and e.g. find solutions with a high certainty when perfect solutions are not attainable.

6 Simulating CASP and PASP using classical ASP

6.1 Introduction

In the previous three chapters, we discussed the CASP and PASP extensions of ASP. However, we did not look at how these extensions can be implemented. In this chapter we show how these problems can be simulated using ASP. As such, we can rely on existing solvers for ASP, such as DLV [Leone et al. 2006] and clasp [Gebser et al. 2011], which have been demonstrated to be highly efficient [Denecker et al. 2009, Calimeri et al. 2011]. Furthermore, since we rely on third party implementations, we immediately benefit from any improvements made to these solvers.

The remainder of this chapter is organised as follows. In Section 6.2 we show how CASP can be simulated with ASP. However, this only applies to standard communicating answer sets. Indeed, no simulation with ASP is currently known to compute the multi-focused answer sets. However, such a simulation would only be feasible for normal component programs where we focus on at most a single program. Indeed, if we considered disjunctive component programs, or if we focus on more than a single program, then we would have a higher expressiveness than the most expressive form of classical ASP, i.e. a simulation using ASP would not be possible. As such, to simulate multi-focused answer sets in general, we cannot rely on a simulation using ASP. In Section 6.3 we provide a simulation of PASP$_Ł$ with ASP. Such a simulation is based on the syntactic counterpart of PASP$_Ł$ that was defined in Section 4.3.2. In Section 6.4 we show that also PASP$_\mathsf{r}$ can be simulated using ASP. While some reasoning tasks turn out to be straightforward to simulate, we show that the most complex reasoning tasks require a more involved simulation. Specifically, in these simulations we will rely on the simulation of ASP to a set of clauses to be able to reason over an ASP program within the simulation program. Finally, in Section 6.5 we end the chapter with a brief discussion.

6.2 Simulating CASP

We start by showing how ASP can be used to compute the communicating answer sets of a communicating disjunctive program. Specifically, there is a direct (linear) translation that transforms $\mathcal{P}$ into a disjunctive program P' such that the answer sets of P' correspond to the answer sets of $\mathcal{P}$. Recall from Chapter 3 that a communicating disjunctive program is a finite set of $\mathcal{P}$-component disjunctive programs, where each $\mathcal{P}$-component disjunctive program is a finite set of $\mathcal{P}$-component disjunctive rules. We have:

Proposition 39

Let $\mathcal{P}$ be a communicating disjunctive program. Let P' be the disjunctive program defined as follows. For every $Q\!:\!l \in (\mathcal{B}_\mathcal{P} \cup \neg\mathcal{B}_\mathcal{P})$ we add the following rules to P':

$$\begin{aligned} guess(Q\_l) &\leftarrow not\ not\_guess(Q\_l) & not\_guess(Q\_l) &\leftarrow not\ guess(Q\_l) \\ &\leftarrow guess(Q\_l), not\ Q\_l & &\leftarrow not\_guess(Q\_l), Q\_l. \end{aligned} \tag{6.1}$$

For every disjunctive communicating rule $\mathcal{P}$ of the form

$$r = Q\!:\!\gamma \leftarrow body$$

with $Q \in \mathcal{P}$, γ a set of literals and $body$ a set of extended situated literals, we add the rule $Q\_\gamma \leftarrow body'$ to P' with Q_γ the disjunctive set of situated literals $\{Q\_l \mid l \in \gamma\}$. We define $body'$ as follows:

$$\begin{aligned} body' = &\{Q\_b \mid Q\!:\!b \in body\} \\ &\cup \{guess(R\_c) \mid R\!:\!c \in body, Q \neq R\} \\ &\cup \{not\ S\_d \mid (not\ S\!:\!d) \in body\}\,. \end{aligned} \tag{6.2}$$

Note that we do not guess situated literals preceded by not, as the guess is already implicitly performed by the reduct of a classical ASP program. We have that $M = \{Q{:}l \mid Q\_l \in M'\}$ is an answer set of $\mathcal{P}$ if and only if M' is an answer set of P.

Due to the extra expressiveness and complexity of multi-focused answer sets, it is clear that in the general case no translation to classical (disjunctive) ASP is possible without an exponential blow-up unless the polynomial hierarchy collapses. Possible future implementations may, however, be based on a translation to other polynomial space (PSPACE) complete problems such as decision problems about QBF formulas or modal logics. A translation to QBF formulas seems to be the most natural. However, such translations are the subject of future research.

6.3 Simulating the PASP$_Ł$ Semantics

A simulation for PASP$_Ł$ can be devised based on the reduct operator that was defined in Section 4.3.2 for PASP programs with weak disjunction. However, this reduct operator was based on logical entailment, which is not readily available in ASP. When we restrict ourselves to strong disjunction we can simplify the reduct operator as follows.

Definition 32

Let P be a possibilistic disjunctive program P and let V be a valuation, i.e. a $V : Lit_P \rightarrow [0,1]$ mapping. The reduct P^V of P w.r.t. V is defined as:

$$\begin{aligned} P^V = \{\ &((l_1;...;l_k \leftarrow l_{k+1},...,l_m), \min(\lambda_{rule}, \lambda_{body})) \ \mid\ \min(\lambda_{rule}, \lambda_{body}) > 0 \\ &\wedge \lambda_{body} = \max\left\{\lambda \mid \{l_{m+1},...,l_n\} \cap V^{\underline{1-\lambda}} = \emptyset, \lambda \in [0,1]\right\} \\ &\wedge ((l_0;...;l_k \leftarrow l_{k+1},...,l_m, not\ l_{m+1},...,not\ l_n), \lambda_{rule}) \in P\} \end{aligned}$$

Note that P^V is a possibilistic positive disjunctive program. If we take P to be a possibilistic normal program, then P^V is is possibilistic simple program. The intuition of this reduct is similar as before. Indeed, on the one hand the certainty of the reduct of a rule is limited by the certainty of the rule itself. Furthermore, the certainty is limited by the certainty of the negative body of the rule. Based on this reduct, we can define a disjunctive program Q that simulates P with P a possibilistic normal program.

Definition 33

Let P be a possibilistic normal program. Let C be a set of certainty values. The normal program Q that simulates P contains for each $(r, \lambda) \in P$ with $r = (l_0 \leftarrow$

$l_1, ..., l_m, not\ l_{m+1}, ..., not\ l_n)$ and for each $\mu \in C$ such that $0 < \mu \leq \lambda$ the rule:

$$l_0\mu \leftarrow l_1\mu, ..., l_m\mu, not\ l_{m+1}\nu, ..., not\ l_n\nu$$

with $\nu = \min\{\xi \mid \xi > 1 - \mu, \xi \in C\}$ and with lc a fresh literal for each $l \in Lit_P$ and $c \in C$.

Example 51

Consider the possibilistic normal program P with the rules

$$\begin{aligned} \mathbf{0.6}\colon & travel\_plane \leftarrow concert\_booked, not\ canceled \\ \mathbf{1}\colon & concert\_booked \leftarrow \\ \mathbf{0.2}\colon & canceled \leftarrow . \end{aligned}$$

This program encodes that we booked a concert. We are quite certain that we will need to travel by plane to get to the concert site. We also read on a blog online that the concert is cancelled, although we only have a low certainty that this is indeed the case given the source of the information.

For compactness, in the remainder of this example we will write tp, cb and can instead of $travel\_plane$, $concert\_booked$ and $canceled$, respectively. We take $C = cert^+(P) = \{0, 0.2, 0.4, 0.5, 0.6, 0.8, 1\}$. The disjunctive program Q defined by Definition 33 is then the program consisting of the rules

$$\begin{array}{rrrr} cb(1) \leftarrow & cb(0.8) \leftarrow & cb(0.6) \leftarrow & cb(0.5) \leftarrow \\ cb(0.4) \leftarrow & cb(0.2) \leftarrow & can(0.2) \leftarrow & \end{array}$$

$$\begin{aligned} tp(0.6) &\leftarrow cb(0.6), not\ can(0.5) \\ tp(0.5) &\leftarrow cb(0.5), not\ can(0.6) \\ tp(0.4) &\leftarrow cb(0.4), not\ can(0.8) \\ tp(0.2) &\leftarrow cb(0.2), not\ can(1) \end{aligned}$$

Q has the answer set

$$\begin{aligned} M = &\{cb(1), cb(0.8), cb(0.6), cb(0.5), cb(0.4), cb(0.2)\} \\ &\cup \{tp(0.6), tp(0.5), tp(0.4), tp(0.2)\} \cup \{can(0.2)\} \end{aligned}$$

which indeed corresponds to the possibilistic answer set $\left\{cb^1, tp^{0.6}, can^{0.2}\right\}$ of P.

Let us now consider that we are entirely certain that we need to travel by plane when we booked the concert and the concert is not cancelled. We thus have the rule

$$\mathbf{1}\colon travel\_plane \leftarrow concert\_booked, not\ canceled.$$

We can easily modify Q in accordance by adding the rules

$$\begin{aligned} tp(1) &\leftarrow cb(1), not\ can(0.2) \\ tp(0.8) &\leftarrow cb(0.8), not\ can(0.4). \end{aligned}$$

Q now has the answer set

$$\begin{aligned} M = &\{cb(1), cb(0.8), cb(0.6), cb(0.5), cb(0.4), cb(0.2)\} \\ &\cup \{tp(0.8), tp(0.6), tp(0.5), tp(0.4), tp(0.2)\} \cup \{can(0.2)\} \end{aligned}$$

which again corresponds to the possibilistic answer set $\left\{cb^1, tp^{0.8}, can^{0.2}\right\}$ of P.

The next two propositions confirm that this simulation is indeed correct and that we can use a classical normal program to obtain the possibilistic answer sets of our possibilistic normal program.

Proposition 40

Let P be a possibilistic normal program and Q the simulation of P defined as in Definition 33 with $C = cert^+(P)$. Let V be the valuation defined by $V(l) = \max\{\lambda \mid l\lambda \in M\}$. If M is a classical answer set of Q, then V is a possibilistic answer set of P.

Proposition 41

Let P be a possibilistic normal program and Q the simulation of P defined as in Definition 33, and $M = \{l\lambda \mid \lambda \leq V(l), \lambda \in C\}$. If V is a possibilistic answer set of P such that $V(l) \in C$ for all $l \in \mathcal{B}_P$, then M is a classical answer set of Q.

Note that only those possibilistic answer sets for which $V(l) \in C$ are found using our simulation. In particular, only a finite number of answer sets can thus be found. As discussed in Section 4.2.2, this has few practical implications since we only need to consider the finite set of certainty values $cert^+(P)$ for brave and cautious reasoning. The simulation as defined above can also be used to simulate possibilistic disjunctive programs. However, such a simulation would generate a large number of extra answer sets that do not directly correspond with a valuation of the corresponding possibilistic disjunctive program. In the example above, if we had $(travel\_plane; travel\_car)$, instead of $travel\_plane$, we would obtain answer sets that contain e.g. $tp(1)$, but we would also have answer sets that contain e.g. $tp(1)$ and $tc(0.8)$ with tc a shorthand for $travel\_car$. While such a simulation is still useful to simulate brave and cautious reasoning, it cannot be used to compute the possibilistic answer sets.

6.4 Simulating the PASP$_\mathrm{r}$ Semantics

Now we show how the semantics presented in Chapter 5 can be simulated using existing formalisms. In particular, the decision problems $\Pi(P \models^{\mathrm{b}} l) \geq \lambda$ or $N(P \models^{\mathrm{c}} l) \geq \lambda$ will be simulated using brave and cautious reasoning over classical programs. The remaining decision problems, which have a higher expressiveness, will be simulated by relying on a translation of ASP to a set of clauses. We start by describing the simulation of the decision problems $\Pi(P \models^{\mathrm{b}} l) \geq \lambda$ or $N(P \models^{\mathrm{c}} l) \geq \lambda$, which share a common base program P_{basic}.

Definition 34

Let P be a possibilistic disjunctive program. We define $P_{\mathrm{basic}}(\lambda)$ as the set of rules:

$$\{r'_i \leftarrow not\ nr'_i \mid (r_i, c_i) \in P, c_i \leq 1 - \lambda\}$$
$$\{nr'_i \leftarrow not\ r'_i \mid (r_i, c_i) \in P, c_i \leq 1 - \lambda\} \tag{6.3}$$
$$\cup \{r'_i \leftarrow \mid (r_i, c_i) \in P, c_i > 1 - \lambda\} \tag{6.4}$$
$$\cup \{head(r_i) \leftarrow body(r_i) \cup \{r'_i\} \mid (r_i, c_i) \in P\} \tag{6.5}$$

Intuitively, the program $P_{\mathrm{basic}}(\lambda)$ simulates the semantics from Section 5.2 using classical ASP. In particular, the rules from (6.3) generate as many answer sets as there are choices of rules such that the possibility of the associated subprograms remains sufficiently high, i.e. greater than or equal to λ. The rules in (6.4) ensure that all rules with a sufficiently high certainty are considered as valid. Depending on the choice made in (6.3), the information encoded in the respective rules is applied using (6.5).

Example 52

Consider the possibilistic normal program P with the rules

$$\mathbf{0.8}\colon b \leftarrow not\ c \qquad \mathbf{0.3}\colon c \leftarrow d, not\ b \qquad \mathbf{0.9}\colon d \leftarrow .$$

We have the classical normal program $P_{\text{basic}}(0.7)$ with the rules

$$\begin{array}{ccc} r_1' \leftarrow & r_2' \leftarrow not\ nr_2' & r_3' \leftarrow \\ & nr_2' \leftarrow not\ r_2' & \\ b \leftarrow not\ c, r_1' & c \leftarrow d, not\ b, r_2' & d \leftarrow r_3' \end{array}$$

The program $P_{\text{basic}}(\lambda)$ can now be extended to solve the main reasoning tasks on the second level of the polynomial hierarchy.

Definition 35

Let P be a possibilistic disjunctive program. The classical disjunctive program $P^{\Pi}_{brave}(l, \lambda)$ to verify $\Pi(P \models^{\text{b}} l) \geq \lambda$ is defined as $P_{\text{basic}}(\lambda) \cup \{\leftarrow not\ l\}$.

Intuitively, the simulation $P^{\Pi}_{brave}(l, \lambda)$ will look for a world in which 'l' is true, and has an associated possibility of λ. If such a world exists, i.e. if we have an answer set, then $\Pi(P \models^{\text{b}} l) \geq \lambda$ is true.

Proposition 42

Let P be a possibilistic disjunctive program and $P^{\Pi}_{brave}(l, \lambda)$ the classical disjunctive program as defined in Definition 35. We have that $\Pi(P \models^{\text{b}} l) \geq \lambda$ iff $P^{\Pi}_{brave}(l, \lambda)$ has a classical consistent answer set.

Definition 36

Let P be a possibilistic disjunctive program. The classical disjunctive program $P^{N}_{cautious}(l, \lambda)$ to verify $N(P \models^{\text{c}} l) \geq \lambda$ with $\lambda > 0$ is defined as $P_{\text{basic}}(1-\lambda') \cup \{\leftarrow l\}$ with $\lambda' \in cert^{+}(P)$ such that $\lambda' < \lambda$ and for which we have that $\nexists \lambda'' \in cert^{+}(P) \cdot \lambda' < \lambda'' < \lambda$.

Note that when $\lambda = 0$, it trivially holds that $N(P \models^{\text{c}} l) \geq \lambda$ is true.

Intuitively, to determine whether $N(P \models^{\text{c}} l) \geq \lambda$ we need to verify that we have that $\max\left\{\pi_P(P') \mid P' \subseteq P \text{ and } P'^{*} \not\models^{\text{c}} l\right\} \leq 1 - \lambda$. In other words, whether we do not have any subprogram P' such that $P'^{*} \not\models^{\text{c}} l$ and $\pi_P(P') > 1-\lambda$. The simulation $P^{N}_{cautious}(l, \lambda)$ intuitively looks for a world with a certainty higher than $1 - \lambda$ in which 'l' is false, i.e. l is not a cautious consequence. If such a world does not exist, i.e. if we find no answer sets, then $N(P \models^{\text{c}} l) \geq \lambda$.

Proposition 43

Let P be a possibilistic disjunctive program, $\lambda > 0$ and $P^N_{cautious}(l, \lambda)$ the classical disjunctive program as defined in Definition 36. We have that $N(P \models^{\mathrm{c}} l) \geq \lambda$ iff $P^N_{cautious}(l, \lambda)$ has no classical consistent answer set.

For the decision problems in Proposition 42 and 43, it was sufficient to find *one answer set* of a particular subprogram that satisfies some condition, which is why we were able to simulate these problems relatively straightforwardly. To decide whether $\Pi\left(P \models^{\mathrm{c}} l\right) \geq \lambda$ or $N\left(P \models^{\mathrm{b}} l\right) \geq \lambda$, on the other hand, we need to verify a particular condition *for each answer set* of a particular subprogram. Since the complexity of these decision problems is higher, we are only able to provide simulations for P a possibilistic normal program as these are already Σ_2^{P}-hard and Π_2^{P}-hard, respectively.

Our simulation is based on the idea that we can use a disjunctive ASP program to reason about the answer sets of a normal ASP program. This will be accomplished by translating the normal ASP program to a set of clauses to ensure that the program is free of negation-as-failure. This is needed to be able to apply saturation techniques over a normal program. If a program is tight (i.e. it has no positive loops), then a translation to clauses can be obtained by determining the completion of the original program [Fages 1994]. Not every ASP program, however, is tight. As such, we will need to rely on more complex translations of ASP programs to sets of clauses, such as the translation based on a characterization in terms of level numbering presented in [Janhunen 2004]. Once we have the translation to a set of clauses, we generate answer sets for every subprogram and we apply saturation techniques to both validate whether a given interpretation is a valid model of the subprogram and to verify whether a given literal is a desired conclusion of the given subprogram.

Definition 37

Let $P = \{p_1, ..., p_n\}$ be a possibilistic normal program and every $p_i = (r_i, c_i)$ for $1 \leq i \leq n$ a possibilistic normal rule. The disjunctive program $P_{complex}(\lambda)$ is defined as the set of rules:

$$\{\mathsf{r}_i \leftarrow not\ \neg\mathsf{r}_i \mid 1 \leq i \leq n, c_i \leq 1 - \lambda\}$$

$$\cup \{\neg\mathsf{r}_i \leftarrow not\ \mathsf{r}_i \mid 1 \leq i \leq n, c_i \leq 1 - \lambda\} \tag{6.6}$$

$$\cup \{\mathsf{r}_i \leftarrow \mid 1 \leq i \leq n, c_i > 1 - \lambda\} \tag{6.7}$$

$$\cup \left\{cl \leftarrow \mid cl \in cls(P^{\mathsf{r}})^{\dagger}\right\} \tag{6.8}$$

$$\cup \left\{(sat \leftarrow a, na) \mid a \in at(cls(P^{\mathsf{r}})^{\dagger})\right\} \tag{6.9}$$

$$
\begin{aligned}
&\cup \left\{(a \leftarrow sat) \mid a \in at(cls(P^{\mathsf{r}})^{\dagger})\right\} \\
&\cup \left\{(na \leftarrow sat) \mid a \in at(cls(P^{\mathsf{r}})^{\dagger})\right\} && (6.10)\\
&\cup \{\leftarrow not\ sat\} && (6.11)\\
&\cup \left\{cl'_r \leftarrow \mid cl \in cls(P^{\mathsf{r}})^{\dagger}\right\} && (6.12)\\
&\cup \left\{\leftarrow a', na' \mid a \in at(cls(P^{\mathsf{r}})^{\dagger})\right\} && (6.13)
\end{aligned}
$$

where $cls(P)$ is a representation of the normal program P as a set of clauses (e.g. [Janhunen 2004]), P^{r} is the set of rules $\{head(r_i) \leftarrow body(r_i), \mathsf{r}_i \mid (r_i, c_i) \in P\}$ with 'r_i' a fresh atom, $C^{\dagger}$ is the set of clauses obtained from C by replacing every occurrence of a negated atom $\neg a$ with a fresh atom na except for the atoms 'r_i' and $at(C)$ is the set of atoms appearing in C from which we remove the atoms 'r_i'. Finally, cl'_r is obtained from a clause cl by replacing every literal from Lit_P with l'.

Proposition 44

Let P be a possibilistic normal program and $P^{\mathrm{c}}_{\Pi}(l, \lambda)$ the disjunctive program defined as $P_{complex}(\lambda) \cup \{sat \leftarrow l\}$. Then $\Pi\left(P \models^{\mathrm{c}} l\right) \geq \lambda$ iff $P^{\mathrm{c}}_{\Pi}(l, \lambda)$ has a classical answer set.

Proposition 45

Let P be a possibilistic normal program and $P^{\mathrm{b}}_{\mathrm{N}}(l, \lambda)$ the disjunctive program defined as $P_{complex}(1 - \lambda') \cup \{sat \leftarrow not\ l\}$ with λ' defined as in Proposition 43. Then $N\left(P \models^{\mathrm{b}} l\right) \geq \lambda$ iff $P^{\mathrm{b}}_{\mathrm{N}}(l, \lambda)$ has no classical answer set.

6.5 Implementation in PASP2ASP

All of the simulations discussed in this section can easily be implemented. Furthermore, in most cases, the simulations are linear and allow us to quickly compute the corresponding communicating or possibilistic answer set. The simulation presented in Definition 37, however, is more elaborate, but can nevertheless easily be implemented using mostly off-the-shelf tools. To verify its effectiveness in practice, we have implemented a prototype which is available from http://www.cwi.ugent.be/kim/pasp2asp/. The translator PASP2SAT prepares an input PASP program by removing the certainties and adding the fresh atom r_i to the body of each rule $r_i \in P$. The resulting ASP program is converted to an equivalent set of clauses using the technique from [Janhunen 2004]. The output in DIMACS CNF form is converted back into ASP rules by CLAUSE2ASP. The rules (6.8) – (6.13) are constructed from the information of the translation to clauses, while λ is needed to add the rules (6.6) – (6.7) and l is required to add the rules that are specific to the simulation of the decision problem as identified in Proposition 44 and 45. An overview of the implementation is given in Figure 6.1. The answer sets of the resulting ASP program can then be computed using an ASP solver for disjunctive programs, e.g. DLV [Eiter et al. 1999, Leone et al. 2006].

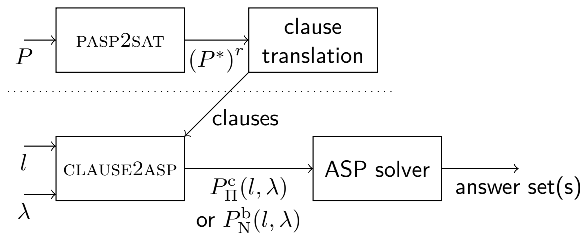


Figure 6.1: overview of the implementation.

While no benchmark instances exist for PASP, we know from Section 5.3 that conformant planning can be modelled as a problem in $\mathrm{PASP_r}$. We can thus compare our solver against $DLV^{\mathcal{K}}$, which is a solver specifically made for conformant planning and which is built on top of DLV, an ASP solver. For both solvers, we consider a world-state encoding in action language $\mathcal{K}$ of the $BT(p)$ and $BMTUC(p,t)$ problem [Eiter et al. 2003]. As we have seen in Section 5.3.2, $BT(p)$ is the basic version of the bomb in the toilet problem. We know that there are p packages, exactly one of which is a bomb. There is one toilet in which we can dump these packages. Once the package with the bomb is dumped in the toilet, the bomb deactivates and we reach a safe state. The goal is to reach this safe state, without knowing which one of the packages is the actual bomb. The $BMTUC(p,t)$ problem is a much harder variant of the $BT(p)$ problem. In this variant, we have p packages and t toilets. Clogging may occur when we dump a package in the toilet, preventing us from dumping additional packages in this toilet. We can flush a toilet when it is clogged, which always unclogs the toilet. As before, a safe state is reached when the bomb is deactivated by dumping it into an unclogged toilet. The goal is once again to arrive at a safe state irrespective of which package contains the actual bomb. We now briefly discuss how PASP2ASP compares to $DLV^{\mathcal{K}}$ on a number of problem instances. The results are summarized in Table 6.1[1].

$BT(p)$	$DLV^{\mathcal{K}}$	PASP2ASP	$BMTUC(p,4)$	$DLV^{\mathcal{K}}$	PASP2ASP
$BT(2)$	0.015 s	0.050 s	$BMTUC(2,4)$	0.015 s	0.075 s
$BT(3)$	0.015 s	0.052 s	$BMTUC(3,4)$	0.015 s	0.072 s
$BT(4)$	0.016 s	0.055 s	$BMTUC(4,4)$	0.017 s	0.128 s
$BT(5)$	0.014 s	0.057 s	$BMTUC(5,4)$	0.144 s	38.876 s
$BT(6)$	0.015 s	0.056 s	$BMTUC(6,4)$	– s	– s

Table 6.1: comparison of PASP2ASP and $DLV^{\mathcal{K}}$ on $BT(p)$ and $BMTUC(p,t)$ instances

On the $BT(p)$ instances our implementation is slower than $DLV^{\mathcal{K}}$. This can be explained by the increased complexity of our simulation, where the translation to clauses generates a considerable number of additional rules. However, similar to $DLV^{\mathcal{K}}$, our results scale well when considering larger instances of $BT(p)$. When looking at the $BMTUC(p,4)$ instances, both our own implementation and $DLV^{\mathcal{K}}$ are able to solve $BMTUC(2,4)$, $BMTUC(3,4)$ and $BMTUC(4,4)$ in less than a second. However, when there are more packages than toilets, our implementation demonstrates a considerable hit in performance. Neither $DLV^{\mathcal{K}}$ nor our own implementation are able to find a conformant plan before the cut-off time of 60 seconds for the $BMTUC(6,4)$ instance. Similar behaviour can be observed for other $BMTUC(p,t)$ instances where we take $t \neq 4$.

Although our solver is somewhat slower than $DLV^{\mathcal{K}}$, it is competitive on most problem instances. Moreover, while $DLV^{\mathcal{K}}$ is optimized for the problem of conformant planning, our solver is more generic and can thus not exploit problem-specific heuristics. In these experiments we furthermore did not consider optimizations based on the tightness of the programs. Indeed, if we check for tightness we can translate an ASP program P more efficiently to a set of clauses by determining the complement $comp(P)$, rather than relying on a more complex translation such as the one we use from [Janhunen 2004].

6.6 Summary

In this chapter we have shown how the extensions of ASP that were presented in Chapter 3, 4 and 5 can be simulated using classical ASP. Many of these simulations are fairly straightforward and rely on a linear translation, which makes them easy to implement.

[1]All results averaged over 3 runs and obtained on a 2.4Ghz Intel Core 2 Duo system with 8GB RAM (using OS X 10.8 64 bit). Run-times longer than 60 s are omitted.

Furthermore, since these simulations rely on classical ASP, they allow us to quickly compute communicating or possibilistic answer sets using current state-of-the-artASP solvers.

We have seen that a simulation does not (yet) exist for the computation of multi-focused answer sets. Still, in theory classical disjunctive ASP has the expressive power to simulate communicating normal programs where we focus on a single component program. When we want to focus on more than one component program, classical ASP lacks sufficient expressivity and we would need a translation to e.g. QBF formulas.

The most complex decision problems of PASP_r, namely deciding whether $\Pi\left(P \models^{\text{c}} l\right) \geq \lambda$ or $N\left(P \models^{\text{b}} l\right) \geq \lambda$, required more involved simulations. Indeed, the simulation needs to reason over the answer sets of the classic program obtained from P. To do this, we had to rely on a translation of ASP to clauses. Our simulation uses such a translation as a black box, and can thus easily be adapted to a more performant version, as advances in this area are made. Furthermore, we implemented this simulation and verified its feasibility. The implementation does not yet take certain optimisations into account, such as verifying whether or not a program is tight. Since most ASP programs are indeed tight, and since tight programs can more easily be translated to a set of clauses with equivalent models, such optimisations may in the future allow us to considerably speed up our implementation. Even though our implementation is still preliminary, our comparison with $DLV^{\mathcal{K}}$ showed that it is already capable of solving real-life problems in reasonable time. For most instances, our solver needed an amount of time which was of the same order of magnitude as $DLV^{\mathcal{K}}$, even though $DLV^{\mathcal{K}}$, being specifically optimized for conformant planning, was faster.

Conclusions

Answer Set Programming (ASP) is a domain-specific programming language based on the stable model semantics for logic programming. An ASP program consists of a set of rules, where each rule is of the form $(conclusion \leftarrow conditions)$. A rule encodes the information that its $conclusion$ must be true whenever all of its $conditions$ are satisfied. To solve a problem using ASP, we encode it in such a way that specific models, called the *answer sets*, correspond with the solutions of the original problem. An important component of ASP is the use of a non-monotonic operator 'not', called *negation-as-failure*, in the condition part of a rule. Intuitively, an expression of the form '$not\ sunny$' is true when there is no proof for '$sunny$'. In this example, $sunny$ is an atom, i.e. a logical formula that contains no connectives and that is either true or false. The use of a non-monotonic operator, where we obtain conclusions based on the absence of information, is different from the use of classical negation. Indeed, concluding that '$\neg sunny$' is true would require us to be able to effectively prove that it is not sunny. Classical negation is also considered in ASP in the form of literals, where a literal is either an atom or the classical negation of an atom.

The use of a non-monotonic operator in ASP influences its expressivity. ASP programs are divided in specific categories based on syntactic restrictions on the rules in an ASP program. The least expressive class of programs that we consider are those in which 'not' does not occur and where each rule has a deterministic conclusion. Such programs are referred to as *simple programs*. Rules in a simple program are of the form $(l_0 \leftarrow l_1, ..., l_m)$. *Normal programs* are ASP programs that do allow for the operator 'not' in the condition part, but where each rule still only has a deterministic conclusion. These rules are of the form $(l_0 \leftarrow l_1, ..., l_m, not\ l_{m+1}, ..., not\ l_n)$. Programs in which we allow 'not' in the conditions and where each rule can have a non-deterministic conclusion are called *disjunctive programs*. Such rules are of the form $(l_0; ...; l_k \leftarrow l_{k+1}, ..., l_m, not\ l_{m+1}, ..., not\ l_n)$. The expressivity of simple programs, normal programs and disjunctive programs is P, NP and Σ_2^{P}, respectively.

The use of a non-monotonic operator in ASP allows us to model and revise knowledge. The revision of knowledge is an important part of common-sense reasoning. For example, we may usually take the train to go to work, unless we find out that there will be a strike. However, we cannot intuitively use ASP to model that different pieces of knowledge have been obtained from different sources, since ASP is defined as one program consisting of a set of rules. Nevertheless, examples are plentiful where different sources provide different pieces of information, such as a police investigation where each witness only knows some of the facts. In this thesis we were particularly interested in how information can be exchanged between different contexts and how this affects the complexity. Furthermore, ASP cannot be used in an intuitive way to reason about uncertainty. For example, in common-sense reasoning we should be able to derive conclusions based on knowledge such as "*the train will possibly be cancelled*", while the uncertainty inherent in this phrase cannot easily be modelled in ASP. In this thesis we looked at how ASP can be extended to reason about qualitative uncertainty (e.g. as in the statement "*I will open my umbrella when I am fairly certain that it is raining outside*"), which is distinct from quantitative uncertainty (e.g. "*the letter will be delivered tomorrow with a 99% certainty*").

To model multi-context systems and uncertainty in ASP, we considered epistemic extensions of ASP in this thesis. Specifically, we have introduced Communicating Answer Set Programming (CASP), an extension of ASP that allows for a network of ASP programs to communicate, and we investigated new semantics for Possibilistic Answer Set Programming (PASP), which adds a layer of uncertainty to ASP. Furthermore, we thoroughly studied the complexity of these extensions to get a good understanding of their expressivity. To clarify all the concepts that are used in this thesis, we recalled some preliminary notions on complexity, ASP, possibility theory and PASP in Chapter 2.

CASP was introduced in Chapter 3, where we considered a network of ASP programs that are able to communicate and where we were particularly interested in the effect that this communication has on the overall complexity. We showed that the introduction of a new literal '$Q\!:\!l$' allows us to exchange information between programs. The intuition of this literal is that of asking a question ("do you believe the literal 'l' to be true?") to some other program (namely the program 'Q'). We found that a network of communicating simple programs (which, individually, have an expressivity of P) are able to simulate a normal program (which has an expressivity of NP). However, a network of normal programs are not expressive enough to simulate a disjunctive program (which has an expressivity of Σ_2^{P}).

An elaboration on this idea of communication is the concept of '*focussing*'. Normally, the solutions of an ASP program without negation-as-failure are defined as its minimal models. In other words: a solution is the minimally necessary information that can be derived from the program. However, when we consider a network of communicating programs, the notion of minimality becomes ambiguous. Indeed, minimality can be defined on a global level, i.e. where we do not differentiate between programs, or minimality can be defined on the level of a single program. The idea of focussing introduced in Section 3.4 is based on this last notion of minimality. In addition, focussing can be applied repeatedly by focussing on a series of different programs in the network. In that case, the order in which we focus becomes important. When we studied the complexity, we saw that every time we focussed on another program in the network a jump is made in the polynomial hierarchy (i.e. from P to NP to Σ_2^{P} *etc.*). In this way it is possible to solve all problems in polynomial space (PSPACE), as long as there are sufficiently many unique programs on which to focus.

Aside from having a network of programs that are able to communicate, we were also interested in reasoning about uncertain information. PASP is an existing extension of ASP [Nicolas et al. 2006] that allows for the representation of, and reasoning about, uncertain information. To this end, the syntax of ASP is extended by associating a weight $\lambda \in]0,1]$ with a rule. As such, a PASP program is a set of rules of the form $(\boldsymbol{\lambda}: \mathit{conclusion} \leftarrow \mathit{conditions})$. This weight can be interpreted as the maximum certainty with which the conclusion of the rule can be derived when all the conditions of the rule are satisfied. Current semantics for PASP, however, can have unintuitive results in certain settings. The underlying cause is the particular interpretation of the non-monotonic operator 'not'. Specifically, the existing semantics interpret a statement such as '$\mathit{not\ sunny}$' as "*we have no strictly positive certainty that 'sunny' is true*". As a result, it is not possible to differentiate between situations in which sunny is somewhat necessary and situations in which sunny is completely necessary. Alternatively, however, '$\mathit{not\ sunny}$' can be interpreted as "*the degree to which '$\neg\mathit{sunny}$' is possible*", or, equivalently, "*the degree to which it is not the case that we can derive 'sunny' with certainty*". In Chapter 4 we looked at how semantics for PASP can be defined as constraints on possibility distributions that adhere to this new interpretation. These new semantics offer intuitively acceptable solutions in many situations where previous semantics for PASP do not.

These new semantics for PASP can be used as a characterization of classical ASP (i.e. without certainty degrees). Interestingly, when extending this characterization to disjunctive programs, two different views of disjunction are unearthed. One of these views corresponds with classical disjunctive programs, whereas the other view leads to new semantics. These new semantics interpret disjunctions mainly in an atomic way. The complexity of programs with this new form of disjunction is in between of the complexity of normal programs and classical disjunctive programs when we allow for classical negation. However, the complexity coincides with the complexity of normal programs when we do not consider classical negation.

Both the new semantics as well as the existing semantics for PASP interpret the weight attached to a rule as the maximum certainty with which the conclusion can be derived when the conditions of the rule are satisfied. As such, these semantics allow us to model that we believe particular information only with a limited certainty. Alternatively, the weight attached to a rule can be interpreted in a different way, namely as the certainty with which we believe the information encoded in the rule itself to be valid. As such we express uncertainty on the level of the model itself, rather than on the level of the information (which is either true or false). Given this understanding we view a PASP program as a set of uncertain rules, rather than as a set of rules with uncertain conclusions. To define the semantics of uncertain rules we can look at the semantics of possibilistic logic. Conclusions that can be derived with a high certainty in possibilistic logic are those conclusions that we are still able to derive after we have removed the least certain information. However, due to the use of a non-monotonic operator in ASP, the omission of rules may allow us to derive new information. This newly derived information may itself be incorrect. Hence, merely omitting the least certain rules is not enough and instead we — conceptually — need to consider all the subsets of uncertain rules. Each such subset corresponds with the assumption that those particular rules are exactly those rules that are valid. We can then associate a possibility with each of these subsets, where a subset of rules is progressively less possible as rules with a progressively higher certainty are omitted from it.

We introduced semantics based on this view in Chapter 5. Furthermore, we compared the new semantics proposed in Chapter 5 with the already discussed semantics from Chapter 4, along with a study of the complexity of the most important decision problems of these new semantics. The semantics of uncertain rules furthermore revealed the concept of optional rules. We showed that a number of important problems in Artificial Intelligence can be expressed in terms of such optional rules and can therefore be modelled using a PASP program. One such problem is conformant planning, where we are uncertain about the initial state of the problem, but we nevertheless want to find a reliable plan that leads us to a specific goal state. An example of such a problem is a preprogrammed robot placed in a building with the goal of closing all the windows (where not necessarily all the windows will currently be open/closed).

Finally, we looked in Chapter 6 at how the proposed extensions can be implemented. Specifically, we looked at how they can be simulated using classical ASP. This offers us a number of advantages. Such simulations tend to be relatively straightforward, as they remain within the ASP framework and we can use performant off-the-shelf tools to compute the answer sets of an ASP program. We showed that CASP can indeed be simulated using ASP, but only if we do not consider focussing. Our first alternative semantics for PASP can also be simulated using ASP by noting that, in practice, only a finite number of certainty degrees need to be considered for the main decision problems. The second semantics for PASP, based on the concept of uncertain rules, can also be simulated using ASP. For the hardest decision problems our simulations rely on a translation of ASP to clauses (i.e. sets of disjunctions of literals).

In conclusion, we have proposed a number of extensions of classical ASP where on the one hand we want to reason about the knowledge of a network of ASP programs and, on the other hand, we want to reason about uncertain information. We showed with CASP that communication indeed influences the complexity of the overall network. In particular, we found that the choice of the communication mechanism crucially influences the complexity of the resulting system. With our first new semantics for PASP we showed that it is possible to find intuitive solutions for settings in which the existing semantics of PASP are unsatisfactory. In addition, these new semantics could be used to characterize ASP, which in turn allowed us to unearth a new form of disjunction in ASP with interesting complexity results. When we interpreted PASP as a set of uncertain rules, we obtained another semantics for PASP that is based on the notion of optional rules. We showed how the concept of optional rules can be used to model some important problems within Artificial Intelligence. Finally, we illustrated that a large part of the extensions that we proposed in this thesis can be simulated using classical ASP. As such, we have efficient implementations for both CASP and PASP.

For future work, improvements to both CASP and PASP can be envisaged, along with the combination of CASP and PASP in a single framework. Below we mention several of these potential extensions that seem very interesting, while fully realizing that this list is not exhaustive. Firstly, in CASP we currently consider a straightforward form of communication where programs can ask questions to each other. More elaborate forms of communication can be envisaged such as a blackboard architecture [Erman et al. 1980], where the different programs share a data region and a moderator controls when a program is allowed to modify the information in the shared data region. Also the topology of the programs could be taken into account. For example, it might be that only some programs are close enough to another program to be able to communicate. It would be interesting to analyse how such communication mechanisms affect the complexity. In addition, when using more complex forms of communication that allow e.g. a shared data region, then inconsistencies may arise. To deal with inconsistencies we would be required to implement techniques based on e.g. possibility theory [Benferhat et al. 2000] or paraconsistent logics [Bremer 2005].

Secondly, no implementation is currently available for computing focussed answer sets of a given CASP program. If we want to simulate focussed answer sets using classical ASP we would need to restrict ourselves to focussing on at most a single program in a communicating normal program. Due to the computational complexity of focussing, however, the general problem of finding the focussed answer sets cannot be implemented by simulating the problem in ASP without an exponential blow-up in the size of the program. Instead, for these problems we would need to rely on approaches with a higher computational complexity, such as certain variants of modal logics or expressing the problem as a Quantified Boolean Formula (QBF) and using solvers for QBF to compute the solutions.

Thirdly, when we looked at a PASP program as a set of uncertain rules, we encoded the certainty of each rule in such a way that it is independent of the certainty of the other rules. Using e.g. possibilistic networks [Benferhat and Smaoui 2007] would allow us to define the relationship between the uncertainties associated with each rule, i.e. it would allow us to specify the certainty that we have in a rule given that some other rules are found to be (in)correct. This would enhance our ability to capture the uncertainty that arises when reasoning about information coming from different sources. Indeed, if a rule from some source is found to be (in)correct, this may affect the certainty that we have towards the other rules encoding the information obtained from that same source.

Finally, combining CASP and PASP in a single framework introduces additional interesting challenges. We could choose to let each program individually and internally reason about uncertainty. This is, in essence, equivalent to introducing situated literals to PASP and considering a group of PASP programs. In addition, we might have that the communication between the programs itself is uncertain. For example, faulty connections could be modelled by associating a weight with the (currently implicit) connections that exist between the different programs.

Samenvatting

Answer Set Programmeren (ASP) is een domein-specifieke programmeertaal gebaseerd op logisch programmeren. Een ASP programma bestaat uit een verzameling van regels. Elke regel is van de vorm $conclusie \leftarrow voorwaarden$, waarbij de conclusie van een regel voldaan is wanneer alle voorwaarden van de regel voldaan zijn. Een probleem kan je oplossen met ASP door een ASP programma te schrijven waarbij de oplossingen van dit ASP programma, de answer sets, overeenkomen met de oplossingen van het originele probleem. Een belangrijke eigenschap van ASP is het gebruik van een '*negatie als falen*'-operator 'not' in de voorwaarden. Intuïtief is een uitdrukking zoals '$not\ zonnig$' waar wanneer er geen bewijs is dat het $zonnig$ is. Hierbij is '$zonnig$' een atoom, *m.a.w.* een logische formule die waar of vals kan zijn en die verder niet opgebouwd is uit andere formules. Het gebruik van een niet-monotone operator, waarbij we conclusies trekken uit de afwezigheid van informatie, wijkt af van het gebruik van klassieke negatie. Zo vereisen we bij klassieke negatie, wanneer we willen vaststellen dat '$\neg zonnig$' waar is, dat we een bewijs hebben om aan te tonen dat het niet zonnig is. In ASP gaan we ook klassieke negatie beschouwen in de vorm van een literaal, wat ofwel een atoom is of de klassieke negatie van een atoom.

Het gebruik van een niet-monotone operator in ASP beïnvloedt de expressiviteit van ASP en maakt het mogelijk om complexere problemen op te lossen. ASP programma's worden onderverdeeld in klassen, naargelang de vorm van de regels die er in voorkomen. De programma's met de laagste expressiviteit die we beschouwen zijn deze waarbij dat de not operator niet voorkomt in de voorwaarden en waarin alle regels een enkelvoudige conclusie hebben. Deze programma's worden *eenvoudige programma's* genoemd. De regels in een eenvoudig programma zijn van de vorm $(l_0 \leftarrow l_1, ..., l_m)$. *Normale* *programma's* zijn programma's waarbij we wel de not operator in de voorwaarden toelaten, maar waarbij alle regels nog steeds een enkelvoudige conclusie hebben. Deze regels zijn van de vorm $(l_0 \leftarrow l_1, ..., l_m, not\ l_{m+1}, ..., not\ l_n)$. Programma's waarin we de not operator in de voorwaarden toelaten en waarbij alle regels meervoudige conclusies kunnen hebben, worden *disjunctieve programma's* genoemd. Deze regels zijn dan van de vorm $(l_0; ...; l_k \leftarrow l_{k+1}, ..., l_m, not\ l_{m+1}, ..., not\ l_n)$. De expressiviteit van eenvoudige programma's, normale programma's en disjunctieve programma's is repsectievelijk P, NP en Σ_2^{P}.

De niet-monotone operator in ASP biedt ons de mogelijkheid om kennis te herzien. Dit doen we ook wanneer we zelf redeneren op basis van gezond verstand. Zo beslissen we bijvoorbeeld om de trein te nemen naar het werk, tenzij we te weten komen dat er een staking zal zijn. In ASP kunnen we echter niet eenvoudig redeneren over de kennis van een ander programma, want ASP wordt gedefiniëerd als één programma met meerdere regels. Nochtans zijn er talloze voorbeelden waarbij verschillende bronnen, of contexten, beschikken over verschillende stukken informatie. Zo heb je bij een politieonderzoek dat verschillende getuigen elk maar een deel van de feiten kennen. In deze thesis ligt de nadruk op het uitzoeken hoe informatie uitgewisseld kan worden tussen ASP programma's en welke invloed dit heeft op de complexiteit. We kunnen ASP ook niet intuïtief gebruiken om te redeneren over onzekerheid. Zo kunnen we bijvoorbeeld wel met ons gezond verstand redeneren over situaties zoals "de trein wordt *mogelijk* afgeschaft", terwijl er geen duidelijke manier bestaat om deze onzekere informatie in ASP te beschrijven. In deze thesis hebben we gekeken hoe ASP uitgebreid kan worden om te redeneren over kwalitatieve onzekerheid (uitdrukking zoals "*als ik redelijk zeker ben dat het buiten regent ga ik mijn paraplu openen*"), wat verschillend is van kwantitatieve onzekerheid ("*een brief wordt met 99% zekerheid de volgende dag geleverd*").

Om kennis afkomstig uit meerdere contexten en onzekere kennis te kunnen modelleren, hebben we in deze thesis epistemische uitbreidingen van ASP bestudeerd. Meer bepaald hebben we de uitbreiding Communicerende ASP (CASP) geïntroduceerd, waarbij meerdere ASP programma's kennis kunnen uitwisselen, en we hebben gekeken naar nieuwe semantieken voor Possibilistische ASP (PASP), waarbij we ASP uitbreiden met mechanismen om te redeneren over onzekerheid. Aanvullend hebben we de complexiteit van deze uitbreidingen bestudeerd om een goed idee te krijgen van hun complexiteit. Om alle begrippen die we in deze thesis beschrijven duidelijk te maken, hebben we voorafgaand werk omtrent complexiteit, ASP, possibiliteitstheorie en PASP beschreven in Hoofdstuk 2.

In Hoofdstuk 3 introduceerden we Communicerende ASP (CASP), waarbij we wilden nagaan hoe we meerdere ASP programma's kunnen laten communiceren en welke communicatiemechanismen de complexiteit beïnvloeden. We toonden hierbij aan dat de introductie van een nieuw soort literaal, $Q{:}l$, ons toelaat om informatie tussen programma's uit te wisselen. Deze literaal komt intuïtief overeen met het stellen van een vraag (*is de literaal 'l' waar?*) aan een ander programma (*namelijk aan Q*). Hierbij stelden we vast dat een netwerk van communicerende eenvoudige programma's (met een individuele expressiviteit die beperkt is tot P) dezelfde expressiviteit oplevert als een normaal programma (met een hogere expressiviteit van NP). Het is echter niet zo dat een netwerk van normale programma's expressief genoeg is om een disjunctief programma te beschrijven (met een complexiteit van Σ_2^{P}).

Een aanvullende uitbreiding op het idee van communicatie betreft het idee van 'focussen'. Normaal worden de oplossingen van een ASP programma beschreven als de minimale modellen van het programma. Met andere woorden: een oplossing is de minimaal noodzakelijke informatie die uit een ASP programma kan worden afgeleid. Wanneer er echter meerdere programma's gebruikt worden waartussen communicatie mogelijk is, is het idee van een minimaal model niet langer uniek gedefiniëerd. Wordt er met een minimaal model een algemeen minimaal model bedoeld, of een model dat minimaal is ten opzichte van één programma? Het idee van focussen dat we introduceerden in Sectie 3.4 is gebaseerd op deze laatste vorm van minimaliteit. Bovendien kan focussen herhaaldelijk toegepast worden en dan wordt het belangrijk in welke volgorde we dit toepassen. Wanneer we de complexiteit bestudeerden, dan konden we zien dat er, elke keer er gefocust wordt, een sprong gemaakt wordt in de polynomiale hiërarchie. Op deze manier is het mogelijk om alle problemen in PSPACE op te lossen met CASP, zolang er maar voldoende unieke programma's zijn om op te focussen.

Naast de mogelijkheid om meerdere programma's met elkaar te laten communiceren, hadden we ook interesse in het redeneren op basis van onzekere kennis. Possibilistische ASP is een bestaande uitbreiding op ASP [Nicolas et al. 2006] waarbij het mogelijk wordt om ASP te combineren met onzekere informatie en eveneens te redeneren op basis van deze onzekere informatie. Om dit mogelijk te maken wordt er syntactisch gezien een gewicht $\lambda \in]0,1]$ gekoppeld aan een regel. Een PASP programma is dus een verzameling van regels van de vorm $(\boldsymbol{\lambda}{:}\, conclusie \leftarrow voorwaarden)$. Dit gewicht kan geïnterpreteerd worden als de maximale zekerheid waarmee we de conclusie van een regel kunnen afleiden als de voorwaarden van die regel vervuld zijn. De huidige aanpak van PASP levert echter resultaten op die niet altijd intuïtief zijn. De oorzaak hiervan ligt in de specifieke interpretatie van de niet-monotone operator 'not'. Zo wordt '$not\ zonnig$' in de huidige aanpak van PASP geïnterpreteerd als "*er is geen enkele zekerheid dat het zonnig is*". Een probleem van deze interpretatie is dat we geen onderscheid kunnen maken tussen "ik ben redelijk zeker dat het '$zonnig$' is" en "ik ben absoluut zeker dat het '$zonnig$' is". We kunnen '$not\ zonnig$' echter ook interpreteren als "*de mate waarin we '$\neg zonnig$' kunnen aannemen*", wat op zijn beurt neerkomt op "*de mate waarin we $zonnig$ niet met zekerheid kunnen afleiden*".

In Hoofdstuk 4 bekeken we hoe we een semantiek op basis van deze nieuwe interpretatie kunnen definiëren aan de hand van restricties op possibiliteitsdistributies. Deze nieuwe semantiek biedt intuïtief aanvaardbare oplossingen in veel situaties waarin de oplossingen volgens de bestaande semantiek van PASP niet intuïtief zijn.

Aanvullend stelden we vast dat de nieuwe semantiek voor PASP gebruikt kan worden om de klassieke versie van ASP (zonder zekerheidsgraden) te karakteriseren. De karakterisatie van normale programma's kan hierbij op twee manieren uitgebreid worden naar disjunctieve programma's. Een van deze uitbreidingen is een karakterisatie van de klassieke semantiek, en de andere is een nieuwe semantiek. Deze nieuwe semantiek voor disjunctieve programma's behandelt de disjuncties vooral als een soort atoom. De complexiteit van deze nieuwe semantiek bevindt zich tussen de complexiteit van normale programma's en disjunctieve programma's wanneer klassieke negatie toegelaten wordt en valt samen met de complexiteit van normale programma's wanneer er geen klassieke negatie gebruikt wordt.

Zowel de bestaande semantiek voor PASP alsook de nieuwe semantiek die we voorstelden interpreteren het gewicht gekoppeld aan een regel als de maximale zekerheid waarmee we de conclusie van die regel kunnen afleiden. Op deze manier kunnen we met deze aanpak uitdrukken dat bepaalde conclusies slechts met een beperkte zekerheid gelden. We kunnen dit gewicht echter ook op een andere manier interpreteren, met name als de zekerheid waarmee de informatie die in de regel beschreven wordt daadwerkelijk geldig is. In dat geval hebben we onzekerheid omtrent ons model, eerder dan onzekerheid over de conclusies die we afleiden. Op deze manier krijgen we een verzameling van onzekere regels, eerder dan een verzameling van regels met onzekere conclusies. Om de semantiek van een verzameling van onzekere regels vast te leggen hebben we gekeken naar possibilistische logica, waarbij de meest onzekere regels weglaten worden. Zo zijn de conclusies die we in possibilistische logica met grote zekerheid kunnen afleiden de conclusies die we nog steeds kunnen afleiden nadat we de meest onzekere informatie weggelaten hebben. Het is echter zo, omdat ASP een niet-monotone operator bevat, dat het weglaten van regels ertoe kan leiden dat we nieuwe kennis afleiden. Deze nieuwe kennis kan echter foutief zijn. Daarom kunnen we niet zomaar de meest onzekere regels weglaten en beschouwen we in de plaats daarvan – conceptueel gezien – alle deelverzamelingen van onzekere regels. Elke deelverzameling komt dan overeen met net die regels waarvan we uitgaan dat ze geldig zijn. Met zo een deelverzameling konden we vervolgens een graad van mogelijkheid associëren, waarbij een deelverzameling van regels minder mogelijk wordt naarmate er regels met een steeds hogere zekerheid uit weggelaten worden.

In Hoofdstuk 5 bespraken we een semantiek gebaseerd op dit idee van onzekere regels. In dit hoofdstuk gingen we bovendien de vergelijking aan met de reeds besproken semantieken uit Hoofdstuk 4 en bestudeerden we de complexiteit van de belangrijkste beslissingsproblemen van deze nieuwe semantiek in detail. We konden deze semantiek van onzekere regels bovendien ook beschouwen als een semantiek van optionele regels. Een aantal belangrijke problemen binnen Artificiële Intelligentie kunnen we beschrijven als problemen met optionele regels en konden we dan ook met PASP oplossen. Een belangrijk planningsprobleem is bijvoorbeeld het probleem waarbij we niet zeker zijn van de begintoestand, maar waarbij we desondanks een betrouwbaar plan willen om een bepaalde eindtoestand te bereiken. Een voorbeeld van een dergelijk probleem is een voorgeprogrammeerde robot die in een gebouw alle ramen moet sluiten (waarbij niet noodzakelijk alle ramen open of gesloten zijn).

Ten slotte bekeken we in Hoofdstuk 6 hoe we de hiervoor besproken uitbreidingen kunnen implementeren. We hebben daarbij nagegaan of het mogelijk is om deze uitbreidingen te simuleren met ASP. Dit biedt een aantal voordelen. Dergelijke simulaties zijn veelal eenvoudiger omdat we binnen het raamwerk van ASP blijven. Daarnaast kunnen we gebruik maken van vrij te verkrijgen programma's om answer sets van een ASP programma te berekenen. We toonden aan dat CASP inderdaad gesimuleerd kan worden met ASP, maar enkel als we geen 'focussing' beschouwen. De semantiek voor PASP die we voorgesteld hebben in Hoofdstuk 4 kon ook gesimuleerd worden met ASP dankzij de vaststelling dat we hiervoor slechts een eindig aantal zekerheidsgraden moeten beschouwen. De tweede semantiek voor PASP, op basis van onzekere regels, konden we eveneens simuleren met ASP. Voor de moeilijkste beslissingsproblemen maken we hierbij gebruik van bestaande vertaling van ASP naar clausules, waarbij een clausule een disjunctie van literalen is.

Samenvattend hebben we uitbreidingen van ASP voorgesteld waarbij we enerzijds willen redeneren over kennis van meerdere ASP programma's en anderzijds willen redeneren over onzekere kennis. We toonden aan met CASP dat communicatie de complexiteit inderdaad beïnvloedt, alhoewel de mate van invloed sterk afhangt van de keuze van communicatiemechanisme. Op basis van onze eerste nieuwe semantiek voor PASP toonden we aan dat we intuïtieve oplossingen kunnen vinden voor problemen waar de bestaande semantiek voor PASP niet toereikend is. Bovendien kon deze semantiek gebruikt worden om ASP te karakteriseren, wat op zijn beurt geleid heeft tot een nieuwe vorm van disjunctie binnen ASP met interessante complexiteitseigenschappen. Beschouwden we PASP als een verzameling van onzekere regels, dan bekwamen we een semantiek die dicht aanleunt bij het idee van optionele regels. We toonden hoe dergelijke optionele regels gebruikt kunnen worden om belangrijke problemen binnen Artificiële Intelligentie te modelleren. We stelden ten slotte vast dat het grootste deel van de uitbreidingen die in deze thesis voorgesteld werden te simuleren zijn door middel van ASP. Daardoor beschikken we over efficiënte implementaties voor zowel CASP als PASP.

Met het oog op toekomstig onderzoek kunnen we kijken naar verbeteringen van zowel CASP en PASP, maar ook naar de combinatie van CASP en PASP. Hieronder vermelden we een aantal uitbreidingen die er heel interessant uit zien, maar waarbij we ons uiteraard realiseren dat deze opsomming van mogelijke uitbreidingen niet compleet is. Ten eerste, als we enkel kijken naar CASP, dan stellen we vast dat we momenteel een voor de hand liggende manier van communicatie beschouwen waarbij programma's vragen aan elkaar kunnen stellen. We kunnen ook meer omvangrijke manieren van communicatie gebruiken zoals een blackboard architectuur [Erman et al. 1980]. In zo een architectuur kunnen meerdere programma's informatie delen via een gezamenlijk informatiebord en bepaalt een bemiddelaar welk programma op welk moment de toestemming krijgt om de informatie op het gedeelde bord aan te passen. Ook de topologie van het probleem kan interessant zijn. Zo kan het bijvoorbeeld zijn dat slechts een aantal programma's zich dicht genoeg bij een ander programma bevinden om met dat programma te kunnen communiceren. Hierbij zou het interessant zijn om na te gaan hoe dergelijke communicatiemechanismen de complexiteit beïnvloeden. Aanvullend is het zo, wanneer we bijvoorbeeld een gedeeld informatiebord gebruiken, dat er tegenstrijdigheden kunnen onstaan. Om met dergelijke tegenstrijdigheden overweg te kunnen, moeten we technieken gebruiken gebaseerd op bijvoorbeeld possibiliteitstheorie [Benferhat et al. 2000] of paraconsistente logica's [Bremer 2005].

Ten tweede beschikken we op dit moment niet over een implementatie voor het berekenen van gefocuste answer sets. Als we dit probleem willen simuleren met ASP, dan moeten we ons beperken tot een netwerk van normale programma's waarbij we maximaal op één programma focussen. Omwille van de complexiteit van focussen kunnen we echter in het algemeen geval het probleem van het bepalen van de gefocuste answer sets niet meer simuleren met ASP. In de plaats daarvan moeten we gebruik maken van aanpakken met een hogere complexiteit, zoals bepaalde varianten van modale logica's of het simuleren van het probleem als een QBF waarbij we vervolgens implementaties voor het oplossen van QBFs kunnen gebruiken om de oplossing te berekenen.

Ten derde gingen we er, wanneer we in PASP het gewicht interpreteerden als de zekerheid waarmee de informatie in die regel inderdaad geldig is, vanuit dat de zekerheid van de regels onderling onafhankelijk was. Door gebruik te maken van possibilistische netwerken [Benferhat and Smaoui 2007] wordt het mogelijk om de relatie tussen de onzekerheidsgraden van verschillende regels te beschrijven, *m.a.w.* het wordt mogelijk om de zekerheid van een regel aan te passen als we te weten komen dat een bepaalde regel al dan niet geldig is. Een dergelijke uitbreiding maakt het mogelijk om informatie afkomstig van verschillende bronnen beter voor te stellen. Zo is het aannemelijk om te veronderstellen, wanneer een regel die informatie beschrijft van een bepaalde bron (on)geldig blijkt te zijn, dat dit een invloed zal hebben op onze zekerheid omtrent de andere regels die informatie beschrijven van dezelfde bron.

Tenslotte ontstaan er nieuwe en interessante uitdagingen wanneer we CASP en PASP willen combineren. We kunnen er bijvoorbeeld voor kiezen om elke programma individueel en intern te laten redeneren over onzekerheid. Dit komt ruwweg neer op het introduceren van literalen van de vorm $Q:l$ in een netwerk van PASP programma's. Aanvullend kunnen we hebben dat de communicatie tussen de verschillende programma's zelf onzeker is. Op basis van deze uitbreiding kunnen we bijvoorbeeld een gewicht koppelen aan het (op dit moment impliciete) communicatiekanaal tussen twee verschillende programma's om op deze manier onbetrouwbare kanalen te modeleren.

Proofs

Proofs of Chapter 3

Proposition 6

Let $\mathcal{P}$ be a communicating simple program. We then have that:

- there always exists at least one answer set of $\mathcal{P}$;
- there is always a unique answer set of $\mathcal{P}$ that is globally minimal;
- we can compute this unique globally minimal answer set in polynomial time.

Proof. We can easily generalise the immediate consequence operator for (classical) simple programs to the case of communicating simple programs. Specifically, the operator $T_{\mathcal{P}}$ is defined w.r.t. an interpretation I of $\mathcal{P}$ as

$$T_{\mathcal{P}}(I) = I \cup \{Q\!:\!l \mid (Q\!:\!l \leftarrow \alpha) \in Q, Q \in \mathcal{P}, \alpha \subseteq I\}$$

where α is a set of $\mathcal{P}$-situated literals. It is easy to see that this operator is monotone. Together with a result from [Tarski 1955] we know that this operator has a least fixpoint. We use $\mathcal{P}^\star$ to denote this fixpoint obtained by repeatedly applying $T_{\mathcal{P}}$ starting from the empty interpretation. Clearly, this fixpoint can be computed in polynomial time.

We need to verify that $\mathcal{P}^\star$ is indeed an answer set. Since $\mathcal{P}$ is a communicating simple program, we know that the reduct $Q^{\mathcal{P}^\star}$ will only remove rules that contain situated literals

$R\!:\!l$ that are not Q-local with $R\!:\!l \notin \mathcal{P}^\star$. In other words, rules that are not applicable ($\alpha \not\subseteq \mathcal{P}^\star$) and that contain non-$Q$-local situated literals are removed. Furthermore, remaining situated literals of the form $R\!:\!l$ that are not Q-local (i.e. those where $R\!:\!l \in \mathcal{P}^\star$) are removed from the body of the remaining rules. Hence the remaining rules are all Q-local. Notice that the operator $T_\mathcal{P}$ is clearly an extension of the operator T_Q. Indeed, for a component simple program Q' that is Q'-local it is easy to verify that if $(Q')^\star = M'$ then $((\mathcal{P}')^\star)_{Q'} = M'$ with $\mathcal{P}' = \{Q'\}$. It then readily follows, since all rules are Q-local and therefore independent of all other component programs, that $\left(Q^{\mathcal{P}^\star}\right)^\star = (\mathcal{P}^\star)_Q$ for all $Q \in \mathcal{P}$.

So far we found that an answer set exists and that it can be computed in polynomial time. All that remains to be shown is that this answer set is globally minimal. This trivially follows from the way we defined the operator $T_\mathcal{P}$ since it only makes true the information that is absolutely necessary, i.e. the information that follows directly from the facts in the communicating simple program. Hence this is the minimal amount of information that needs to be derived for a set of situated literals to be a model of the communicating simple program at hand and thus the fixpoint $\mathcal{P}^\star$ is the globally minimal answer set. $\square$

Lemma 1

Let $\mathcal{P} = \{Q_1, ..., Q_n\}$ and let $\mathcal{P}' = \{Q'_1, ..., Q'_n, N_1, ..., N_n\}$ with $\mathcal{P}$ a communicating normal program and $\mathcal{P}'$ the communicating simple program that simulates $\mathcal{P}$ defined in Definition 11. Let M be an answer set of $\mathcal{P}$ and let the interpretation M' be defined as:

$$\begin{aligned} M' = &\{Q'_i\!:\!a \mid Q_i\!:\!a \in M\} \\ &\cup \left\{Q'_i\!:\!\neg b^\dagger \mid Q_i\!:\!b \notin M\right\} \\ &\cup \left\{N_i\!:\!\neg b^\dagger \mid Q_i\!:\!b \notin M\right\} \\ &\cup \left\{N_i\!:\!a^\dagger \mid Q_i\!:\!a \in M\right\}. \end{aligned} \tag{3.5}$$

For each $i \in \{1, ..., n\}$ it holds that $(Q'_i{+})^{M'} = \left\{l \leftarrow \alpha' \mid l \leftarrow \alpha \in Q_i^M\right\}$ with $Q'_i{+}$ the set of rules defined in (3.1) with $\alpha' = \{Q'_i\!:\!b \mid Q_i\!:\!b \in \alpha\}$.

Proof. To prove this, we first show that any rule of the form $(l \leftarrow \alpha) \in Q_i^M$ reappears in $(Q'_i{+})^{M'}$ under the form $(l \leftarrow \alpha')$ for any $i \in \{1, ..., n\}$. The second step, showing that the converse also holds, can then be done in an analogous way.

Suppose $(l \leftarrow \alpha) \in Q_i^M$ for some $i \in \{1, ..., n\}$. By the definition of the reduct we know that there is some rule of the form $(l \leftarrow \alpha \cup \mathit{not}\ \beta \cup \gamma) \in Q_i$ such that $\beta \cap M = \emptyset$

and $\gamma \subseteq M$ is a set of situated literals of the form $Q_j\!:\!d$ with $i \neq j$, $1 \leq j \leq n$. From Definition 11, we know that the communicating normal rule $(Q_i\!:\!l \leftarrow \alpha \cup \mathit{not}\ \beta \cup \gamma)$ is transformed into the rule $(Q'_i\!:\!l \leftarrow \alpha' \cup \beta' \cup \gamma')$ with $\alpha' = \{Q'_i\!:\!b \mid Q_i\!:\!b \in \alpha\}$, $\beta' = \left\{N_k\!:\!\neg c^\dagger \mid Q_k\!:\!c \in \beta, k \in \{1,...,n\}\right\}$ and $\gamma' = \left\{Q'_j\!:\!d \mid Q_j\!:\!d \in \gamma, j \in \{1,...,n\}\right\}$. We show that, indeed, $(l \leftarrow \alpha) \in Q_i^M$ reappears in $(Q'_i+)^{M'}$ under the form $(l \leftarrow \alpha')$.

First, whenever $Q_k\!:\!c \in \beta$, we know that $Q_k\!:\!c \notin M$ since $\beta \cap M = \emptyset$. From the construction of M' we have that $N_k\!:\!\neg c^\dagger \in M'$. Similarly, since $\gamma \subseteq M$ we know from the construction of M' that $Q'_j\!:\!d \in M'$ whenever $Q_j\!:\!d \in \gamma$. Hence when determining the reduct $(Q'_i+)^{M'}$, the extended situated literals in β' and γ' will be deleted.

Finally, whenever $\alpha \cap M \neq \emptyset$ we know from the construction of M' that $Q'_i\!:\!b \in M'$ whenever $Q_i\!:\!b \in \alpha$. Clearly, when determining the reduct, none of these extended situated literals will be deleted as they are Q'_i-local. Hence it is clear that the reduct of the communicating rule $(Q'_i\!:\!l \leftarrow \alpha' \cup \beta' \cup \gamma')$ is the rule $Q'_i\!:\!l \leftarrow \alpha'$. This completes the first part of the proof. As indicated, the second part of the proof is completely analogous. $\square$

Proposition 7

Let $\mathcal{P} = \{Q_1,...,Q_n\}$ and let $\mathcal{P}' = \{Q'_1,...,Q'_n,N_1,...,N_n\}$ with $\mathcal{P}$ a communicating normal program and $\mathcal{P}'$ the communicating simple program that simulates $\mathcal{P}$ as defined in Definition 11. If M is an answer set of $\mathcal{P}$, then M' is an answer set of $\mathcal{P}'$ with M' defined as in Lemma 1.

Proof. This proof is divided into two parts. In part 1 we only consider the component programs Q'_i with $i \in \{1,...,n\}$ and show that $\left((Q'_i)^{M'}\right)^\star = (M')_{Q'_i}$. In part 2 we do the same, but we only consider the component programs N_i with $i \in \{1,...,n\}$. As per Definition 10 we have then shown that M' is indeed an answer set of $\mathcal{P}'$.

Consider a component program Q'_i with $i \in \{1,...,n\}$. By Definition 11 we have that $Q'_i = (Q'_i+) \cup (Q'_i-)$ and thus

$$(Q'_i)^{M'} = (Q'_i+)^{M'} \cup (Q'_i-)^{M'}. \tag{6.14}$$

For Q'_i- we know by construction that it only contains rules that are of the form $(Q'_i\!:\!\neg b^\dagger \leftarrow N_i\!:\!\neg b^\dagger)$ and that the only rules of this form are in Q'_i-. Therefore, due to the definition of the reduct, we have

$$(Q'_i-)^{M'} = \left\{\neg b^\dagger \leftarrow \mid N_i\!:\!\neg b^\dagger \in M'\right\}$$

and because of the construction of M', see (3.5), we obtain

$$(Q'_i-)^{M'} = \left\{\neg b^\dagger \leftarrow \mid b \notin M_{Q_i}\right\}. \tag{6.15}$$

Hence $(Q'_i-)^{M'}$ only contains facts about literals that, by construction of Q'_i, do not occur in Q'_i+. This means that from (6.14) and (6.15) we obtain

$$\left((Q'_i)^{M'}\right)^\star = \left((Q'_i+)^{M'}\right)^\star \cup \left\{\neg b^\dagger \leftarrow \mid b \notin M_{Q_i}\right\}. \tag{6.16}$$

From Lemma 1 we know that $(Q'_i+)^{M'} = \left\{l \leftarrow \alpha' \mid l \leftarrow \alpha \in Q_i^M\right\}$ where we have that $\alpha' = \{Q'_i:b \mid Q_i:b \in \alpha\}$. Because of the definition of an answer set of a communicating program we have

$$M_{Q_i} = \left(Q_i^M\right)^\star = \left((Q'_i+)^{M'}\right)^\star. \tag{6.17}$$

Combining this with (6.16) we get

$$\begin{aligned}\left((Q'_i)^{M'}\right)^\star &= M_{Q_i} \cup \left\{\neg b^\dagger \mid b \notin M_{Q_i}\right\} \\ &= (M')_{Q'_i} && \text{(definition of } M'\text{, see (3.5))}\end{aligned}$$

This concludes the first part of the proof.

In the second part of the proof, we only consider the component programs N'_i with $i \in \{1, ..., n\}$. By construction of N_i we know that all the rules of the form $\neg b^\dagger \leftarrow Q'_i:\neg b^\dagger$ and $b^\dagger \leftarrow Q'_i:b$ are in N_i and that all the rules in N_i are of this form. We have

$$(N_i)^{M'} = \left\{\neg b^\dagger \leftarrow \mid Q'_i:\neg b^\dagger \in M'\right\} \cup \left\{b^\dagger \leftarrow \mid Q'_i:b \in M'\right\}$$

which, due to the definition of M' can be written as

$$= \left\{\neg b^\dagger \leftarrow \mid b \notin M_{Q_i}\right\} \cup \left\{b^\dagger \leftarrow \mid b \in M_{Q_i}\right\}$$

from which it follows that $\left((N_i)^{M'}\right)^\star = (M')_{N_i}$. □

Lemma 2

Let $\mathcal{P} = \{Q_1, ..., Q_n\}$ and let $\mathcal{P}' = \{Q'_1, ..., Q'_n, N_1, ..., N_n\}$ with $\mathcal{P}$ a communicating normal program and $\mathcal{P}'$ the communicating simple program that simulates $\mathcal{P}$. Assume that M' is an answer set of $\mathcal{P}'$ and that $(M')_{N_i}$ is total w.r.t. $\mathcal{B}_{N_i}$ for all $i \in \{1, ..., n\}$. Let M be defined as

$$M = \left\{Q_i:b \mid Q'_i:b \in \left((Q'_i+)^{M'}\right)^\star\right\} \tag{3.6}$$

For each $i \in \{1, ..., n\}$, it holds that $(Q'_i{+})^{M'} = \left\{l \leftarrow \alpha' \mid l \leftarrow \alpha \in Q_i^M\right\}$ with $\alpha' = \{Q'_i{:}b \mid Q_i{:}b \in \alpha\}$.

Proof. To prove this, we first show that any rule of the form $(l \leftarrow \alpha') \in (Q'_i{+})^{M'}$ reappears in Q_i^M under the form $l \leftarrow \alpha$ for any $i \in \{1, ..., n\}$. We then show that the converse also holds, which is rather analogous to the proof of the first part of Lemma 1. Due to some technical subtleties in the second part of the proof, however, we present the proof in detail.

Suppose $(l \leftarrow \alpha') \in (Q'_i{+})^{M'}$. By the definition of the reduct of a communicating simple program we know that there is some communicating simple rule of the form $(l \leftarrow \alpha' \cup \beta' \cup \gamma') \in Q'_i{+}$ such that $\beta' \subseteq M'$ is a set of situated literals of the form $N_k{:}\neg c^\dagger$ and $\gamma' \subseteq M'$ is a set of situated literals of the form $Q'_j{:}d$ with $i \neq j$ and $1 \leq j, k \leq n$.

From the definition of $Q'_i{+}$, we know that $(Q'_i{:}l \leftarrow \alpha' \cup \beta' \cup \gamma')$ corresponds to a rule $(l \leftarrow \alpha \cup \mathit{not}\ \beta \cup \gamma) \in Q_i$ where we have that $\alpha = \{Q_i{:}b \mid Q'_i{:}b \in \alpha'\}$, $\beta = \left\{Q_k{:}c \mid N_k{:}\neg c^\dagger \in \beta'\right\}$ and $\gamma = \left\{Q_j{:}d \mid Q'_j{:}d \in \gamma'\right\}$. We show that, indeed, $(l \leftarrow \alpha') \in (Q'_i{+})^{M'}$ reappears in Q_i^M under the form $(l \leftarrow \alpha)$.

First, since $\beta' \subseteq M'$, whenever $N_k{:}\neg c^\dagger \in \beta'$ we know that $N_k{:}\neg c^\dagger \in M'$. Since M' is a model (indeed, it is an answer set) it is an interpretation (and thus consistent). Therefore, if $N_k{:}\neg c^\dagger \in M'$ then surely $N_k{:}c^\dagger \notin M'$. Now, if we were to have $Q'_k{:}c \in M'$, then applying the immediate consequence operator on the rule $N_k{:}c^\dagger \leftarrow Q'_k{:}c$ found in the component program N_k would force us to have $N_k{:}c^\dagger \in M'$ which results in a contradiction. Hence we find that $Q'_k{:}c \notin M'$. By Definition 11 we know that $Q'_k = (Q'_k{+}) \cup (Q'_k{-})$ and thus, by the definition of the reduct, we know that $(Q'_k)^{M'} = (Q'_k{+})^{M'} \cup (Q'_k{-})^{M'}$. Then we find that $((Q'_k)^{M'})^\star = ((Q'_k{+})^{M'})^\star \cup ((Q'_k{-})^{M'})^\star$ since all the rules in $Q'_k{-}$ have fresh literals in the head and literals from N_k in the body and hence cannot interact with the rules from $Q'_k{+}$ which only depend on information derived from $Q'_k{+}$ and N_k in their bodies. Recall from the definition of an answer set of a communicating program that $\forall Q'_k \in \mathcal{P}' \cdot (Q'_k{:}M'_{Q'_k}) = \left((Q'_k)^{M'}\right)^\star$. Since we already found that $Q'_k{:}c \notin M'$ we must have $Q'_k{:}c \notin \left((Q'_i{+})^{M'}\right)^\star$, or, because of the definition of M, that $Q_k{:}c \notin M$. Hence when determining the reduct $(l \leftarrow \alpha \cup \mathit{not}\ \beta \cup \gamma)^M$, the extended situated literals in $\mathit{not}\ \beta$ will be deleted.

In a similar way of reasoning, since $\gamma' \subseteq M'$ and because $\gamma = \left\{Q_j{:}d \mid Q'_j{:}d \in \gamma'\right\}$ we know from the construction of M that $\gamma \subseteq M$. Hence when determining the reduct, the situated literals in γ will be deleted. Finally, since $\alpha' \subseteq M'$ and because $\{Q_i{:}b \mid Q'_i{:}b \in \alpha'\} \subseteq M$ we know from the construction of M that $\alpha \in M$. Clearly, when determining the reduct, none of the situated literals in α will be deleted as they are Q_i-local. Hence the reduct of the communicating rule $(Q_i : l \leftarrow \alpha \cup not\ \beta \cup \gamma)$ is the rule $Q_i : l \leftarrow \alpha$. This completes the first part of the proof.

We now come to the second part. This time we show that any rule of the form $(l \leftarrow \alpha) \in Q_i^M$ reappears in $(Q_i'+)^{M'}$ under the form $(l \leftarrow \alpha')$ for any $i \in \{1, ..., n\}$.

Suppose $(l \leftarrow \alpha) \in Q_i^M$ for some $i \in \{1, ..., n\}$. By the definition of the reduct we know that there is some rule of the form $(l \leftarrow \alpha \cup not\ \beta \cup \gamma) \in Q_i$ such that $\beta \cap M = \emptyset$ and $\gamma \subseteq M$ is a set of situated literals of the form $Q_j : d$ with $i \neq j$, $1 \leq j \leq n$. From Definition 11, we know that the communicating normal rule $(Q_i : l \leftarrow \alpha \cup not\ \beta \cup \gamma)$ is transformed into the rule $(Q_i' : l \leftarrow \alpha' \cup \beta' \cup \gamma')$ with $\alpha' = \{Q_i' : b \mid Q_i : b \in \alpha\}$, $\beta' = \left\{N_k : \neg c^\dagger \mid Q_k : c \in \beta, k \in \{1, ..., n\}\right\}$ and $\gamma' = \left\{Q_j' : d \mid Q_j : d \in \gamma, j \in \{1, ..., n\}\right\}$. We show that, indeed, $(l \leftarrow \alpha) \in Q_i^M$ reappears in $(Q_i'+)^{M'}$ under the form $(l \leftarrow \alpha')$ when M'_{N_i} is total w.r.t. $\mathcal{B}_{N_i}$ for all $i \in \{1, ..., n\}$.

First, since $\gamma \subseteq M$ we know from the construction of M that $Q_j : d \in M$ whenever $Q_j' : d \in \gamma'$. Also, when $Q_k : c \in \beta$, we know that $Q_k : c \notin M$ since $\beta \cap M = \emptyset$. From the construction of M we then know that $Q_k' : c \notin M'$ and since M' is an answer set we readily obtain that $N_k : c^\dagger \notin M'$ due to the construction of N_k. Together with the requirement that M'_{N_k} is total w.r.t. $\mathcal{B}_{N_k}$ we then must have that $N_k : \neg c^\dagger \in M'$. Hence when determining the reduct $(Q_i'+)^{M'}$, the extended situated literals in β' and γ' will be deleted.

Finally, whenever $\alpha \cap M \neq \emptyset$ we know from the construction of M' that $Q_i' : b \in M'$ whenever $Q_i : b \in (\alpha \cap M)$. Clearly, when determining the reduct, none of these extended situated literals will be deleted as they are Q_i'-local. Hence it is clear that the reduct of the communicating rule $(Q_i' : l \leftarrow \alpha' \cup \beta' \cup \gamma')$ is the rule $Q_i' : l \leftarrow \alpha'$. This completes the second part of the proof. □

Proposition 8

Let $\mathcal{P} = \{Q_1, ..., Q_n\}$ and let $\mathcal{P}' = \{Q_1', ..., Q_n', N_1, ..., N_n\}$ with $\mathcal{P}$ a communicating normal program and $\mathcal{P}'$ the communicating simple program that simulates $\mathcal{P}$. Assume that M' is an answer set of $\mathcal{P}'$ and that $(M')_{N_i}$ is total w.r.t. $\mathcal{B}_{N_i}$ for all $i \in \{1, ..., n\}$. Then the interpretation M defined in Lemma 2 is an answer set of $\mathcal{P}$.

Proof. Lemma 2 tells us that $(Q_i'+)^{M'} = \left\{l \leftarrow \alpha' \mid l \leftarrow \alpha \in Q_i^M\right\}$ where we have $\alpha' = \{Q_i' : b \mid Q_i : b \in \alpha\}$. Hence we have $\left((Q_i'+)^{M'}\right)^\star = \left(Q_i^M\right)^\star$ since repeatedly applying the immediate consequence operator must conclude the same literals l due to the correspondence of the rules in the reducts and because of the way α' is defined. Since we defined M

as

$$\left\{Q_i\!:\!b \mid Q'_i\!:\!b \in \left(\left(Q'_i+\right)^{M'}\right)^{\star}\right\}$$

it follows immediately that M is an answer set of $\mathcal{P}$ since

$$\forall i \in \{1, ..., n\} \cdot \left(Q_i^M\right)^{\star} = M_{Q_i} \tag{6.18}$$

which completes the proof. □

Proposition 9

Let $\mathcal{P}$ be a communicating simple program. We then have:

- there always exists at least one $(Q_1, ..., Q_n)$-focused answer set of $\mathcal{P}$;
- we can compute this $(Q_1, ..., Q_n)$-focused answer set in polynomial time.

Proof. We know from Proposition 6 that we can always find a globally minimal answer of $\mathcal{P}$ in polynomial time. Due to the way we defined the immediate fixpoint operator $T_{\mathcal{P}}$ this operator only makes true the information that is absolutely necessary, i.e. the minimal amount of information that can be derived (for each component program). It is then easy to see that no component program can derive any less information (we have no negation-as-failure) and thus that this globally minimal answer set is also locally minimal and thus a $(Q_1, ..., Q_n)$-focused answer set of $\mathcal{P}$. □

Proposition 10

Let ϕ and $\mathcal{P}$ be as in Definition 13. We have that a QBF ϕ of the form $\phi = \exists X_1 \forall X_2 \cdots \Theta X_n \cdot p(X_1, X_2, \cdots X_n)$ is satisfiable if and only if $Q_0\!:\!sat$ is true in some $(Q_1, ..., Q_{n-1})$-focused answer set of $\mathcal{P}$. Furthermore, we have that a QBF ϕ of the form $\phi = \forall X_1 \exists X_2 \cdots \Theta X_n \cdot p(X_1, X_2, \cdots X_n)$ is satisfiable if and only if $Q_0\!:\!sat$ is true in all $(Q_1, ..., Q_{n-1})$-focused answer sets of $\mathcal{P}$.

Proof. We give a proof by induction. Assume we have a QBF ϕ_1 of the form $\exists X_1 \cdot p(X_1)$ with $\mathcal{P}_1 = \{Q_0\}$ the communicating normal program corresponding with ϕ_1 according to Definition 13. If the formula $p_1(X_1)$ of the QBF ϕ_1 is satisfiable then we know that there is a ()-focused answer set M of $\mathcal{P}_1$ such that $Q_0\!:\!sat \in M$. Otherwise, we know that $Q_0\!:\!sat \notin M$ for all ()-answer sets M of $\mathcal{P}_1$. Hence the induction hypothesis is valid for $n = 1$.

Assume the result holds for any QBF ϕ_{n-1}, which is an expression that is of the form $\exists X_1 \forall X_2 \ldots \Theta X_{n-1} \cdot p_{n-1}(X_1, X_2, ..., X_{n-1})$. We show in the induction step that it holds for any QBF ϕ_n of the form $\exists X_1 \forall X_2 \ldots \overline{\Theta} X_n \cdot p_n(X_1, X_2, ..., X_n)$. Let $\mathcal{P} = \{Q_0, ..., Q_{n-1}\}$ and $\mathcal{P}' = \{Q'_0, ..., Q'_{n-2}\}$ be the communicating normal programs that correspond with ϕ_n and ϕ_{n-1}, respectively. Note that the component programs $Q_2, ..., Q_{n-1}$ are defined in exactly the same way as the component programs $Q'_1, ..., Q'_{n-2}$, the only difference being the name of the component programs. What is of importance in the case of ϕ_n is therefore only the additional rules in Q_0 and the new component program Q_1. The additional rules in Q_0 merely generate the corresponding interpretations, where we now need to consider the possible interpretations of the variables from X_n as well. The rules in the new component program Q_1 ensure that $Q_1{:}x \in M$ whenever $Q_0{:}x \in M$ and $Q_1{:}\neg x \in M$ whenever $Q_0{:}\neg x \in M$ for every M an answer set of $\mathcal{P}$ and $x \in (X_1 \cup ... \cup X_{n-1})$. Depending on n being even or odd, we get two distinct cases:

- if n is even, then we have $(sat \leftarrow Q_0{:}sat) \in Q_1$ and we know that the QBF ϕ_n has the form $\exists X_1 \forall X_2 \ldots \forall X_n \cdot p_n(X_1, X_2, ..., X_n)$. Let us consider what happens when we determine the (Q_1)-focused answer sets of $\mathcal{P}$. Due to the construction of Q_1, we know that $M'_{Q_1} \subset M_{Q_1}$ can only hold for two answer sets M' and M of $\mathcal{P}$ if M' and M correspond to identical interpretations of the variables in $X_1 \cup ... \cup X_{n-1}$. Furthermore, $M'_{Q_1} \subset M_{Q_1}$ is only possible if $Q_1{:}sat \in M$ while $Q_1{:}sat \notin M'$.

  Now note that given an interpretation of the variables in $X_1 \cup ... \cup X_{n-1}$, there is exactly one answer set for each choice of X_n. When we have M' with $Q_1{:}sat \notin M'$ this implies that there is an interpretation such that, for some choice of X_n, this particular assignment of values of the QBF does not satisfy the QBF. Similarly, if we have M with $Q_1{:}sat \in M$ then the QBF is satisfied for that particular choice of X_n. Determining (Q_1)-focused answer sets of $\mathcal{P}$ will eliminate M since $M'_{Q_1} \subset M_{Q_1}$. In other words, for identical interpretations of the variables in $X_1 \cup ... \cup X_{n-1}$, the answer set M' encodes a counter-example that shows that for these interpretations it does not hold that the QBF is satisfied for all choices of X_n. Focussing thus eliminates those answer sets that claim that the QBF is satisfiable for the variables in $X_1 \cup ... \cup X_{n-1}$. When we cannot find such $M'_{Q_1} \subset M_{Q_1}$ this is either because none of the interpretations satisfy the QBF or all of the interpretations satisfy the QBF. In both cases, there is no need to eliminate any answer sets. We thus effectively mimic the requirement that the QBF ϕ_n should hold for all X_n.

- if n is odd, then $(\neg sat \leftarrow Q_0{:}\neg sat) \in Q_1$ and we know that the QBF ϕ_n has the form $\exists X_1 \forall X_2 \ldots \exists X_n \cdot p_n(X_1, X_2, ..., X_n)$. As before, we know that $M'_{Q_1} \subset M_{Q_1}$ can only hold for two answer sets M' and M of $\mathcal{P}$ if M' and M correspond to identical interpretations of the variables in $X_1 \cup ... \cup X_{n-1}$. However, this time $M'_{Q_1} \subset M_{Q_1}$ is only possible if $Q_1 \!:\! \neg sat \in M$ while $Q_1 \!:\! \neg sat \notin M'$.

If we have M with $Q_1 \!:\! \neg sat \in M$ then the QBF is not satisfied for that particular choice of X_n, whereas when M' with $Q_1 \!:\! \neg sat \notin M'$ then there is an interpretation such that, for some choice of X_n, this particular assignment of the variables does satisfy the QBF. Determining (Q_1)-focused answer sets of $\mathcal{P}$ will eliminate M since $M'_{Q_1} \subset M_{Q_1}$. For identical interpretations of the variables in $X_1 \cup ... \cup X_{n-1}$, the answer set M' encodes a counter-example that shows that for these interpretations there is some choice of X_n such that the QBF is satisfied. Focussing thus eliminates those answer sets that claim that the QBF is not satisfiable for the variables in $X_1 \cup ... \cup X_{n-1}$. When we cannot find such $M'_{Q_1} \subset M_{Q_1}$ this is either because none of the interpretations satisfy the QBF or all of the interpretations satisfy the QBF. In both cases, there is no need to eliminate any answer sets. We effectively mimic the requirement that the QBF ϕ_n should hold for some X_n.

For a QBF of the form $\forall X_1 \exists X_2 \ldots \Theta X_n \cdot p(X_1, X_2, ..., X_n)$, with $\Theta = \exists$ if n is even and $\Theta = \forall$ otherwise, the proof is analogous. In the base case, we know that a QBF ϕ_1 of the form $\forall X_1 \cdot p(X_1)$ is satisfiable only when for every ()-focused answer set M of $\mathcal{P}_1 = \{Q_0\}$ we find that $Q_0 \!:\! sat \in M$. Otherwise, we know that there exists some ()-focused answers sets M of $\mathcal{P}_1$ such that $Q_0 \!:\! sat \notin M$. Hence the induction hypothesis is valid for $n = 1$. The induction step is then entirely analogous to what we have proven before, with the only difference being that the cases for n being even or odd are swapped. Finally, since the first quantifier is $\forall$, we need to verify that $Q_0 \!:\! sat$ is true in every $(Q_1, ..., Q_{n-1})$-focused answer set of $\mathcal{P}$. ☐

Proposition 11

Let $\mathcal{P}$ be a communicating normal program with $Q_i \in \mathcal{P}$. The problem of deciding whether there exists a $(Q_1, ..., Q_n)$-focused answer set M of $\mathcal{P}$ such that $Q_i \!:\! l \in M$ (brave reasoning) is in $\Sigma^{\mathrm{P}}_{n+1}$.

Proof. We show the proof by induction on n. In the case where $n = 1$, we need to guess a (Q_1)-focused answer set M of $\mathcal{P}$ which can clearly be done in polynomial time. We now need to verify that this is indeed a (Q_1)-focused answer set which is a problem in coNP. Indeed, verifying that M is not a (Q_1)-focused answer set can be done using the following procedure in NP:

- guess an interpretation M'

- verify that M' is an answer set of $\mathcal{P}$
- verify that $M'_{Q_1} \subset M_{Q_1}$.

Hence, to find a (Q_1)-focused answer set, we guess an interpretation, verify that it is an answer set in polynomial time, and we subsequently use an NP oracle to decide whether this answer set is (Q_1)-focused, i.e. the problem is in Σ_2^{P}. Assume that there exists an algorithm to compute the $(Q_1, ..., Q_{n-1})$-focused answer sets of $\mathcal{P}$ that is in Σ_n^{P}. In a similar fashion, we can guess a $(Q_1, ..., Q_n)$-focused answer set and verify there is no $(Q_1, ..., Q_n)$-focused answer set M' of $\mathcal{P}$ such that $M'_{Q_n} \subset M_{Q_n}$ using a Σ_n^{P} oracle, i.e. the algorithm is in $\Sigma_{n+1}^{\mathsf{P}}$. □

Proposition 12

Let $\mathcal{P}$ be a communicating simple program with $Q_i \in \mathcal{P}$. The problem of deciding whether there exists a $(Q_1, ..., Q_n)$-focused answer set M of $\mathcal{P}$ such that $Q_i : l \in M$ (brave reasoning) is in $\Sigma_{n+1}^{\mathsf{P}}$.

Proof. We know from Proposition 7 that one normal program can be simulated by a communicating simple program with two component programs. Since only the program Q_0 in the simulation in Definition 13 includes negation-as-failure, it suffices to add a single simple component program in order to simulate the negation-as-failure. Since the number of component programs is of no importance in Proposition 11, the result readily follows. □

Proposition 13

Let $\mathcal{P}$ be a communicating disjunctive program with $Q_i \in \mathcal{P}$. The problem of deciding whether $Q_i : l \in M$ with M a $(Q_1, ..., Q_n)$-focused answer set of $\mathcal{P}$ is in $\Sigma_{n+2}^{\mathsf{P}}$.

Proof. This result can easily be verified by looking at the proof of Proposition 11 and noticing that the only part of the algorithm that is affected by the use of communicating disjunctive programs $\mathcal{P}$ is the base step. In this base step, we use an oracle in NP to check whether our guess M is indeed an answer set of $\mathcal{P}$. Since M is an answer set of $\mathcal{P}$ iff $\forall i \in \{1, ..., n\} \cdot (Q_i^M)^\star = M_{Q_i}$ and since Q_i is a disjunctive component program we know that we will instead need an oracle in Σ_2^{P} to deal with communicative disjunctive programs. The remainder of the algorithm sketch remains unaffected. □

Proofs of Chapter 4

Lemma 3

Let L be a set of literals, $M \subseteq L$ a consistent set of literals and let the possibility distribution π be defined as $\pi(\omega) = 1$ if $\omega \models M$ and $\pi(\omega) = 0$ otherwise. Then $M = \{l \mid N(l) = 1, l \in L\}$.

Proof. It is easy to see that for every $l \in M$ we have $N(l) = 1$. Indeed, assume that $l \in M$. Then $\omega \models \neg l$ (which is equivalent to $\omega \not\models l$) implies $\omega \not\models M$, and, by the definition of π, that $\pi(\omega) = 0$. We thus obtain

$$N(l) = 1 - \Pi(\neg l) = 1 - \max\{\pi(\omega) \mid \omega \models \neg l\} = 1.$$

Furthermore, for every $l \notin M$ we must have $N(l) = 0$. Indeed, assume that $l \notin M$, then $M \cup \{\neg l\}$ is consistent. Thus there exists a world ω_0 such that $\omega_0 \models (M \cup \{\neg l\})$, i.e. $\omega_0 \models M$ and $\omega_0 \models \neg l$, or, by the definition of π, $\pi(\omega_0) = 1$ and $\omega_0 \models \neg l$. We thus obtain

$$\begin{aligned} N(l) = 1 - \Pi(\neg l) &= 1 - \max\{\pi(\omega) \mid \omega \models \neg l\} \\ &\leq 1 - \pi(\omega_0) = 0 \end{aligned}$$

Since, by construction of π, $N(l)$ is either 0 or 1, this concludes the proof. □

Proposition 14

Let P be a simple program. If $\pi \in S_P$ then either the unique consistent answer set of P is given by $M = \{l \mid N(l) = 1, l \in Lit_P\}$ or π is the vacuous distribution, in which case P does not have any consistent answer sets.

Proof. We can write the simple program P as $P = P' \cup C$ with C the set of constraint rules and P' the set of all the remaining rules. Since $\pi \in S_P$, we also know that for every rule $r \in P$ we have that π satisfies the constraint $\gamma(r)$.
We now consider the two cases stated in the proposition:

- $\exists \omega \in \Omega \cdot \pi(\omega) > 0$ (i.e. π is not the vacuous distribution)
  This implies that there is no constraint rule $r \in C$ that is violated by M. Indeed, we cannot have for a constraint rule $r = (\leftarrow l_1, ..., l_m)$ that $N(l_1) = ... = N(l_m) = 1$ since $\gamma(r)$ would then imply that $N(\perp) = 1$, i.e. $\forall \omega \in \Omega \cdot \pi(\omega) = 0$. Hence we find

that $\{l_1, ..., l_m\} \not\subseteq M$ due to the construction of M, i.e. r is not violated by M. As such, we know for the remainder of this part of the proof that we only need to take the rules in P' into account, as the rules in C are not applicable.

We now verify that M is a model of P. First recall that every rule $r \in P$ of the form $r = (l_0 \leftarrow l_1, ..., l_m)$ imposes the constraint $\gamma(r) = (N(l_0) \geq \min(N(l_1), ..., N(l_m)))$. Thus, whenever $N(l_1) = ... = N(l_m) = 1$ we know that $\gamma(r)$ enforces that $N(l_0) = 1$. Since π satisfies $\gamma(r)$ and due to the construction of M we thus have that $l_0 \in M$ whenever $\{l_1, ..., l_m\} \subseteq M$, i.e. M is a model of P.

In addition, we can show that M is a consistent model. If M were not a consistent model, then there would be a literal l such that both $l \in M$ and $\neg l \in M$. Thus, by construction of M, we would have that $N(l) = 1$ and $N(\neg l) = 1$. In this case we have $\min(N(l), N(\neg l)) = 1$ and, because of the min-decomposability of N w.r.t. conjunction, $N(l \wedge \neg l) = 1$. This would imply that $N(\bot) = 1$ i.e. we would have that $\forall \omega \in \Omega \cdot \pi(\omega) = 0$. Since we assumed that this is not the case, we must have that M is a consistent model.

We can now verify that M is a minimal model. To see this, assume that M is a consistent model, but not a minimal model of P. Since M is not a minimal model, we know that there exists another consistent model M' of P such that $M' \subset M$. Let us take π' such that $\pi'(\omega) = 1$ if $\omega \models M'$ and $\pi'(\omega) = 0$ otherwise. From Lemma 3 we obtain that $M' = \{l \mid N'(l) = 1, l \in Lit_P\}$ with N' the necessity measure induced by π'. We then have that $\pi' \in C_P$. Indeed, by assumption M' is a model and thus for every rule $r \in P$ with $r = (l_0 \leftarrow l_1, ..., l_m)$ we have $l_0 \in M'$ whenever $\{l_1, ..., l_m\} \subseteq M'$. Due to the relationship between M' and N' and since for every literal l we know that $N'(l) \in \{0, 1\}$ by construction, we have that π' satisfies the constraint $\gamma'(r) = N'(l_0) \geq \min(N'(l_1), ..., N'(l_m))$ imposed by every rule $r \in P$ and hence $\pi' \in C_P$.

We now show that $M' \subset M$ leads to a contradiction. Since $M' \subset M$ we know that there is some literal $l \in M \setminus M'$ with $N'(l) < N(l)$ due to the definitions of M and M'. Thus $\exists \omega \in \Omega \cdot \pi'(\omega) > \pi(\omega)$. Furthermore we have that $\forall \omega \in \Omega \cdot \pi'(\omega) \geq \pi(\omega)$ by construction, unless it were the case that $\pi'(\omega) = 0$ while $\pi(\omega) > 0$ for some ω. Given the construction of M, we know that whenever $\omega \not\models M$, i.e. whenever for some $l \in M$ we have $\omega \not\models l$ (or equivalently $\omega \models \neg l$), that $\pi(\omega) = 0$ since $l \in M$ implies that $\max\{\pi(\omega) \mid \omega \models \neg l\} = 0$. Hence whenever $\pi(\omega) > 0$, we know that $\omega \models M$. Thus we find $\omega \not\models M'$ by construction and $\omega \models M$, which cannot be the case since $M' \subset M$. Hence we conclude that we must have $\exists \omega \in \Omega \cdot \pi'(\omega) > \pi(\omega)$ and $\forall \omega \in \Omega \cdot \pi'(\omega) \geq \pi(\omega)$, i.e. we find that $\pi \notin S_P$. This is a contradiction; M must therefore be a minimal model of P.

- $\forall \omega \in \Omega \cdot \pi(\omega) = 0$
  We consider the two possible cases in which a program $P = P' \cup C$ has the vacuous distribution as a minimally specific possibilistic model. In that case, we may have $\pi \in S_{P'}$, i.e. even without considering the constraints, the vacuous distribution is a minimally specific model of the rules in P'. Otherwise we have $\pi \in C_P \setminus C_{P'}$, i.e. we need to consider the constraint rules in C to obtain the vacuous distribution.
  Assume that $\pi \in S_{P'}$. We have that $M = Lit_{P'}$ is trivially a model of P'. We can furthermore prove that M is a minimal model. Indeed, let $M' \subset M$. Note that due to the definition of a model in ASP, we must either have that M' is a consistent model or that $M' = Lit_P$, which cannot be the case since $M' \subset Lit_P$. Let π' be defined as in the first part of this proof. We can then apply the same line of reasoning as in the first part of this proof where we show that M is indeed a minimal model and hence an answer set of P'.
  Clearly, since M is an inconsistent answer set of P', then due to the semantics of constraint rules we know that either P is an inconsistent program (if M violates no constraint rules in C) and thus has Lit_P as the unique inconsistent answer set or P has no answer sets (if M violates some constraint rule in C).
  Now assume that $\pi \in C_P \setminus C_{P'}$. We then know that there exists some π' with $\pi' > \pi$ such that $\pi' \in S_{P'}$. As in the first part of this proof we obtain that $M' = \{l \mid N'(l) = 1, l \in Lit_{P'}\}$, with N' the necessity measure induced by π', is the unique answer set of P'. Because of the semantics of constraint rules we furthermore know that either M' is the answer set of P or that P has no answer set (i.e. M' violates some constraint rule in C).
  We know by assumption that we obtain π, the vacuous distribution, from π' by considering the constraints associated with the constraint rules in C. We furthermore know that for a rule $r = (\leftarrow l_1, ..., l_m)$ with $r \in C$ we must have that $\min(N'(l_1), ..., N'(l_m)) > 0$, as otherwise π' would be a model of C, i.e. we would be in the first case of this proof. Thus, it readily follows that M' is not an answer set of P because M' violates some constraint $r \in C$, i.e. P has no answer sets.

□

Proposition 15

Let P be a simple program. If M is an answer set of P then the possibility distribution π defined by $\pi(\omega) = 1$ iff $\omega \models M$ and $\pi(\omega) = 0$ otherwise belongs to S_P.

Proof. If M is an answer set of P, then M is by definition a model of P. If M is consistent, then for every rule $r = (l_0 \leftarrow l_1, ..., l_m)$ with $r \in P$ we know that $l_0 \in M$

whenever $\{l_1, ..., l_m\} \subseteq M$. Otherwise, we know that $M = Lit_P$. Furthermore, due to Lemma 3 we know that defining $\pi(\omega) = 1$ if $\omega \models M$ and $\pi(\omega) = 0$ gives us $M = \{l \mid N(l) = 1, l \in Lit_P\}$. It is then easy to see that π satisfies every constraint in C_P and thus that $\pi \models C_P$.

We now show that π is a minimally specific possibilistic model. To prove this, assume that this is not the case, i.e. $\pi \notin S_P$. This implies that there exists some other possibilistic model π' such that $\pi' > \pi$ and in particular that there is some world ω such that $\pi'(\omega) > \pi(\omega) = 0$. By definition, $\pi(\omega) = 0$ if $\omega \not\models M$, i.e. $\pi(\omega) = 0$ if for some literal $l \in M$ we have that $\omega \not\models l$ or, equivalently, $\omega \models \neg l$. Then since $\pi'(\omega) > 0$, we find that $N'(l) < 1$ whereas $N(l) = 1$ due to the construction of π. Now let $M' = \{l \mid N'(l) = 1, l \in Lit_P\}$. It is easy to see that M' is a model of P since π' is by assumption a possibilistic model, i.e. π' satisfies the constraints $N'(l_0) \geq \min(N'(l_1), ..., N'(l_m))$ imposed by the rules $(l_0 \leftarrow l_1, ..., l_m) \in P$ and thus $l_0 \in M'$ whenever $\{l_1, ..., l_m\} \subseteq M'$ due to the construction of M'. However, since $\pi' > \pi$ we know that $M' \subseteq M$ and due to $N'(l) < 1$ and $N(l) = 1$ for some $l \in M$ we know that $M' \subset M$. Thus we find that M is not a minimal model and therefore that M is not an answer set, which is a contradiction. □

Proposition 18

Let P be a disjunctive program, V a valuation and let $\pi \in S^{\mathrm{s}}_{(P,V)}$ be such that

$$\forall l \in Lit_P \cdot V(l) = N(l) \text{ ; and} \tag{4.8}$$

$$\forall l \in Lit_P \cdot N(l) \in \{0, 1\} \tag{4.9}$$

then $M = \{l \mid N(l) = 1, l \in Lit_P\}$ is an answer set of the disjunctive program P.

Proof. We need to prove, when M is consistent, that M is a minimal model of the positive disjunctive program P^M. Consider a rule $r \in P$ with r a rule of the form $(l_0; ...; l_k \leftarrow l_{k+1}, ..., l_m, not\ l_{m+1}, ..., not\ l_n)$. We know from Definition 18 that $\pi \in S^{\mathrm{s}}_{(P,V)}$ satisfies the constraint

$$\max(N(l_0), ..., N(l_k)) \geq \min(N(l_{k+1}), ..., N(l_m), 1 - V(l_{m+1}), ..., 1 - V(l_n)). \tag{6.19}$$

Note that due to (4.8) and (4.9) we know that $V(l_{m+1}), ..., V(l_n)$ all belong to $\{0, 1\}$. Moreover, because we use the minimum, as soon as $V(l_i) = 1$ and thus $(1 - V(l_i)) = 0$ for some $i \in \{m + 1, ..., n\}$, the constraint (6.19) becomes trivial. Indeed, the constraint becomes $\max(N(l_0), ..., N(l_k)) \geq 0$. Correspondingly we know that in that case we have $V(l_i) = 1$ or, equivalently, $N(l_i) = 1$ then that $l_i \in M$ per definition of M. Thus the rule r will be completely omitted from the reduct P^M. Otherwise, when $V(l_{m+1}) = ... = V(l_n) = 0$, the constraint simplifies to

$$\max(N(l_0), ..., N(l_k)) \geq \min(N(l_{k+1}), ..., N(l_m)) \tag{6.20}$$

in which case the reduct P^M will contain the rule $(l_0; ...; l_k \leftarrow l_{k+1}, ..., l_m)$. Also, continuing in the same line of reasoning, we find that $C^{\text{s}}_{P^M} = C^{\text{s}}_{(P,V)}$ and $S^{\text{s}}_{P^M} = S^{\text{s}}_{(P,V)}$.

We can then verify that M is a minimal model of P^M. In the same line of reasoning as in the proof of Proposition 14, we can show that M is a model of P^M. Specifically, we now have that the constraint imposed by a rule ensures that $N(l_i) = 1$ for some l_i with $0 \leq i \leq k$. Assuming that M is a consistent model, we can use the same line of reasoning as in the proof of Proposition 14 to show that M is a minimal model of P^M.

Finally, we need to ensure that Lit_P is the unique answer set of P if P has no consistent answer sets. This implies that M is inconsistent, as otherwise we would be in the case mentioned above. Inconsistencies can only arise since either $N(\bot) = 1$, i.e. because a constraint was violated, or $N(l) = 1$ and $N(\neg l) = 1$ with $l \in Lit_P$, i.e. because the program has two inconsistent conclusions. This is due to the constraints induced by P and due to (4.9), from which we know that these necessity degrees can only be 1 (if they were 0, they would not cause inconsistencies). Furthermore, we have that $l \wedge \neg l \equiv \bot$. As such, we know that for some $l \in Lit_P$ we have that $\Pi(\neg l) = \Pi(l) = 0$. By definition of the possibility measure Π this means that for every $\omega \in \Omega$ we have that $\pi(\omega) = 0$. This can only be the case if $S^{\text{s}}_{(P,V)}$ is a singleton, since for every other possibility distribution $\pi' \neq \pi$ we have that $\pi' > \pi$. We thus find that $S^{\text{s}}_{(P,V)} = \{\pi\}$ and that $\forall l \in Lit_P \cdot N(l) = 1$, i.e. $M = Lit_P$. □

Proposition 19

Let P be a disjunctive program. If M is an answer set of P, there is a valuation V, defined as $V(l) = 1$ if $l \in M$ and $V(l) = 0$ otherwise, and a possibility distribution π, defined as $\pi(\omega) = 1$ if $\omega \models M$ and $\pi(\omega) = 0$ otherwise, such that $\pi \in S^{\text{s}}_{(P,V)}$ and for every $l \in Lit_P$ we have $V(l) = N(l)$.

Proof. When M is consistent, it readily follows from Lemma 3 that $N(l) = 1$ if $l \in M$ and $N(l) = 0$ otherwise. If M is inconsistent, i.e. if $M = Lit_P$, then for every ω we have $\pi(\omega) = 0$ since there does not exist $\omega \models Lit_P$, i.e. for every $l \in Lit_P$ we have $N(l) = 1$. Hence in both cases we find that $V(l) = N(l)$.

We now show that $\pi \in S^{\text{s}}_{(P,V)}$. We start by showing that $C^{\text{s}}_{P^M} = C^{\text{s}}_{(P,V)}$. Since M is an answer set of the disjunctive program P, it is also a minimal model of the reduct

P^M. This reduct is obtained by considering the rules $r \in P$ which are of the form $(l_0; ...; l_k \leftarrow l_{k+1}, ..., l_m, not\ l_{m+1}, ..., not\ l_n)$. The corresponding constraint $\gamma_V(r)$ in $C^{\mathrm{s}}_{(P,V)}$ is then

$$\max(N(l_0), ..., N(l_k)) \geq \min(N(l_1), ..., N(l_m), 1 - V(l_{m+1}), ..., 1 - V(l_n)).$$

For every rule $r \in P$ we have $r' \in P^M$ with $r' = (l_1; ...; l_k \leftarrow l_{k+1}, ..., l_m)$ whenever $\{l_{m+1}, ..., l_n\} \cap M = \emptyset$. Notice that this implies that $V(l_{m+1}) = ... = V(l_n) = 0$ due to how we defined V and thus we obtain

$$\max(N(l_0), ..., N(l_k)) \geq \min(N(l_{k+1}), ..., N(l_m))$$

which corresponds exactly to the constraint induced by $r' \in P^M$. Otherwise, whenever $\{l_{m+1}, ..., l_n\} \cap M \neq \emptyset$ we have that r has no counterpart in P^M. Similarly, the constraint then reduces to the form $\max(N(l_0), ..., N(l_k)) \geq 0$, which is trivially true. We thus find that $C_{P^M} = C_{(P,V)}$ and $S_{P^M} = S_{(P,V)}$.

It now readily follows that π is a possibilistic model of P^M. Indeed, for every constraint

$$\max(N(l_0), ..., N(l_k)) \geq \min(N(l_{k+1}), ..., N(l_m))$$

imposed by a rule $(l_0; ...; l_k \leftarrow l_{k+1}, ..., l_m) \in P^M$ we have that there exists some $l_i \in M$ with $0 \leq i \leq k$ whenever $\{l_{k+1}, ..., l_m\} \subseteq M$. Otherwise M would not be a model of P^M and then M would certainly not be an answer set. Due to the construction of π, where $N(l) = 1$ whenever $l \in M$, it readily follows that π satisfies every constraint in $C^{\mathrm{s}}_{P^M}$, i.e. π is a possibilistic model of P^M. Similar as in the proof of Proposition 15, we can then verify that π is indeed a minimally specific possibilistic model. □

Proposition 20

Let P be a possibilistic positive clausal program without possibilistic constraint rules. Then $P^\star_{\mathrm{w}}$ is a possibilistic answer set of P.

Proof. We need to prove that $P^\star_{\mathrm{w}}$ is a possibilistic answer set of P, or, equivalently, that the minimally specific possibility distribution π encoded by the set of constraints $\left\{N(e) \geq \lambda \mid e^\lambda \in P^\star_{\mathrm{w}}\right\}$ is a possibilistic answer set of P.

To see this, recall that for every possibilistic rule $p \in P$ of the form $p = (r, \lambda)$ with $r = (e_0 \leftarrow e_1, ..., e_m)$ we know from Definition 22 that we have the corresponding constraint $N(e_0) \geq \lambda'$ in C^{w}_P with $\lambda' = \min(N(e_1), ..., N(e_m), \lambda)$. We obtained $P^\star_{\mathrm{w}}$ by repeatedly applying the operator T^{w}_P as defined in Definition 25, starting from the empty set $E = \emptyset$, until the fixpoint is reached. As such, we know that when ${e_0}^{\lambda''} \in P^\star_{\mathrm{w}}$ then $\exists r = (e_0 \leftarrow e_1, ..., e_m) \in P_{\lambda''}$ such that $\forall i \in 1, ..., m \cdot (P^\star_{\mathrm{w}})^{\lambda''} \models e_i$.

We then have that $\lambda'' = \lambda'$. Indeed, assume that $\lambda'' < \lambda'$. This would imply that either $r \notin P_{\lambda'}$ or that $\exists i \in 1, ..., m \cdot (P^\star_{\mathrm{w}})^{\lambda'} \not\models e_i$. The first cannot be the case since this would imply that $p = (r, \lambda) \in P$ with $\lambda < \lambda'$, i.e. $\lambda' \neq \min(N(e_1), ..., N(e_m), \lambda)$. Similarly, the latter cannot be the case since then $\min(N(e_1), ..., N(e_m)) < \lambda'$. A similar line of reasoning can be used to verify that we do not have that $\lambda'' > \lambda'$ as this would imply that $P^\star_{\mathrm{w}}$ is not a least fixpoint.

Since we have that $\lambda' = \lambda''$ this implies that the constraints imposed by $P^\star_{\mathrm{w}}$ and C^{w}_P are the same. Hence also their minimally specific possibility distributions are the same. Thus we find that $\pi \in S^{\mathrm{w}}_P$. $\square$

Proposition 21

A set of weighted clauses E is a possibilistic answer set of the possibilistic clausal program P without possibilistic constraint rules iff E is a possibilistic answer set of P^E.

Proof. $(\Rightarrow)$ Let the set of weighted clauses E be a possibilistic answer set of P and π the corresponding possibility distribution such that $E = \left\{e^{N(e)} \mid e \in \mathit{Clause}_P\right\}$. Furthermore, we choose an E' such that $E'(e) = N(e)$. From Definition 23 we know that $\pi \in S^{\mathrm{w}}_{(P,\pi_{E'})}$ since E is a possibilistic answer set of P. Let us now consider any possibilistic rule $p \in P$ with $p = (r, \lambda)$, $r = (e_0 \leftarrow e_1, ..., e_m, \mathit{not}\ e_{m+1}, ..., \mathit{not}\ e_n)$ and $\lambda \in]0, 1]$. We know from Definition 22 that π satisfies the constraint

$$N(e_0) \geq \min(N(e_1), ..., N(e_m), 1 - N_{E'}(e_{m+1}), ..., 1 - N_{E'}(e_n), \lambda). \tag{6.21}$$

This constraint either reduces to the trivial constraint $N(e_0) \geq 0$ (whenever $N_{E'}(e_{m+1}) = 1$ or ... or $N_{E'}(e_n) = 1$) or it simplifies to

$$N(e_0) \geq \min(N(e_1), ..., N(e_m), \lambda') \tag{6.22}$$

with $\lambda' = \min(1 - E'(e_{m+1}), ..., 1 - E'(e_n), \lambda)$. Note that we can also write λ' as $\lambda' = \min(\lambda_{body}, \lambda_{rule})$ with $\lambda = \lambda_{rule}$ and $\lambda_{body} = 1 - \max(N_{E'}(e_{m+1}), ..., N_{E'}(e_n))$. Hence we find from Definition 26 that the reduct P^E will contain the rule $((e_0 \leftarrow e_1, ..., e_m), \lambda')$. In the same line of reasoning we find that $C^{\mathrm{w}}_{P^E} = C^{\mathrm{w}}_{(P,\pi_{E'})}$ and $S^{\mathrm{w}}_{P^E} = S^{\mathrm{w}}_{(P,\pi_{E'})}$. Hence we find that E is also a possibilistic answer set of P^E since $\pi \in S^{\mathrm{w}}_{P^E}$.

$(\Leftarrow)$ Let the set of weighted clauses E be a possibilistic answer set of P^E and π the corresponding possibility distribution such that $E = \left\{e^{N(e)} \mid e \in \mathit{Clause}_P\right\}$. Furthermore, we choose an E' such that $E'(e) = N(e)$. We now follow a similar line of reasoning as before. Indeed, we know from Definition 22 that π satisfies the constraints (6.22) induced by the rules in P^E. By definition of the reduct from Definition 26 we know that the rules in P^E are obtained from a corresponding program P consisting of rules of the form $p = (r, \lambda)$ with $r = (e_0 \leftarrow e_1, ..., e_m, not\ e_{m+1}, ..., not\ e_n)$, each of which induces the constraint (6.21). Specifically, due to the definition of the reduct operator, we know that the rule $p' = (r', \lambda') \in P^E$ corresponds with a rule $p = (r, \lambda) \in P$ such that $\lambda' = \min(1 - N_{E'}(e_{m+1}), ..., 1 - N_{E'}(e_n), \lambda)$ and for which we know that $N_{E'}(e_{m+1}) \neq 1$ and ... and $N_{E'}(e_n) \neq 1$. For all other rules in P that do not correspond with a rule in P^E we know from Definition 26 that $N(e_0) \geq 0$, i.e. the rule encodes trivial information and can be ignored. As such we again find that $C^{\text{w}}_{P^E} = C^{\text{w}}_{(P, \pi_{E'})}$ and $S^{\text{w}}_{P^E} = S^{\text{w}}_{(P, \pi_{E'})}$. Hence we find that E is also a possibilistic answer set of P since $\pi \in S^{\text{w}}_P$. $\square$

Proposition 22: possibilistic normal program; brave reasoning

Let P be a possibilistic normal program. The problem of deciding whether there exists a possibilistic answer set V of P such that $V(l) \geq \lambda$ is NP-complete.

Proof. (membership) Notice that the reduct defined in Definition 26 can also be applied to possibilistic normal programs. Indeed, possibilistic normal programs are a special cases of possibilistic clausal programs where every clause consists of exactly one literal. Since possibilistic normal programs are a special case, it readily follows from previous proofs that we can use this syntactic method to find possibilistic answer sets of possibilistic normal programs. Furthermore, when considering a possibilistic normal program, we can simplify the reduct. We can write $\forall i \in \{m+1, ..., n\} \cdot e_i \notin V^{\underline{1-\lambda}}$ instead of $\forall i \in \{m+1, ..., n\} \cdot V^{\underline{1-\lambda}} \not\models e_i$, i.e. the reduct P^V with P a possibilistic normal program can be determined in polynomial time.

To determine whether $V(l) \geq \lambda$ with V a possibilistic answer set we need to guess a valuation V such that $V(l) \geq \lambda$. Given such a non-deterministic guess, we can determine the reduct P^V in polynomial time. We can then verify in polynomial time using the immediate consequence operator T_{P^V} from Definition 25 whether V is indeed a possibilistic answer set of P^V and thus a possibilistic answer set of P. Indeed, since we are dealing with literals we can simplify $V^\lambda \models e_i$ to $e_i \in V^\lambda$ to make this operator polynomial, similar as how we did for the reduct from Definition 26. Hence determining whether $V(l) \geq \lambda$ with V a possibilistic answer set is an NP problem.

(hardness) We reduce the problem of determining the satisfiability of a QBF of the form $\phi = \exists X \cdot p(X)$ with $p(X)$ in disjunctive normal form (DNF), i.e. of the form $\theta_1 \vee ... \vee \theta_n$

with each θ_i a conjunction of literals, to the problem of deciding whether there exists a possibilistic answer set V such that $V(l) \geq \lambda$. We define the possibilistic normal program P_ϕ corresponding to ϕ as

$$\begin{aligned} P_\phi = &\{\mathbf{1}\!: x \leftarrow not\ \neg x \mid x \in X\} \cup \{\mathbf{1}\!: \neg x \leftarrow not\ x \mid x \in X\} && (6.23)\\ &\cup \{\mathbf{1}\!: sat \leftarrow \theta_t \mid 1 \leq t \leq n\} && (6.24)\end{aligned}$$

where we identify the conjunction of literals θ_t in (6.24) with a set of literals. It readily follows that the QBF is satisfiable if and only if $V(sat) = 1$. Indeed, the rules in (6.23) generate as many possibilistic answer sets as there are interpretations of X. The rules from (6.24) ensure that 'sat' is contained in the possibilistic answer set whenever for a chosen interpretation of X it holds that $p(X)$ is satisfiable. It readily follows from the construction of P_ϕ that $N(sat) = 1$ iff ϕ is satisfiable. Hence we have reduced the boolean satisfiability problem to the problem of determining whether there exists a possibilistic answer set V such that $V(sat) = 1$. □

Proposition 23: possibilistic normal program; cautious reasoning

Let P be a possibilistic normal program. The problem of deciding whether for all possibilistic answer sets V of P we have that $V(l) \geq \lambda$ is coNP-complete.

Proof. (membership) We show that the complementary problem is in NP. To determine whether there exists a possibilistic answer set V with $V(l) < \lambda$, we guess such a valuation V. Given this non-deterministic guess, we can determine the reduct P^V in polynomial time (where we take P^V as discussed in Proposition 22). We can then verify in polynomial time using the immediate consequence operator T_{P^V} from Definition 25 (simplified as in the proof of Proposition 22) whether V is indeed a possibilistic answer set of P^V and thus a possibilistic answer set of P. Hence determining whether there exists a possibilistic answer set V such that $V(l) < \lambda$ is a problem in NP. Deciding whether for all possibilistic answer sets V we have that $V(l) \geq \lambda$ is thus in coNP.

(hardness) Analogous to the hardness proof in Proposition 22 where we now solve a QBF of the form $\phi = \forall X \cdot p(X)$ and where we are interested in whether for all possibilistic answer sets V we have that $V(sat) = 1$. In particular, the possibilistic answer sets of P are exactly the models of the proposed boolean satisfiability problem (SAT) problem (or QBF). Hence the problem described in this proposition corresponds with the problem of entailment checking. □

Proposition 24: possibilistic disjunctive program; brave reasoning

Let P be a possibilistic disjunctive program. The problem of deciding whether there is a possibilistic answer set V such that $V(l) \geq \lambda$ is a Σ_2^{P}-complete problem.

Proof. (membership) We discussed in the proof of Proposition 22 how the reduct defined in Definition 26 can also be applied to possibilistic normal programs. In addition, notice that the reduct only affects the body of the rule. As such, we can also apply the reduct (where we consider literals instead of clauses, which ensures that the reduct can be determined in polynomial time) to a possibilistic disjunctive program to obtain a possibilistic positive disjunctive program. Furthermore, because the reduct only affects the body, it is easy to see that this syntactic method is also correct for possibilistic disjunctive programs (i.e. the possibilistic answer sets obtained through the syntactic method corresponds perfectly with the semantical definition from Definition 19). Indeed, in the proof of Proposition 21 we then have that (6.21) becomes $\max(N(l_0), ..., N(l_k)) \geq \min(N(l_{k+1}), ..., N(l_m), 1 - V(l_{m+1}), ..., 1 - V(l_n), \lambda)$ whereas (6.22) becomes the constraint $\max(N(l_0), ..., N(l_k)) \geq \min(N(e_{k+1}), ..., N(e_m), \lambda)$. As such, we can verify that indeed $S^{\mathrm{s}}_{P^V} = S^{\mathrm{s}}_{(P,V)}$. We can thus prove correctness of this syntactic approach following a similar line of reasoning as in Proposition 21.

To determine whether there is a possibilistic answer set V such that $V(l) \geq \lambda$ we need to guess a valuation V such that $V(l) \geq \lambda$. Given such a non-deterministic guess, we can determine the reduct P^V in polynomial time. Since P^V is a possibilistic positive disjunctive program, we know that P^V does not necessarily have a unique possibilistic answer set. We can show that V is not an answer set by guessing a V' such that $V' \subset V$ with V' a model of P^V. Thus, to verify in constant time whether V is a possibilistic answer set of the possibilistic positive disjunctive program P^V, we can rely on an NP-oracle. Thus, determining whether there is a possibilistic answer set V of a possibilistic disjunctive program such that $V(l) \geq \lambda$ is in $\mathsf{NP}^{\mathsf{NP}}$, i.e. in Σ_2^{P}.

(hardness) We reduce the problem of determining the satisfiability of a QBF of the form $\phi = \exists X_1 \forall X_2 \cdot p(X_1, X_2)$ with $p(X_1, X_2)$ in DNF, i.e. of the form $\theta_1 \vee ... \vee \theta_n$ with each θ_i a conjunction of literals, to the problem of deciding whether there exists a possibilistic answer set V of a possibilistic disjunctive program such that $V(l) \geq \lambda$. We define the possibilistic disjunctive program P_ϕ corresponding to ϕ as

$$\begin{aligned} P_\phi = &\{\mathbf{1} : x; x' \leftarrow \mid x \in (X_1 \cup X_2)\} && (6.25)\\ &\cup \{\mathbf{1} : sat \leftarrow \theta'_t \mid 1 \leq t \leq n\} && (6.26)\\ &\cup \{\mathbf{1} : x \leftarrow sat \mid x \in X_2\} \cup \{\mathbf{1} : x' \leftarrow sat \mid x \in X_2\} && (6.27) \end{aligned}$$

where we identify the conjunction of literals θ'_t in (6.26) with a set of literals and where we replace occurrences of negated atoms of the form $\neg x$ with fresh atoms x'.

It readily follows that the QBF is satisfiable if and only if $V(sat) = 1$. Indeed, the rules in (6.25) generate as many possibilistic answer sets as there are interpretations of X_1 and X_2. The rules from (6.26) ensure that 'sat' is contained in the possibilistic answer set whenever, for a chosen interpretation of X_1 and X_2, it holds that $p(X_1, X_2)$ is satisfiable. It readily follows from the construction of P_ϕ that $N(sat) = 1$. The rules in (6.27) then employ saturation to ensure that it holds for every interpretation of X_2. Indeed, assume that for some interpretation of X_1 and some interpretation of X_2 it holds that $p(X_1, X_2)$ is satisfiable, but not for some other interpretation of X_2. In particular, assume that the first interpretation gives rise to the possibility distribution π_{sat} and the latter to the possibility distribution π_{notsat}. Then, clearly, $N_{\text{notsat}}(sat) = 0$. Furthermore, since an interpretation of X_2 is a strict subset of X_2 due to (6.25), we know that due to the rules in (6.27) that $\forall \omega \cdot \pi_{\text{notsat}}(\omega) \leq \pi_{\text{sat}}(\omega)$ and in particular that $\exists \omega \cdot \pi_{\text{notsat}}(\omega) < \pi_{\text{sat}}(\omega)$, i.e. π_{sat} is not a least specific possibility distribution and would not give rise to a possibilistic answer set. Thus, if $p(X_1, X_2)$ did not hold for every interpretation of X_2 we would not have that $N(sat) = 1$. □

Proposition 25: possibilistic disjunctive program; cautious reasoning

Let P be a possibilistic disjunctive program. The problem of deciding whether for all possibilistic answer sets V we have that $V(l) \geq \lambda$ is a Π^{P}_2-complete problem.

Proof. (membership) Take $P' = P \cup \{\mathbf{1} : l' \leftarrow not\ l\}$. An answer set V' of P' exists with $V'(l') > 1 - \lambda$ iff we do not have for all answer sets V of P that $V(l) \geq \lambda$. As such, this is the complementary problem of Proposition 24 and thus in in Σ^{P}_2.
(hardness) We reduce the problem of determining the satisfiability of a QBF of the form $\phi = \forall X_1 \exists X_2 \cdot p(X_1, X_2)$ with $p(X_1, X_2)$ in conjunctive normal form (CNF), i.e. of the form $\theta_1 \wedge ... \wedge \theta_n$ with each θ_i a disjunction of literals, to the problem of deciding whether for all possibilistic answer sets V of a possibilistic disjunctive program we have that $V(l) \geq \lambda$. We define the possibilistic disjunctive program P_ϕ corresponding to ϕ as

$$P_\phi = \{\mathbf{1} : x; x' \leftarrow \mid x \in (X_1 \cup X_2)\} \tag{6.28}$$
$$\cup \{\mathbf{1} : unsat \leftarrow \theta'_t \mid 1 \leq t \leq n\} \tag{6.29}$$
$$\cup \{\mathbf{1} : x \leftarrow unsat \mid x \in X_2\} \cup \{\mathbf{1} : x' \leftarrow unsat \mid x \in X_2\} \tag{6.30}$$
$$\cup \{\mathbf{1} : sat \leftarrow not\ unsat\} \tag{6.31}$$

where θ'_t in (6.29) is obtained by taking the negation of θ_t, i.e. if θ_t is the disjunction $x_{11} \vee x_{21} \vee \neg x_{12}$ then θ'_t is the conjunction $x'_{11} \wedge x'_{21} \wedge x_{12}$. Furthermore, we identify the resulting conjunction of literals with a set of literals and we replace occurrences of negated atoms of the form $\neg x$ with fresh atoms x'.

It readily follows that the QBF is satisfiable if and only if $V(sat) = 1$. Indeed, the rules in (6.28) generate as many possibilistic answer sets as there are interpretations of X_1 and X_2. The rules from (6.29) ensure that '$unsat$' is contained in the possibilistic answer set whenever, for a chosen interpretation of X_1 and X_2, it holds that $p(X_1, X_2)$ is unsatisfiable. It readily follows from the construction of P_ϕ that $N(unsat) = 1$. The rules in (6.30) then employ saturation to ensure that $p(X_1, X_2)$ is unsatisfiable for every interpretation of X_2. Indeed, assume that for some interpretation of X_1 and some interpretation of X_2 it holds that $p(X_1, X_2)$ is unsatisfiable, but that there is some other interpretation of X_2 such that $p(X_1, X_2)$ is satisfiable. In particular, assume that the first interpretation gives rise to the possibility distribution π_{unsat} and the latter to the possibility distribution π_{sat}. Then, clearly, $N_{\text{sat}}(unsat) = 0$. Furthermore, since an interpretation of X_2 is a strict subset of X_2 due to (6.28), we know that due to the rules in (6.30) that $\exists \omega \cdot \pi_{\text{sat}}(\omega) \leq \pi_{\text{unsat}}(\omega)$ and in particular that $\exists \omega \cdot \pi_{\text{sat}}(\omega) < \pi_{\text{unsat}}(\omega)$, i.e. π_{unsat} is not a least specific possibility distribution and would not give rise to a possibilistic answer set. Finally, due to (6.31) we know that $N(sat) = 1$ whenever $N(unsat) = 0$. Thus, if $p(X_1, X_2)$ was unsatisfiable for all interpretations of X_2 for some interpretation of X_1 we would not have that $N(sat) = 1$. □

Proposition 26: weak disjunction, positive clausal program; brave reasoning

Let P be a positive clausal program. The problem of deciding whether a clause 'e' is entailed by a consistent answer set E of P is BH$_2$-hard.

Proof. It readily follows from Lemma 4 that when E is the answer set of P we have that E is unique. It may however be that P does not have a consistent answer set. For instance, $P = \{(a \leftarrow), (\neg a \leftarrow)\}$ does not have a consistent answer set.

To prove hardness, i.e. to prove that we can solve BH$_2$-complete problems using brave reasoning over positive clausal programs, we show that the sat-unsat problem can be modelled using positive clausal programs. Sat-unsat is the canonical BH$_2$-complete problem and consists of determining for some pair (T, S) of propositional theories in conjunctive normal form (CNF) whether T is satisfiable and S is unsatisfiable. We can see the propositional theory T as a formula of the form

$$\phi = (k_{11} \vee \ldots \vee k_{1i_1}) \wedge (k_{21} \vee \ldots \vee k_{2i_2}) \wedge \ldots \wedge (k_{n1} \vee \ldots \vee k_{ni_n})$$

and the propositional theory S as a formula of the form

$$\psi = (s_{11} \vee \ldots \vee s_{1j_1}) \wedge (s_{21} \vee \ldots \vee s_{2j_2}) \wedge \ldots \wedge (s_{m1} \vee \ldots \vee s_{mj_m})$$

where k_{ij} and s_{ij} are literals such that none of the literals used in ϕ occur in ψ and vice versa. The positive clausal program $P = P_1 \cup P_2$ that can be used to solve the SAT-unSAT problem contains the set of rules P_1:

$$\begin{aligned} k_{11} \vee \ldots \vee k_{1i_1} &\leftarrow \\ k_{21} \vee \ldots \vee k_{2i_2} &\leftarrow \\ &\vdots \\ k_{n1} \vee \ldots \vee k_{ni_n} &\leftarrow \end{aligned}$$

and the set of rules P_2:

$$\begin{aligned} unsat \vee s_{11} \vee \ldots \vee s_{1j_1} &\leftarrow \\ unsat \vee s_{21} \vee \ldots \vee s_{2j_2} &\leftarrow \\ &\vdots \\ unsat \vee s_{m1} \vee \ldots \vee s_{mj_m} &\leftarrow . \end{aligned}$$

The rules in P_1 are used to determine whether T is satisfiable. Indeed, since an answer set $M \neq Clause_P$ of P must, by definition, be consistent, it readily follows that whenever P has a consistent answer set E we must have that the formula ϕ is satisfiable. The rules in P_2 are used to verify whether S is unsatisfiable. Notice that we will only be able to derive '$unsat$' from P if and only if S is unsatisfiable. Indeed, '$unsat$' can only be derived from an answer set E if '$unsat$' is true in every model of program P. If S were satisfiable, we could always take a model of S, which would automatically be a model of program P_2 in which '$unsat$' is false, thus preventing us from deriving '$unsat$'. Hence we find that there exists a consistent answer set E of the positive clausal program P such that $E \models unsat$ iff T is satisfiable and S is unsatisfiable. □

Proposition 27: weak disjunction, positive clausal program; brave reasoning

Let P be a positive clausal program. The problem of deciding whether a clause 'e' is entailed by a consistent answer set E of P is in BH$_2$.

Proof. We translate the problem of deciding whether 'e' is entailed by a consistent answer set M of the positive clausal program P to the problem of consistency and entailment checking in meta-epistemic logic (MEL) [Banerjee and Dubois 2009]. MEL corresponds to a fragment of KD modal logic [Huth and Ryan 2004] where nesting of propositional formulas and modalities is not allowed (i.e. the depth of modal operators is limited to exactly 1).

Let $\mathcal{V} = 2^{\mathcal{B}_P}$ be the set of all interpretations, where each interpretation $w \in \mathcal{V}$ is defined as a mapping $w : \mathcal{B}_P \rightarrow \{0, 1\}$. For a formula ϕ in propositional logic, $w \models \phi$ indicates that w satisfies ϕ, i.e. that w is a model of ϕ. The set of models of a propositional formula ϕ is denoted by $[\phi] = \{w \mid w \models \phi\}$. The epistemic state of an agent $\mathcal{E}$ is represented by $K \subseteq \mathcal{V}$. We have $K \models \Box\phi$ iff $K \subseteq [\phi]$, $K \models \neg\phi$ iff $K \not\models \phi$, $K \models \phi \wedge \psi$ iff $K \models \phi$ and $K \models \psi$ and we have $\vee$ defined in the usual way, i.e. $\phi \vee \psi := \neg(\neg\phi \wedge \neg\psi)$. The models of MEL-formulae are called meta-models to avoid confusion with the models of propositional formulae.

For every $r \in P$, where r is of the form

$$e_0 \leftarrow e_1, ..., e_m \tag{6.32}$$

with e_i clauses for $0 \leq i \leq m$, we add the MEL-formula

$$\neg\Box(e_1 \wedge ... \wedge e_m) \vee \Box e_0. \tag{6.33}$$

to the MEL theory $\mathcal{K}$. What this tells us is that either $e_1 \wedge ... \wedge e_m$ is not entailed by K or, that e_0 is entailed by K. We now show that every meta-model K of the MEL theory $\mathcal{K}$ corresponds with a possibilistic model π of P iff P has consistent models, which is because $N(e_i) = 1$ iff $K \models \Box(e_i)$. Let K be a meta-model of the MEL theory $\mathcal{K}$ and let π be defined as $\pi(\omega) = 1$ if $\omega \in K$ and $\pi(\omega) = 0$ otherwise. For a rule of the form (6.32) it can readily be seen that π is a possibilistic model. Indeed, assume that $N(e_1) = ... = N(e_m) = 1$. This implies that for every world ω such that $\omega \models \neg e_1 \vee ... \vee \neg e_m$ we have that $\pi(\omega) = 0$. Hence, for every ω such that $\pi(\omega) = 1$ we must have that e_0 is satisfied in ω as otherwise the MEL-formula (6.33) is not satisfied, i.e. we must have $N(e_0) = 1$. When we do not have that $N(e_1) = ... = N(e_m) = 1$ then the constraint $N(e_0) \geq \min(N(e_1), ..., N(e_m))$ is vacantly satisfied. It can also easily be seen that every meta-model corresponds with exactly one possibilistic model given the construction. It readily follows that for π a possibilistic model of P we have that $K = \{\omega \mid \pi(\omega) = 1\}$ is also a meta-model of $\mathcal{K}$ using a similar line of reasoning, given that P is a consistent program.

Due to the correspondence between meta-models and possibilistic models, we have that a possibilistic model corresponding with a consistent answer set only exists if the associated MEL-theory $\mathcal{K}$ is satisfiable. From Lemma 4 we also know that the clause e is entailed by the answer set of P iff for every possibilistic model π of P we have that $N(e) = 1$, as otherwise we would not have for $\pi \in S_P^{\text{w}}$ that $N(e) = 1$. Hence we know that e is entailed by an answer set of P if $K \models \Box e$ for every meta-model K of $\mathcal{K}$. A clause 'e' thus belongs to a consistent answer set M of P if $\mathcal{K}$ is satisfiable and for every meta-model K of $\mathcal{K}$ it holds that $K \models \Box e$. As satisfiability and entailment in KD for a modal depth of 1 are in NP and coNP respectively [Nguyen 2005], this concludes the proof. $\square$

Corollary 5: weak disjunction, positive clausal program; answer set existence

Determining whether a positive clausal program P has a consistent answer set is an NP-complete problem.

Proof. The problem of determining whether a positive clausal program P has an answer set reduces to the problem of satisfiability checking in MEL as in the proof of Proposition 27, which is a problem in NP. Furthermore, from the proof of Proposition 26 we know that satisfiability of a propositional theory can be checked by verifying whether a program P has an answer set, hence this problem is also NP-hard. $\square$

Corollary 6: weak disjunction, positive clausal program; cautious reasoning

Cautious reasoning, i.e. determining whether a clause 'e' is entailed by every answer set E of a positive clausal program P is coNP-complete.

Proof. This problem reduces to the problem of entailment checking in MEL as in the proof of Proposition 27, which is a problem in coNP. Furthermore, from the proof of Proposition 26 we know that unsatisfiability of a propositional theory can be checked by verifying whether some clause 'e' (in particular, '$unsat$' in Proposition 26) is entailed by the answer sets of a positive clausal program P, hence this problem is also coNP-hard. $\square$

Proposition 28

Let P be a positive clausal program without classical negation. We can find the unique answer set of P in polynomial time.

Proof. Applying the simplified immediate consequence operator can be done in polynomial time. Furthermore, only a polynomial number of applications of the operator are necessary since it is never possible to derive more information than the union of all the heads of all the rules and because after each application we either obtain at least one new clause or we have found an answer set. $\square$

Proposition 29: weak disjunction; brave reasoning

Let P be a clausal program. The problem of deciding whether a clause 'e' is entailed by a consistent answer set E of P is Σ_2^{P}-hard.

Proof. We reduce the problem of determining the satisfiability of a QBF of the form $\phi = \exists X_1 \forall X_2 \cdot p(X_1, X_2)$ with $p(X_1, X_2)$ in DNF to the problem of deciding whether a clause 'e' is entailed by a consistent answer set E of P. Since the problem of determining satisfiability of QBFs of that form is the canonical Σ_2^{P}-complete problem, this shows that the problem of deciding whether a clause 'e' is entailed by a consistent answer set E of P is indeed Σ_2^{P}-hard. To prove this we use the clausal program P_ϕ that simulates ϕ from Definition 27.

The rules in (4.12) ensure that as many candidate answer sets are generated as there are interpretations of X_1. The rules from (4.13) verify whether, for the chosen interpretation of X_1, it holds that $p(X_1, X_2)$ is satisfied for all interpretations of X_2. To see this, we first draw the attention of the reader to the fact that these rules are exactly the construct used in Proposition 26 to simulate entailment. Hence we are only able to derive 'sat' if the set of propositional clauses $\{\neg\theta_1, ..., \neg\theta_n\}$ is unsatisfiable. Notice furthermore that the set $\{\neg\theta_1, ..., \neg\theta_n\}$ is unsatisfiable iff formula $p(X_1, X_2)$ is false. Finally, the rule (4.14) eliminates every answer set in which 'sat' is not true. Thus the program P_ϕ only has answer sets from which 'sat' can be entailed or it does not have any answer sets at all. Hence the problem of deciding whether a clause 'sat' is entailed by an answer set M of P_ϕ corresponds to determining whether or not a QBF of the form $\phi = \exists X_1 \forall X_2 \cdot p(X_1, X_2)$ is satisfiable. □

Proposition 30: weak disjunction; brave reasoning

Let P be a clausal program. The problem of deciding whether a clause 'e' is entailed by a consistent answer set E of P is in Σ_2^{P}.

Proof. To show that this problem is indeed in Σ_2^{P}, we construct an algorithm to decide whether a clause 'e' is entailed by an answer set E of P:

```
guess a subset $E$ of $\{head(r) \mid r \in P\}$
verify that this interpretation is an answer set as follows:
    calculate the reduct $P^E$ of the clausal program $P$
    calculate the fixpoint $(P^E)^\star_{\mathrm{w}}$
    verify that $E$ is entailed by $(P^E)^\star_{\mathrm{w}}$
```

The first step of the algorithm requires a choice and makes our algorithm non-deterministic. Next, we verify whether this guess is indeed an answer set which involves taking the reduct (which can be done in polynomial time), computing the fixpoint (which depends on entailment, an NP-complete problem) and finally determining whether our guess is entailed by this fixpoint (which we found in Corollary 4 to be a problem that is BH_2-complete). The penultimate and last step thus require the use of an oracle that can solve NP problems in constant time. Hence we find that the problem of deciding whether a clause 'e' is entailed by an answer set E of P is indeed in Σ_2^{P}. □

Corollary 8: weak disjunction; answer set existence

Determining whether a clausal program P has a consistent answer set is an Σ_2^{P}-complete problem.

Proof. Notice that in the proof of Proposition 29 the program P_ϕ only has answer sets from which 'sat' can be entailed or that it does not have any answer sets at all. Hence the problem of determining the satisfiability of a QBF of the form $\phi = \exists X_1 \forall X_2 \cdot p(X_1, X_2)$ with $p(X_1, X_2)$ in DNF can be reduced to the problem of deciding answer set existence for a clausal program P. □

Proposition 31: weak disjunction; cautious reasoning

Cautious reasoning, i.e. determining whether a clause 'e' is entailed by every answer set E of a clausal program P, is Π_2^{P}-complete.

Proof. This problem is complementary to brave reasoning, i.e. we verify that there does not exist an answer set E' of P such that '$\neg e$' is entailed by E'. □

Proofs of Chapter 5

Proposition 32

Let P be a simple PASP program. For each literal 'l' we have that $N\left(P \models^{\mathrm{c}} l\right) \geq \lambda$ iff $M(l) \geq \lambda$ with M the possibilistic answer set of P under the semantics from PASP$_{Ł}$, which in turn coincides with the semantics from PASP$_{\mathsf{G}}$.

Proof. For each literal 'l' we have that $N\left(P \models^{\mathrm{c}} l\right) \geq \lambda$ iff $M(l) \geq \lambda$ with M the possibilistic answer set of P under the semantics from [Nicolas et al. 2006] and as given in Section 2.5. To see this, note that $M(l) \geq \lambda$ iff l is in the unique answer set M_λ of the λ-cut of P. Indeed, we can only conclude $M(l) \geq \lambda$ if we have only used rules with associated weights equal or greater than λ to deduce 'l'. Similarly, $N\left(P \models^{\mathrm{c}} l\right) \geq \lambda$ iff there does not exist some subprogram P' such that $\pi_P(P') > 1 - \lambda$ for which we have that $P' \not\models^{\mathrm{c}} l$. But, the only subprograms for which $\pi_P(P') > 1 - \lambda$ are exactly those subprograms from which we did not remove any rules with associated weights equal or greater than λ. Thus we have that l can be deduced from every subprogram P' with $\pi_P(P') > 1 - \lambda$. In particular, because of the monotonicity of inference of simple programs, it suffices to consider the subprogram that corresponds exactly with the λ-cut of P. Hence $N\left(P \models^{\mathrm{c}} l\right) \geq \lambda$ iff $M(l) \geq \lambda$. As discussed in Section 4.2.2, where we investigated the relationship between PASP$_{\mathsf{G}}$ and PASP$_{Ł}$, the semantics of PASP$_{\mathsf{G}}$ and PASP$_{Ł}$ coincide for possibilistic simple programs. Thus, the result also holds for M the possibilistic answer set of P under PASP$_{Ł}$. □

Proposition 33

Let P be a possibilistic normal program. Deciding whether

$$\Pi\left(P \models^{\mathrm{b}} l\right) \geq \lambda \text{ is NP-complete;}$$
$$N\left(P \models^{\mathrm{c}} l\right) \geq \lambda \text{ is coNP-complete;}$$
$$\Pi\left(P \models^{\mathrm{c}} l\right) \geq \lambda \text{ is } \Sigma_2^{\mathsf{P}}\text{-complete;}$$
$$N\left(P \models^{\mathrm{b}} l\right) \geq \lambda \text{ is } \Pi_2^{\mathsf{P}}\text{-complete.}$$

Proof. Part 1: deciding whether $\Pi\left(P \models^{\mathrm{b}} l\right) \geq \lambda$ *is* NP*-complete.*
(membership) To determine whether $\Pi\left(P \models^{\mathrm{b}} l\right) \geq \lambda$ we need to guess a subset P' of rules from P such that $\pi_P(P') \geq \lambda$ and an interpretation M which includes 'l'. Given such a non-deterministic guess, we can verify in polynomial time whether M is indeed an

answer set and hence whether $P'^{*} \models^{\mathrm{b}} l$. Hence determining whether $\Pi\big(P \models^{\mathrm{b}} l\big) \geq \lambda$ is in NP.

(hardness) NP-hardness follows trivially from the NP-hardness of brave reasoning for classical normal programs. □

Proof. Part 2: deciding whether $N\,(P \models^{\mathrm{c}} l) \geq \lambda$ is coNP*-complete.*
(membership) We will show that the complementary problem is in NP. To determine whether $N\,(P \models^{\mathrm{c}} l) \ngeq \lambda$ we guess a subset P' of rules from P such that $\pi_P(P') > 1 - \lambda$ and a consistent interpretation M, which does not include 'l'. Given such a non-deterministic guess, we can verify in polynomial time that M is an answer set of P'^{*} and hence that $P'^{*} \not\models^{\mathrm{c}} l$. From Definition 29 we know that $N\,(P \models^{\mathrm{c}} l) = 1 - \max\{\pi_P(Q) \mid Q \subseteq P$ and $Q^{*} \not\models^{\mathrm{c}} l\} \leq 1 - \pi_P(P') < \lambda$. In other words: determining whether $N\,(P \models^{\mathrm{c}} l) \ngeq \lambda$ is in NP. Deciding whether $N\,(P \models^{\mathrm{c}} l) \geq \lambda$ is thus in coNP.

(hardness) The coNP-hardness follows trivially from the coNP-hardness of cautious reasoning for classical normal programs. □

Proof. Part 3: deciding whether $\Pi\,(P \models^{\mathrm{c}} l) \geq \lambda$ is Σ_2^{P}-complete.
(membership) To determine whether $\Pi\,(P \models^{\mathrm{c}} l) \geq \lambda$ we need to guess a subset P' of rules from P such that $\pi_P(P') \geq \lambda$. We cannot immediately guess an interpretation to determine whether $P'^{*} \models^{\mathrm{c}} l$ since this requires that 'l' is true in every single interpretation. Given a non-deterministic guess of P', however, we can rely on an NP-oracle [Baral 2003] to verify in constant time whether $P'^{*} \models^{\mathrm{c}} l$, as P'^{*} is a classical normal program. Hence determining whether $\Pi\,(P \models^{\mathrm{c}} l) \geq \lambda$ is in $\mathsf{NP}^{\mathsf{NP}}$, i.e. in Σ_2^{P}.

(hardness) We reduce the problem of determining the satisfiability of a QBF of the form $\phi = \exists X_1 \forall X_2 \cdot p(X_1, X_2)$ with $p(X_1, X_2)$ in DNF, i.e. of the form $\theta_1 \vee \ldots \vee \theta_n$ with each θ_i a conjunction of literals, to the problem of deciding whether $\Pi\,(P \models^{\mathrm{c}} l) \geq \lambda$. We define the possibilistic normal program P_ϕ corresponding to ϕ as

$$\begin{aligned} P_\phi = &\{\mathbf{0.5}\colon x \leftarrow \mid x \in X_1\} \cup \{\mathbf{0.5}\colon \neg x \leftarrow \mid x \in X_1\} && (6.34)\\ &\cup \{\mathbf{1}\colon x \leftarrow \mathit{not}\ \neg x \mid x \in X_2\} \\ &\cup \{\mathbf{1}\colon \neg x \leftarrow \mathit{not}\ x \mid x \in X_2\} && (6.35)\\ &\cup \{\mathbf{1}\colon \mathit{sat} \leftarrow \theta_t \mid 1 \leq t \leq n\} && (6.36)\end{aligned}$$

where we identify the conjunction of literals θ_t in (6.36) with a set of literals. We now show that the QBF is satisfiable if and only if $\Pi\,(P_\phi \models^{\mathrm{c}} \mathit{sat}) \geq 0.5$.

The rules in (6.34) ensure that there are as many subprograms $P' \subseteq P_\phi$ as there are interpretations of X_1. The subprograms P' with $\pi_{P_\phi}(P') > 0$ then contain the rules (6.35) that generate as many answer sets as there are interpretations of X_2. The rules from (6.36) ensure that 'sat' is contained in the classical answer set whenever for a chosen interpretation of X_1 and X_2 it holds that $p(X_1, X_2)$ is satisfied. Notice that the certainty attached to the rules ensures that removing any of the rules from (6.35) or (6.36) results in $\pi_{P_\phi}(P') = 0$, i.e. it indicates that these rules are completely necessary.

We then have that $\Pi\left(P_\phi \models^{\mathrm{c}} sat\right) \geq 0.5$ if and only if the QBF is satisfiable. Indeed, from the construction of P_ϕ, and in particular from the rules (6.34), we know that for every interpretation of X_1 there will be a corresponding consistent subprogram for which the possibility is 0.5. Also, $P'^* \models^{\mathrm{c}} sat$ if and only if P' corresponds to an interpretation of X_1 such that $p(X_1, X_2)$ is consistent for every interpretation of X_2. Using the consistent possibility measure (i.e. finding $\max\left\{\pi_{P_\phi}(P') \mid P' \subseteq P_\phi \text{ and } P'^* \models^{\mathrm{c}} sat\right\}$) implies that the QBF is satisfied whenever we find at least one such an interpretation X_1.

Some of the subprograms of P_ϕ may either be inconsistent subprograms or may correspond to partial interpretations of X_1. However, the inconsistent subprograms P' have $\pi_{P_\phi}(P') = 0$ by definition and can therefore never be used to derive $\Pi\left(P_\phi \models^{\mathrm{c}} sat\right) \geq 0.5$. Furthermore, any subprogram P' with incomplete assignments for the variables in X_1 from which we can conclude that $P'^* \models^{\mathrm{c}} sat$ can trivially be extended to a subprogram P'' to which we add some rules from (6.34) to complete the assignment for the variables in X_1 and we will still be able to conclude that $P''^* \models^{\mathrm{c}} sat$. □

Proof. Part 4: deciding whether $N\left(P \models^{\mathrm{b}} l\right) \geq \lambda$ is Π_2^{P}-complete.
(membership) We will show that the complementary problem is in Σ_2^{P}. To determine whether $N\left(P \models^{\mathrm{b}} l\right) \not\geq \lambda$ we guess a subset P' of rules from P such that $\pi_P(P') > 1 - \lambda$. Given a non-deterministic guess for P', we rely on an NP-oracle [Baral 2003] to verify in constant time that $P'^* \not\models^{\mathrm{b}} l$. Similar as in *Part 2* of this proof this gives us a counter-example for $N\left(P \models^{\mathrm{b}} l\right) \geq \lambda$. Hence determining whether $N\left(P \models^{\mathrm{b}} l\right) \geq \lambda$ is in co$\left(\mathsf{NP}^{\mathsf{NP}}\right)$, i.e. in Π_2^{P}.
(hardness) Let Q be the program defined as $P \cup \{\mathbf{1} : x \leftarrow not\ l\}$ with x a fresh literal. Then $N\left(Q \models^{\mathrm{b}} l\right) \geq \lambda$ if and only if $\Pi\left(Q \models^{\mathrm{c}} x\right) \leq 1 - \lambda$. Indeed, we know from Definition 29 that $N\left(Q \models^{\mathrm{b}} l\right) \geq \lambda$ is true whenever $1 - \max\left\{\pi_Q(P') \mid P' \subseteq Q \text{ and } P'^* \not\models^{\mathrm{b}} l\right\} \geq \lambda$, i.e. whenever we have that $\max\left\{\pi_Q(P') \mid P' \subseteq Q \text{ and } P'^* \not\models^{\mathrm{b}} l\right\} \leq 1 - \lambda$. Because the newly added rule $(x \leftarrow not\ l)$ will be in every subprogram P' with $\pi_Q(P') > 0$ (since the certainty attached to this rule is 1), we know that there is at least one answer set in which l is true if and only if it is not the case that x is true in every answer set. Thus, the previous inequality is equivalent to $\max\left\{\pi_Q(P') \mid P' \subseteq Q \text{ and } P'^* \models^{\mathrm{c}} x\right\} \leq 1 - \lambda$ and, by applying Definition 29, to $\Pi\left(Q \models^{\mathrm{c}} x\right) \leq 1 - \lambda$. Since the set of certainty values associated with the rules is finite, this equation is equivalent to $\Pi\left(Q \models^{\mathrm{c}} x\right) < \lambda'$ for some λ'. Hence we have that $\neg(\Pi\left(Q \models^{\mathrm{c}} x\right) \geq \lambda')$. This problem is therefore the complement of the decision problem from *Part 3* of this proof. Thus deciding whether $N\left(P \models^{\mathrm{b}} l\right) \geq \lambda$ is Π_2^{P}-hard. □

Proposition 34

Let P be a possibilistic disjunctive program. Deciding whether

$$\Pi\left(P \models^{\mathrm{b}} l\right) \geq \lambda \text{ is } \Sigma_2^{\mathsf{P}}\text{-complete;}$$
$$N\left(P \models^{\mathrm{c}} l\right) \geq \lambda \text{ is } \Pi_2^{\mathsf{P}}\text{-complete;}$$
$$\Pi\left(P \models^{\mathrm{c}} l\right) \geq \lambda \text{ is } \Sigma_3^{\mathsf{P}}\text{-complete;}$$
$$N\left(P \models^{\mathrm{b}} l\right) \geq \lambda \text{ is } \Pi_3^{\mathsf{P}}\text{-complete.}$$

Proof. Part 1: deciding whether $\Pi\left(P \models^{\mathrm{b}} l\right) \geq \lambda$ *is* Σ_2^{P}*-complete.*
(membership) Analogous to the proof in *Part 1* of the proof of Proposition 33, where we are able to verify in constant time that M is an answer set of P^*, which now is a positive possibilistic program, by using an NP-oracle.
(hardness) Analogous to the proof of the hardness in *Part 1* of the proof of Proposition 33. □

Proof. Part 2: deciding whether $N\left(P \models^{\mathrm{c}} l\right) \geq \lambda$ *is* Π_2^{P}*-complete.*
Entirely analogous to the proof in *Part 2* of the proof of Proposition 33. □

Proof. Part 3: deciding whether $\Pi\left(P \models^{\mathrm{c}} l\right) \geq \lambda$ *is* Σ_3^{P}*-complete.*
(membership) Analogous to the membership proof in *Part 3* of the proof of Proposition 33, but where we now require a Σ_2^{P}-oracle to verify in constant time whether for P'^*, with P'^* being a classical disjunctive program, we have that $P'^* \models^{\mathrm{c}} l$.
(hardness) Analogous to the hardness proof in *Part 4* of the proof of Proposition 33, where we can now reduce the complement of this problem to an instance of the decision problem from *Part 4* of this proof. □

Proof. Part 4: deciding whether $N\left(P \models^{\mathrm{b}} l\right) \geq \lambda$ *is* Π_3^{P}*-complete.*
(membership) Analogous to the proof of membership proof in *Part 4* of the proof of Proposition 33.
(hardness) We reduce the problem of determining the satisfiability of a QBF of the form $\psi = \forall X_1 \exists X_2 \forall X_3 \cdot p(X_1, X_2, X_3)$ with $p(X_1, X_2, X_3)$ in DNF, i.e. of the form $\theta_1 \vee ... \vee \theta_n$ with each θ_i a conjunction of literals, to the problem of deciding whether $N\left(P \models^{\mathrm{b}} l\right) \geq \lambda$.

We define the possibilistic disjunctive program P_ψ corresponding to ψ as

$$\begin{aligned} P_\psi = &\{\mathbf{1}: x; x' \leftarrow \mid x \in X_3\} && (6.37)\\ &\cup \{\mathbf{0.5}: x \leftarrow \mid x \in X_1\} \cup \{\mathbf{0.5}: \neg x \leftarrow \mid x \in X_1\} && (6.38)\\ &\cup \{\mathbf{1}: x \leftarrow \mathit{not}\ \neg x \mid x \in X_2\} \\ &\cup \{\mathbf{1}: \neg x \leftarrow \mathit{not}\ x \mid x \in X_2\} && (6.39)\\ &\cup \{\mathbf{1}: \mathit{sat} \leftarrow \theta'_t \mid 1 \leq t \leq n\} && (6.40)\\ &\cup \{\mathbf{1}: x \leftarrow \mathit{sat} \mid x \in X_3\} \cup \{\mathbf{1}: x' \leftarrow \mathit{sat} \mid x \in X_3\} && (6.41) \end{aligned}$$

where we identify the conjunction of literals θ'_t in (6.40) with a set of literals and we have furthermore replaced all negative literals of the form $\neg x$ by a fresh atom of the form x' for every $x \in X_3$. We now show that the QBF is satisfiable if and only if $N\left(P_\psi \models^{\mathrm{b}} \mathit{sat}\right) \geq 0.25$.

The rules in (6.38) ensure that there are at least as many subprograms $P' \subseteq P_\psi$ as there are interpretations of X_1. Furthermore, the subprograms P' with $\pi_{P_\psi}(P') > 0$ contain the rules (6.37) and (6.39), which generate as many answer sets as there are interpretations of $(X_2 \cup X_3)$. The rule (6.40) ensures that 'sat' is contained in the classical answer set whenever for a chosen interpretation of X_1, X_2 and X_3 it holds that $p(X_1, X_2, X_3)$ is satisfied. Notice that the certainty attached to the rules ensures that removing any of the rules from (6.37), (6.39), (6.40) or (6.41) results in $\pi_{P_\psi}(P') = 0$, i.e. it indicates that these rules are completely necessary.

Thus far, we have not discussed the rules from (6.41). These rules work together with the rules from (6.37) to resolve the last $\forall$. Indeed, the rules from (6.41) implement a saturation technique [Baral 2003] over a disjunctive program to ensure that sat will only be true in an answer set when $p(X_1, X_2, X_3)$ is satisfied for every interpretation of X_3, given some interpretation of (X_1, X_2). In particular, let P' be a subprogram of P_ψ with $\pi_{P_\psi}(P') > 0$, and let M be an answer set of P' that contains sat. Because of the rules from (6.38) and (6.39) in P', M contains literals corresponding to the variables of X_1 and X_2, and as such defines an interpretation of X_1 and X_2. Furthermore, because of the saturation rules (6.41), M contains the literals x and x' for every $x \in X_3$. Now suppose that there would exist an interpretation M' of P' that contains the same literals as M corresponding to the variables of X_1 and X_2 but that does not contain sat, then we have that $M' \subset M$. Furthermore, since M' contains the same literals as M corresponding to the variables of X_2, $(P')^M = (P')^{M'}$. If M' would be an answer set of P', then by definition it would be a minimal model of $(P')^M$. This, together with $M' \subset M$, would contradict the fact that M is an answer set of P'. Hence M' is not an answer set of P'. We conclude that when sat is contained in an answer set M of P', then sat is contained in all answer sets of P' that contain the same literals corresponding to the variables of X_1 and

X_2 as M, regardless of which choice is made by rules (6.37) for the literals corresponding to the variables of X_3, i.e. regardless of the interpretation of X_3.

We then have that the QBF is satisfied iff $N\left(P_\psi \models^{\mathrm{b}} sat\right) \geq 0.25$. Indeed, from the construction of P_ψ, and in particular from the rules (6.38), we know that for every interpretation of X_1 there will be a corresponding consistent subprogram P' for which the possibility is 0.5. Also, $P'^{*} \models^{\mathrm{b}} sat$ if and only if P' has an answer set such that $p(X_1, X_2, X_3)$ is satisfied for every interpretation of X_3. Using the necessity measure, it then holds that the QBF is satisfied for every interpretation of X_1. Finally, note that we need to consider a necessity strictly smaller than 0.5, e.g. 0.25. Indeed, we have

$$\begin{aligned}
& N(P_\psi \models^{\mathrm{b}} sat) \geq \lambda \\
\equiv\; & 1 - \max\left\{\pi_{P_\psi}(P') \mid P' \subseteq P_\psi \text{ and } P'^{*} \not\models^{\mathrm{b}} sat\right\} \geq \lambda \\
\equiv\; & \max\left\{\pi_{P_\psi}(P') \mid P' \subseteq P_\psi \text{ and } P'^{*} \not\models^{\mathrm{b}} sat\right\} \leq 1 - \lambda \\
\equiv\; & \forall P' \subseteq P_\psi, P'^{*} \not\models^{\mathrm{b}} sat \cdot \pi_{P_\psi}(P') \leq 1 - \lambda \\
\equiv\; & \forall P' \subseteq P_\psi, \pi_{P_\psi}(P') > 1 - \lambda \cdot P'^{*} \models^{\mathrm{b}} sat.
\end{aligned}$$

Furthermore, the possibility associated with each subprogram P' is $\pi_{P_\psi}(P') \in \{0, 0.5, 1\}$. Hence, by verifying $N(P_\psi \models^{\mathrm{b}} sat) \geq \lambda$ with $\lambda = 0.5$, we have only verified that sat is a brave conclusion of those subprograms P' with $\pi_P(P') = 1$. Instead, we want to verify for those subprograms P' with $\pi_P(P') = 0.5$ whether $P' \models^{\mathrm{b}} sat$, i.e. we need to verify whether $N\left(P_\psi \models^{\mathrm{b}} sat\right) \geq \lambda$ for an arbitrary λ in $]0.5, 1]$.

Some of the subprograms of P_ψ may either be inconsistent subprograms or may correspond to partial interpretations of X_1. However, the inconsistent subprograms P' have $\pi_{P_\psi}(P') = 0$ by definition and can therefore never be used to derive $N\left(P_\psi \models^{\mathrm{b}} sat\right) \geq 0.75$ (also, we already established that for every interpretation of X_1 there will be a corresponding consistent subprogram P'). Furthermore, any subprogram P' with incomplete assignments for the variables in X_1 from which we can conclude that $P'^{*} \models^{\mathrm{b}} sat$ can trivially be extended to a subprogram P'' to which we add some rules from (6.38) to complete the assignment for the variables in X_1 and we will still be able to conclude that $P''^{*} \models^{\mathrm{b}} sat$. Thus, these additional subprograms do not affect our ability to derive $N\left(P_\psi \models^{\mathrm{b}} sat\right) \geq 0.25$. □

Proposition 35

Let P be a possibilistic disjunctive program. Let $\langle H, T, O\rangle$ be an abductive diagnosis program with $H = \{r'_i \mid (r_i, c_i) \in P, c_i \leq 1 - \lambda\}$, $T = P_{\text{elem}}(\lambda)$ and $O = \{l\}$. We have that $\Pi(P \models^{\text{c}} l) \geq \lambda$ iff the abductive diagnosis program $\langle H, T, O\rangle$ has a cautious explanation.

Proof. We want to determine whether $\Pi\left(P \models^{\text{c}} l\right) \geq \lambda$. By Definition 29 we know this is equivalent to $\max\left\{\pi_P(P') \mid P' \subseteq P \text{ and } P'^* \models^{\text{c}} l\right\} \geq \lambda$, or, determining whether there exists some subprogram $P' \subseteq P$ with $P'^* \models^{\text{c}} l$ such that $\pi_P(P') \geq \lambda$. Since we want $\pi_P(P') \geq \lambda$, we must have that $P_{\text{req}} \subseteq P'^*$ with $P_{\text{req}} = \{r \mid (r, c) \in P, c > 1 - \lambda\}$. Thus the problem reduces to determining whether $P_{\text{opt}} \subseteq \{r \mid (r, c) \in P, c \leq 1 - \lambda\}$ we have $P'^* = (P_{\text{req}} \cup P_{\text{opt}})$ such that $P'^* \models^{\text{c}} l$. By construction of $P_{\text{elem}}(\lambda)$ we know that every rule in P_{req} is applied. Furthermore, every explanation $E \subseteq H$ corresponds with a choice of P_{opt}. This choice of P_{opt} will be applied by the rules in $P_{\text{elem}}(\lambda)$. Finally, the observation O is that 'l' must be a conclusion. Clearly, then $\Pi\left(P \models^{\text{c}} l\right) \geq \lambda$ when $P_{\text{elem}}(\lambda) \cup E \models^{\text{c}} O$, i.e. when $\langle H, T, O\rangle$ has a cautious explanation. □

Proposition 36

Let P be a possibilistic disjunctive program. Let $\langle H, T, O\rangle$ be an abductive diagnosis program with $H = \{r'_i \mid (r_i, c_i) \in P, c_i \leq \lambda'\}$, $T = P_{\text{elem}}(1-\lambda') \cup \{\mathit{missing} \leftarrow \mathit{not}\ l\}$ and $O = \{\mathit{missing}\}$. Let $\lambda' \in \mathit{cert}^+(P)$ be such that $\lambda' < \lambda$ and for which we have that $\nexists \lambda'' \in \mathit{cert}^+(P) \cdot \lambda' < \lambda'' < \lambda$. We have that $N(P \models^{\text{b}} l) \geq \lambda$ iff the abductive diagnosis program $\langle H, T, O\rangle$ has no cautious explanations.

Proof. We need to determine whether

$$\min\left\{1 - \pi_P(P') \mid P' \subseteq P \text{ and } P'^* \not\models^{\text{b}} l\right\} \geq \lambda.$$

In other words, we need to determine if for every $P' \subseteq P$ with $P'^* \not\models^{\text{b}} l$ we have that $\pi_P(P') \leq 1 - \lambda$, or, that for every $P' \subseteq P$ with $\pi_P(P') > 1 - \lambda$ we have that $P'^* \models^{\text{b}} l$. Clearly, $P_{\text{elem}}(1 - \lambda')$ ensures that all the rules with a high enough certainty are applied to ensure that $\pi_P(P') > 1 - \lambda$. Finally, the rule $(\mathit{missing} \leftarrow \mathit{not}\ l)$ is used to verify, together with the observation O, whether $P'^* \models^{\text{b}} l$. Indeed, $\mathit{missing}$ is true in an answer set when l is not in the answer set. Thus, to verify whether $P'^* \models^{\text{b}} l$ it suffices to verify whether $P'^* \not\models^{\text{c}} \mathit{missing}$. The remainder of the proof is then analogous to the proof in Proposition 35. □

Proposition 37

Let P_{abd} be the possibilistic normal program defined for an abductive diagnosis program $\langle H, T, O\rangle$ as

$$\begin{aligned}
&\{\mathbf{0.5}\colon block\_h \leftarrow \mid h \in H\} && (5.3)\\
&\cup \{\mathbf{1}\colon h \leftarrow not\ block\_h \mid h \in H\} && (5.4)\\
&\cup \{\mathbf{1}\colon goal \leftarrow O\} && (5.5)\\
&\cup \{\mathbf{1}\colon r \mid r \in T\}\,. && (5.6)
\end{aligned}$$

It holds that $\langle H, T, O\rangle$ has a cautious explanation iff $\Pi\left(P_{\text{abd}} \models^{\text{c}} goal\right) \geq 0.5$. In particular, E is a cautious explanation iff for $P' = P_{\text{abd}} \setminus \{block\_h \leftarrow \mid h \in E\}$ we have that $(P')^* \models^{\text{c}} goal$.

Proof. For $\Pi\left(P_{\text{abd}} \models^{\text{c}} goal\right) \geq 0.5$, we must have some $P' \subseteq P_{\text{abd}}$ such that $P' \models^{\text{c}} goal$ and $\pi(P') \geq 0.5$. Thus, clearly, all the rules defined in (5.4), (5.5) and (5.6) must be in P'. It is furthermore easy to see that for every $h \in E$ we have that $(h \leftarrow) \in \left((P')^*\right)^M$ (since the corresponding rules from (5.3) are removed from the subprogram P') for every answer set M of P'^* and, since $(P')^* \models^{\text{c}} goal$, that E is a cautious explanation. ☐

Proposition 38

Let P_{con} be the possibilistic normal program defined for a conformant planning problem with the atom '$goal$' the desired goal fluent. We express the domain knowledge as a normal ASP program $P_{\text{act}} \cup P_{\text{rem}}$. Then P_{con} is:

$$\begin{aligned}
&\{\mathbf{0.5}\colon block\_i \leftarrow \mid r_i \in P_{\text{act}}\} && (5.7)\\
&\cup \{\mathbf{1}\colon head(r_i) \leftarrow body(r_i) \cup \{not\ block\_i\} \mid r_i \in P_{\text{act}}\} && (5.8)\\
&\cup \{\mathbf{1}\colon r \mid r \in P_{\text{rem}}\} && (5.9)\\
&\cup \{\mathbf{1}\colon\ \leftarrow not\ goal\} && (5.10)
\end{aligned}$$

A conformant plan exists iff $\Pi\left(P_{\text{con}} \models^{\text{c}} goal\right) \geq 0.5$.

Proof. When $\Pi\left(P_{\text{con}} \models^{\text{c}} goal\right) \geq 0.5$ then, by definition, there exists a subprogram $P' \subseteq P$ such that $(P')^* \models^{\text{c}} goal$ with $\pi_P(P') \geq 0.5$. Since $\pi_P(P') \geq 0.5$ we know that all the rules from (5.8), (5.9) and (5.10) are in P'. Thus, only rules from (5.7) may be in $P \setminus P'$.

In that case, the corresponding rule from (5.8) ensures that for every answer set M of $(P')^*$ we have that $(head(r_i) \leftarrow body(r_i)) \in ((P')^*)^M$. Thus, the action is no longer blocked and can be applied. Because of the available actions we can, regardless of the initial state described in (5.9), cautiously derive '$goal$'. Indeed, otherwise we know due to (5.10) that M is not a model. In other words: the choice made in (5.7) corresponds with a set of actions that form a cautious plan for the given planning problem. $\square$

Proofs of Chapter 6

Proposition 39

Let $\mathcal{P}$ be a communicating disjunctive program. Let P' be the disjunctive program defined as follows. For every $Q\!:\!l \in (\mathcal{B}_{\mathcal{P}} \cup \neg\mathcal{B}_{\mathcal{P}})$ we add the following rules to P':

$$\begin{aligned} guess(Q\_l) &\leftarrow not\ not\_guess(Q\_l) & not\_guess(Q\_l) &\leftarrow not\ guess(Q\_l) \\ &\leftarrow guess(Q\_l), not\ Q\_l & &\leftarrow not\_guess(Q\_l), Q\_l. \end{aligned} \tag{6.1}$$

For every disjunctive communicating rule $\mathcal{P}$ of the form

$$r = Q\!:\!\gamma \leftarrow body$$

with $Q \in \mathcal{P}$, γ a set of literals and $body$ a set of extended situated literals, we add the rule $Q\_\gamma \leftarrow body'$ to P' with Q_γ the disjunctive set of situated literals $\{Q\_l \mid l \in \gamma\}$. We define $body'$ as follows:

$$\begin{aligned} body' = &\{Q\_b \mid Q\!:\!b \in body\} \\ &\cup \{guess(R\_c) \mid R\!:\!c \in body, Q \neq R\} \\ &\cup \{not\ S\_d \mid (not\ S\!:\!d) \in body\}\,. \end{aligned} \tag{6.2}$$

Note that we do not guess situated literals preceded by not, as the guess is already implicitly performed by the reduct of a classical ASP program. We have that $M = \{Q\!:\!l \mid Q\_l \in M'\}$ is an answer set of $\mathcal{P}$ if and only if M' is an answer set of P.

Proof. The essential difference between a disjunctive program and a communicating disjunctive program is in the reduct. More specifically, the difference is in the treatment of situated literals of the form $R\!:\!l$ which are not Q-local. Indeed, such literals can, like naf-literals, be guessed and verified whether or not they are stable, i.e. whether or not the minimal model of the reduct corresponds to the initial guess. It can readily be seen that this behaviour is mimicked by the rules in (6.1). The first two rules guess whether or not some situated literal $Q\!:\!l$ is true, while the last two rules ensure that our guess is stable; i.e. we are only allowed to guess $Q\!:\!l$ when we are later on actually capable of deriving $Q\!:\!l$. The purpose of (6.2) is then to ensure that guessing of situated literals is only used when the situated literal in question is not Q-local and is not preceded by negation-as-failure. □

Proposition 40

Let P be a possibilistic normal program and Q the simulation of P defined as in Definition 33 with $C = cert^+(P)$. Let V be the valuation defined by $V(l) = \max\{\lambda \mid l\lambda \in M\}$. If M is a classical answer set of Q, then V is a possibilistic answer set of P.

Proof. Let $(r, \lambda) \in P$ with $r = (l_0 \leftarrow l_1, ..., l_m, not\, l_{m+1}, ..., not\, l_n)$. Due to the construction of Q, as given in Definition 33, we know that for each $(r, \lambda) \in P$ and for each $\mu \in C$ we have $(l_0\mu \leftarrow l_1\mu, ..., l_m\mu, not\, l_{m+1}\nu, ..., not\, l_n\nu) \in Q$ with $\nu = \min\{\xi \mid \xi > 1 - \mu, \xi \in C\}$. By Definition 5 on the classical reduct we know that $(l_0\mu \leftarrow l_1\mu, ..., l_m\mu) \in Q^M$ if and only if $\{l_{m+1}\nu, ..., l_n\nu\} \cap M = \emptyset$. By construction of V we thus know that $V(l_i) < \nu$ for all $i \in \{m + 1, ..., n\}$. Specifically, since we know that there does not exist some ξ such that $\nu > \xi > 1 - \mu$, we have that $V(l_i) \leq 1 - \mu$ or, equivalently, that $l_i \notin V^{\underline{1-\mu}}$. From Definition 26, we then find that $(r', \lambda') \in P^V$ with $r' = (l_0 \leftarrow l_1, ..., l_m)$ and with $\lambda' = \min(\lambda, \nu)$. Note in particular that we always implicitly have that $(r', \lambda'') \in P^V$ whenever $(r', \lambda') \in P^V$ and $\lambda'' \leq \lambda'$. Conversely we know that $(r', \lambda') \in P^V$ with $\lambda' \leq \lambda$ and $r' = (l_0 \leftarrow l_1, ..., l_m)$ iff $\{l_{m+1}, ..., l_n\} \cap V^{\underline{1-\lambda'}} = \emptyset$. By construction of V this means that $l_i\lambda'' \notin M$ for all $\lambda'' > 1 - \lambda'$ and all $i \in \{m + 1, ..., n\}$. By definition of the Gelfond-Lifschitz reduct, we thus obtain that $(l_0\lambda' \leftarrow l_1\lambda', ..., l_m\lambda') \in Q^M$.

Now let M_k be the result of k times applying the immediate consequence operator T_{Q^M}, starting from the empty interpretation. Similarly, let V_k be the valuation resulting from k times applying the possibilistic immediate consequence operator T_{P^V}, starting from the minimal valuation. We show by induction that for all $l \in Lit_P$ and all λ in C ($\lambda > 0$), it holds that $l\lambda \in M_k$ iff $V_k(l) \geq \lambda$, for all $\lambda \in \mathbb{N}$. Clearly this holds for $k = 0$.

To show the induction step, assume that $l_0\lambda' \in M_k \setminus M_{k-1}$ with $k \geq 1$. This means that there is a rule $(l_0\lambda' \leftarrow l_1\lambda', ..., l_m\lambda') \in Q^M$ such that $l_1\lambda' \in M_{k-1}, ..., l_m\lambda' \in M_{k-1}$. This means however, that $V_{k-1}(l_1) \geq \lambda', ..., V_{k-1}(l_m) \geq \lambda'$ and thus that $V_k(l_0) \geq \lambda'$ is obtained by applying T_{P^V} on V_{k-1}, since we already established that P^V contains the rules $(l_0 \leftarrow l_1, ..., l_m)$ with certainty λ'. Conversely, we also find that when $V_k(l_0) \geq \lambda'$ and $V_{k-1}(l_0) < \lambda'$, it must be the case that $l_0\lambda' \in M_k \setminus M_{k-1}$. Thus we find that the least fixpoint of T_{P^V} is V and therefore, by Proposition 20, that V is a possibilistic answer set of P. $\square$

Proposition 41

Let P be a possibilistic normal program and Q the simulation of P defined as in Definition 33, and $M = \{l\lambda \mid \lambda \leq V(l), \lambda \in C\}$. If V is a possibilistic answer set of P such that $V(l) \in C$ for all $l \in \mathcal{B}_P$, then M is a classical answer set of Q.

Proof. Entirely analogous to the proof of Proposition 40. ☐

Proposition 42

Let P be a possibilistic disjunctive program and $P^{\Pi}_{brave}(l, \lambda)$ the classical disjunctive program as defined in Definition 35. We have that $\Pi(P \models^{\text{b}} l) \geq \lambda$ iff $P^{\Pi}_{brave}(l, \lambda)$ has a classical consistent answer set.

Proof. We want to determine whether $\Pi\left(P \models^{\text{b}} l\right) \geq \lambda$. By Definition 29 we know this is equivalent to $\max\left\{\pi_P(P') \mid P' \subseteq P \text{ and } P'^{*} \models^{\text{b}} l\right\} \geq \lambda$, or, determining whether there exists some subprogram $P' \subseteq P$ with $P'^{*} \models^{\text{b}} l$ such that $\pi_P(P') \geq \lambda$. Since we want $\pi_P(P') \geq \lambda$, this implies that $P_{\text{req}} \subseteq P'^{*}$ with $P_{\text{req}} = \{r \mid (r, c) \in P, c > 1 - \lambda\}$. Thus the problem reduces to determining whether for the rules $P_{\text{opt}} \subseteq \{r \mid (r, c) \in P, c \leq 1 - \lambda\}$ we have $P'^{*} = (P_{\text{req}} \cup P_{\text{opt}})$ such that $P'^{*} \models^{\text{b}} l$. By construction of $P_{\text{basic}}(\lambda)$, in particular due to the rules (6.4), we know that every rule in P_{req} is chosen. Furthermore, every choice made in (6.3) corresponds with a choice of P_{opt}. This choice P_{opt}, along with the rules P_{req}, will be applied by the rules in P_{basic} due to the rules (6.5). Finally, the addition of the rule $\{\leftarrow \textit{not } l\}$ ensures that 'l' must be a conclusion of some answer set of the simulation $P^{\Pi}_{\text{brave}}(l, \lambda)$, or otherwise $P^{\Pi}_{\text{brave}}(l, \lambda)$ will not have any answer sets. Clearly, then $\Pi\left(P \models^{\text{b}} l\right) \geq \lambda$ when $P^{\Pi}_{\text{brave}}(l, \lambda)$ has a classical consistent answer set. ☐

Proposition 43

Let P be a possibilistic disjunctive program, $\lambda > 0$ and $P^{N}_{cautious}(l, \lambda)$ the classical disjunctive program as defined in Definition 36. We have that $N(P \models^{\text{c}} l) \geq \lambda$ iff $P^{N}_{cautious}(l, \lambda)$ has no classical consistent answer set.

Proof. Analogous to the proof in Proposition 42. ☐

Proposition 44

Let P be a possibilistic normal program and $P_{\Pi}^{\mathrm{c}}(l, \lambda)$ the disjunctive program defined as $P_{complex}(\lambda) \cup \{sat \leftarrow l\}$. Then $\Pi\,(P \models^{\mathrm{c}} l) \geq \lambda$ iff $P_{\Pi}^{\mathrm{c}}(l, \lambda)$ has a classical answer set.

Proof. We want to determine whether $\Pi\,(P \models^{\mathrm{c}} l) \geq \lambda$, i.e. whether there exists a $P' \subseteq P$ such that $P'^{*} \models^{\mathrm{c}} l$ and $\pi_P(P') \geq \lambda$. The latter condition means that $(r, c) \in P'$ for every $(r, c) \in P$ with $c > 1 - \lambda$. Similar as in Proposition 42, the rules in (6.6) and (6.7) generate as many answer sets as there are subprograms $P' \subseteq P$ for which $\pi_P(P') \geq \lambda$.

For each such subprogram P' we want to determine whether P'^{*} has 'l' a cautious conclusion. By construction, $\{cl \leftarrow \mid cl \in cls(P^r)\}$ is equivalent to P^r. In particular, every model of these rules corresponds to an answer set of P^r. Since we removed classical negation in (6.8), however, we need to add the rules in (6.9) to ensure that 'sat' is contained in the answer set whenever 'a' and the opposite atom 'na' are true at the same time. The intuition of making 'sat' true is thus to indicate that this is not a valid answer set of the subprogram P'. The rule $(sat \leftarrow l)$ is used to try to make 'l' false, by once again ensuring that 'sat' is contained in the answer set whenever 'l' is in the answer set. Intuitively, we thus say that an answer set in which 'l' is true is undesirable, i.e. we prefer answer sets of P' in which 'l' is false. The rule (6.11) is then used to block all answer sets in which 'sat' is false. In other words: unless for every answer set of P' we have that 'l' is true in the answer set, we have not found that 'l' is a cautious consequence of P'.

Thus far we have not discussed the use of the rules (6.10). Together with the atom 'sat', these rules are used to implement a saturation technique [Baral 2003] over our disjunctive simulation, as discussed in detail in Section 2.2. To recount, the intuition of saturation is that we use the property that an answer set is a *minimal* model. In particular, the rules in (6.10) will add all the atoms under consideration to the model M to try and prevent it from being an answer set. Indeed, if we find a model $M' \subseteq M$ then clearly M cannot be an answer set. As such, we can ensure that consistent models of P' are preferred over inconsistent models, and that models of P' in which 'l' is false are preferred over models in which 'l' is true. Then, only if no consistent answer set (in which 'l' is false) exists for P', will we have that 'sat' is true in an answer set of $P_{\Pi}^{\mathrm{c}}(l, \lambda)$.

Finally, when a subprogram P' is inconsistent, then $\pi(P') = 0$, i.e. we do not want to consider this subprogram. Notice, however, that the rule (6.9) would not work as expected in this case. Indeed, if P' is inconsistent it does not have a consistent model and the saturation technique would not exclude this subprogram. As such, we repeat our simulation of the subprogram P' in (6.12) and use constraints in (6.13) to effectively block inconsistent subprograms. $\square$

Proposition 45

Let P be a possibilistic normal program and $P_{\mathrm{N}}^{\mathrm{b}}(l,\lambda)$ the disjunctive program defined as $P_{complex}(1-\lambda') \cup \{sat \leftarrow not\ l\}$ with λ' defined as in Proposition 43. Then $N\left(P \models^{\mathrm{b}} l\right) \geq \lambda$ iff $P_{\mathrm{N}}^{\mathrm{b}}(l,\lambda)$ has no classical answer set.

Proof. This proof is analogous to the proof of Proposition 44, similar as how the proof of Proposition 43 was analogous to the proof of Proposition 42. □

Appendices

Appendix 1: ASP encoding of Example 49

Below is a listing of the ASP encoding of the conformant planning problem described in Example 49. Notice that the ASP encoding of Example 49 does not compute a conformant plan. Indeed, the $DLV^{\mathcal{K}}$ uses a two-step process to compute the conformant plan. In particular, this encoding merely defines the behaviour of the conformant planning problem but does not encode the knowledge on how to determine whether such a plan is indeed conformant. This is to be expected, since $DLV^{\mathcal{K}}$ first computes a plan and then verifies whether such a plan is conformant.

```
% actions can be performed or not
dunk(1,0) ; -dunk(1,0).
dunk(2,0) ; -dunk(2,0).
dunk(3,0) ; -dunk(3,0).
dunk(4,0) ; -dunk(4,0).
dunk(1,1) ; -dunk(1,1).
dunk(2,1) ; -dunk(2,1).
dunk(3,1) ; -dunk(3,1).
dunk(4,1) ; -dunk(4,1).
dunk(1,2) ; -dunk(1,2).
dunk(2,2) ; -dunk(2,2).
dunk(3,2) ; -dunk(3,2).
dunk(4,2) ; -dunk(4,2).
dunk(1,3) ; -dunk(1,3).
dunk(2,3) ; -dunk(2,3).
dunk(3,3) ; -dunk(3,3).
dunk(4,3) ; -dunk(4,3).

% dunking a package disables the
% package in the next time step
-armed(2,1) :- dunk(2,0).
-armed(4,1) :- dunk(4,0).
-armed(3,1) :- dunk(3,0).
-armed(1,1) :- dunk(1,0).
-armed(2,2) :- dunk(2,1).
-armed(4,2) :- dunk(4,1).
-armed(3,2) :- dunk(3,1).
-armed(1,2) :- dunk(1,1).
-armed(2,3) :- dunk(2,2).
-armed(4,3) :- dunk(4,2).
-armed(3,3) :- dunk(3,2).
-armed(1,3) :- dunk(1,2).
-armed(2,4) :- dunk(2,3).
-armed(4,4) :- dunk(4,3).
-armed(3,4) :- dunk(3,3).
-armed(1,4) :- dunk(1,3).

% inertia w.r.t. -armed
-armed(2,2) :- -armed(2,1).
-armed(4,2) :- -armed(4,1).
-armed(3,2) :- -armed(3,1).
-armed(1,2) :- -armed(1,1).
-armed(2,3) :- -armed(2,2).
-armed(4,3) :- -armed(4,2).
-armed(3,3) :- -armed(3,2).
-armed(1,3) :- -armed(1,2).
-armed(2,4) :- -armed(2,3).
-armed(4,4) :- -armed(4,3).
-armed(3,4) :- -armed(3,3).
-armed(1,4) :- -armed(1,3).

% the state remains unsafe unless
% all packages have been disarmed
% by a given time step
unsafe(1) :- not -armed(1,1).
unsafe(1) :- not -armed(2,1).
unsafe(1) :- not -armed(3,1).
unsafe(1) :- not -armed(4,1).
unsafe(2) :- not -armed(1,2).
unsafe(2) :- not -armed(2,2).
unsafe(2) :- not -armed(3,2).
unsafe(2) :- not -armed(4,2).
unsafe(3) :- not -armed(1,3).
unsafe(3) :- not -armed(2,3).
unsafe(3) :- not -armed(3,3).
unsafe(3) :- not -armed(4,3).
unsafe(4) :- not -armed(1,4).
unsafe(4) :- not -armed(2,4).
unsafe(4) :- not -armed(3,4).
unsafe(4) :- not -armed(4,4).

% we enforce concurrent events, i.e.
% we cannot dunk two packages at the
% same time step.
:- dunk(2,0), dunk(4,0).
:- dunk(2,0), dunk(3,0).
:- dunk(2,0), dunk(1,0).
:- dunk(4,0), dunk(2,0).
:- dunk(4,0), dunk(3,0).
:- dunk(4,0), dunk(1,0).
:- dunk(3,0), dunk(2,0).
:- dunk(3,0), dunk(4,0).
:- dunk(3,0), dunk(1,0).
:- dunk(1,0), dunk(2,0).
:- dunk(1,0), dunk(4,0).
:- dunk(1,0), dunk(3,0).
:- dunk(2,1), dunk(4,1).
:- dunk(2,1), dunk(3,1).
:- dunk(2,1), dunk(1,1).
:- dunk(4,1), dunk(2,1).
:- dunk(4,1), dunk(3,1).
:- dunk(4,1), dunk(1,1).
:- dunk(3,1), dunk(2,1).
:- dunk(3,1), dunk(4,1).
:- dunk(3,1), dunk(1,1).
:- dunk(1,1), dunk(2,1).
:- dunk(1,1), dunk(4,1).
:- dunk(1,1), dunk(3,1).
:- dunk(2,2), dunk(4,2).
:- dunk(2,2), dunk(3,2).
:- dunk(2,2), dunk(1,2).
:- dunk(4,2), dunk(2,2).
:- dunk(4,2), dunk(3,2).
:- dunk(4,2), dunk(1,2).
:- dunk(3,2), dunk(2,2).
:- dunk(3,2), dunk(4,2).
:- dunk(3,2), dunk(1,2).
:- dunk(1,2), dunk(2,2).
:- dunk(1,2), dunk(4,2).
:- dunk(1,2), dunk(3,2).
:- dunk(2,3), dunk(4,3).
:- dunk(2,3), dunk(3,3).
:- dunk(2,3), dunk(1,3).
:- dunk(4,3), dunk(2,3).
:- dunk(4,3), dunk(3,3).
:- dunk(4,3), dunk(1,3).
:- dunk(3,3), dunk(2,3).
:- dunk(3,3), dunk(4,3).
:- dunk(3,3), dunk(1,3).
:- dunk(1,3), dunk(2,3).
:- dunk(1,3), dunk(4,3).
:- dunk(1,3), dunk(3,3).

% the goal is to reach
% a safe state in 4 steps
goal :- not unsafe(4).

% omit answer sets that
% do not reach the goal
:- not goal.
```

Appendix 2: conformant plan encoding in PASP$_r$ (Proposition 38)

The PASP$_r$ program corresponding with Appendix 1 according to Proposition 38 assumes all rules to be certain, except for those rules encoding an action. These action rules are encoded as follows:

```
% optional rules describe if the action is blocked
0.5: block_dunk(1,0).
0.5: block_dunk(2,0).
0.5: block_dunk(3,0).
0.5: block_dunk(4,0).
0.5: block_dunk(1,1).
0.5: block_dunk(2,1).
0.5: block_dunk(3,1).
0.5: block_dunk(4,1).
0.5: block_dunk(1,2).
0.5: block_dunk(2,2).
0.5: block_dunk(3,2).
0.5: block_dunk(4,2).
0.5: block_dunk(1,3).
0.5: block_dunk(2,3).
0.5: block_dunk(3,3).
0.5: block_dunk(4,3).

% if the action is not blocked, it is executed
1: dunk(1,0) :- not block_dunk(1,0).
1: dunk(2,0) :- not block_dunk(2,0).
```

```
1: dunk(3,0) :- not block_dunk(3,0).
1: dunk(4,0) :- not block_dunk(4,0).
1: dunk(1,1) :- not block_dunk(1,1).
1: dunk(2,1) :- not block_dunk(2,1).
1: dunk(3,1) :- not block_dunk(3,1).
1: dunk(4,1) :- not block_dunk(4,1).
1: dunk(1,2) :- not block_dunk(1,2).
1: dunk(2,2) :- not block_dunk(2,2).
1: dunk(3,2) :- not block_dunk(3,2).
1: dunk(4,2) :- not block_dunk(4,2).
1: dunk(1,3) :- not block_dunk(1,3).
1: dunk(2,3) :- not block_dunk(2,3).
1: dunk(3,3) :- not block_dunk(3,3).
1: dunk(4,3) :- not block_dunk(4,3).
```

Appendix 3: output of PASP2SAT on Example 47

```
extensive.
minor :- not extensive, r1.
moaning.
conscious :- moaning, r2.
nowait :- not beyond, not internal, not conscious, extensive, r3.
beyond :- not nowait, not conscious, extensive, r4.
nosebleed.
internal :- nosebleed, r5.
internal :- nosebleed, lowblood, r6.
 :- nowait, beyond, extensive.
 :- not nowait, not beyond, extensive.
```

Appendix 4: output of PASP2ASP for deciding $P \models^{\mathrm{b}}_{N} \mathit{beyond}^{0.8}$ of Example 47

The output found below is the disjunctive ASP program that simulates the decision problem $P \models^{\mathrm{b}}_{N} \mathit{beyond}^{0.8}$ for Example 47 based on the implementation defined in Definition 37. Since the output of PASP2SAT as given in Appendix 3 can be verified to be tight, the completion as defined in Section 2.2.4 can be used to transform the program into set of clauses. The output program, which has answer sets if and only if $P \models^{\mathrm{b}}_{N} \mathit{beyond}^{0.8}$ is given by:

```
r1.
r2 :- not n_r2.
n_r2 :- not r2.
r3.
```

```
r4.
r5 :- not n_r5.
n_r5 :- not r5.
r6.

n_nowait; n_beyond; n_extensive.
nowait; beyond; n_extensive.
extensive.
minor; extensive; -r1.
n_minor; n_extensive.
n_minor; r1.
moaning.
conscious; n_moaning; -r2.
n_conscious; moaning.
n_conscious; r2.
nowait; beyond; internal; conscious; n_extensive; -r3.
n_nowait; n_beyond.
n_nowait; n_internal.
n_nowait; n_conscious.
n_nowait; extensive.
n_nowait; r3.
beyond; nowait; conscious; n_extensive; -r4.
n_beyond; n_nowait.
n_beyond; n_conscious.
n_beyond; extensive.
n_beyond; r4.
internal; n_nosebleed; -r5.
internal; n_nosebleed; n_lowblood; -r6.
n_internal; nosebleed; nosebleed.
n_internal; nosebleed; r5.
n_internal; lowblood; nosebleed.
n_internal; lowblood; r5.
n_internal; r6; nosebleed.
n_internal; r6; r5.
nosebleed.
n_lowblood.

extensive :- saturate.
n_extensive :- saturate.
minor :- saturate.
n_minor :- saturate.
moaning :- saturate.
n_moaning :- saturate.
conscious :- saturate.
n_conscious :- saturate.
nowait :- saturate.
n_nowait :- saturate.
beyond :- saturate.
```

```
n_beyond :- saturate.
internal :- saturate.
n_internal :- saturate.
nosebleed :- saturate.
n_nosebleed :- saturate.
lowblood :- saturate.
n_lowblood :- saturate.

n_nowait_s; n_beyond_s; n_extensive_s.
nowait_s; beyond_s; n_extensive_s.
extensive_s.
minor_s; extensive_s; -r1.
n_minor_s; n_extensive_s.
n_minor_s; r1.
moaning_s.
conscious_s; n_moaning_s; -r2.
n_conscious_s; moaning_s.
n_conscious_s; r2.
nowait_s; beyond_s; internal_s; conscious_s; n_extensive_s; -r3.
n_nowait_s; n_beyond_s.
n_nowait_s; n_internal_s.
n_nowait_s; n_conscious_s.
n_nowait_s; extensive_s.
n_nowait_s; r3.
beyond_s; nowait_s; conscious_s; n_extensive_s; -r4.
n_beyond_s; n_nowait_s.
n_beyond_s; n_conscious_s.
n_beyond_s; extensive_s.
n_beyond_s; r4.
internal_s; n_nosebleed_s; -r5.
internal_s; n_nosebleed_s; n_lowblood_s; -r6.
n_internal_s; nosebleed_s; nosebleed_s.
n_internal_s; nosebleed_s; r5.
n_internal_s; lowblood_s; nosebleed_s.
n_internal_s; lowblood_s; r5.
n_internal_s; r6; nosebleed_s.
n_internal_s; r6; r5.
nosebleed_s.
n_lowblood_s.

saturate :- extensive, n_extensive.
saturate :- minor, n_minor.
saturate :- moaning, n_moaning.
saturate :- conscious, n_conscious.
saturate :- nowait, n_nowait.
saturate :- beyond, n_beyond.
saturate :- internal, n_internal.
saturate :- nosebleed, n_nosebleed.
saturate :- lowblood, n_lowblood.

 :- extensive_s, n_extensive_s.
 :- minor_s, n_minor_s.
 :- moaning_s, n_moaning_s.
 :- conscious_s, n_conscious_s.
 :- nowait_s, n_nowait_s.
 :- beyond_s, n_beyond_s.
 :- internal_s, n_internal_s.
 :- nosebleed_s, n_nosebleed_s.
 :- lowblood_s, n_lowblood_s.


saturate :- beyond.
:- not saturate.
```

List of Publications

2013 **Semantics for possibilistic answer set programs: uncertain rules versus rules with uncertain conclusions** (Kim Bauters, Steven Schockaert, Martine De Cock, Dirk Vermeir), *submitted to a journal listed in the Web of Science SCI - Science Citation Index*.

2013 **Characterizing and Extending Answer Set Semantics using Possibility Theory** (Kim Bauters, Steven Schockaert, Martine De Cock, Dirk Vermeir), *submitted to a journal listed in the Web of Science SCI - Science Citation Index*.

2013 **Answer set programs with optional rules: a possibilistic approach** (Kim Bauters, Steven Schockaert, Martine De Cock, Dirk Vermeir), *accepted for the 2nd Workshop on Weighted Logics for Artificial Intelligence (WL4AI)*, 2013.

2013 **Expressiveness of Communication in Answer Set Programming** (Kim Bauters, Jeroen Janssen, Steven Schockaert, Martine De Cock, Dirk Vermeir), *Theory and Practice of Logic Programming (TPLP)*, 13(3), pp. 361–394, 2013.

2012 **Possible and Necessary Answer Sets of Possibilistic Answer Set Programs** (Kim Bauters, Steven Schockaert, Martine De Cock, Dirk Vermeir), *In Proceedings of the 24th International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 836–843, 2012.

2011 **Weak and strong disjunction in possibilistic ASP** (Kim Bauters, Steven Schockaert, Martine De Cock, Dirk Vermeir), *In Proceedings of the 5th International Conference on Scalable Uncertainty Management (SUM)*, volume 6929, pp. 475–488, 2011.

2011 **Communicating ASP and the polynomial hierarchy** (Kim Bauters, Steven Schockaert, Martine De Cock, Dirk Vermeir), *In Proceedings of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, volume 6645, pp. 67–79, 2011.

2011 **Modeling coalition formation using multi-focused answer sets** (Kim Bauters), *In European Summer School in Logic, Language and Informatics (ESSLLI) Student Session*, 2011.

2010 **Possibilistic Answer Set Programming Revisited** (Kim Bauters, Steven Schockaert, Martine De Cock, Dirk Vermeir), *In Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 48–55, 2010.

2010 **Communicating Answer Set Programs** (Kim Bauters, Jeroen Janssen, Steven Schockaert, Dirk Vermeir, Martine De Cock), *In Technical Communications of the 26th International Conference on Logic Programming*, volume 7, pp. 34–43, 2010.

2010 **Towards Possibilistic Fuzzy Answer Set Programming** (Kim Bauters, Steven Schockaert, Martine De Cock, Dirk Vermeir), *In Proceedings of the 13th International Workshop on Non-monotonic reasoning (NMR)*, 2010.

Bibliography

[Amgoud and Prade 2004] Leila Amgoud and Henri Prade (2004). *Reaching Agreement Through Argumentation: A Possibilistic Approach*. In *Proceedings of the 9th International Conference on Principles of Knowledge Representation and Reasoning (KR'04)*, pp. 175–182.

[Balduccini and Gelfond 2003] Marcello Balduccini and Michael Gelfond (2003). *Logic Programs with Consistency-Restoring Rules*. In *International Symposium on Logical Formalization of Commonsense Reasoning, AAAI 2003 Spring Symposium Series*, pp. 9–18.

[Banerjee and Dubois 2009] Mohua Banerjee and Didier Dubois (2009). *A Simple Modal Logic for Reasoning about Revealed Beliefs*. In *Proceedings of the 10th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU'09)*, pp. 805–816.

[Baral 2003] Chitta Baral (2003). *Knowledge, Representation, Reasoning and Declarative Problem Solving*: Cambridge University Press.

[Baral et al. 2009] Chitta Baral, Michael Gelfond, and Nelson Rushton (2009). *Probabilistic reasoning with answer sets*. Theory and Practice of Logic Programming, 9(1), 57–144.

[Baral and Subrahmanian 1993] Chitta Baral and VS Subrahmanian (1993). *Dualities between alternative semantics for logic programming and nonmonotonic reasoning*. Journal of Automated Reasoning, 10(3), 399–420.

[Bauters 2011] Kim Bauters (2011). *Modeling coalition formation using multi-focused answer sets*. In *Proceedings of the European Summer School in Logic, Language and Information Student Session (ESSLLI'11)*.

[Bauters et al. 2013a] Kim Bauters, Jeroen Janssen, Steven Schockaert, Martine De Cock, and Dirk Vermeir (2013a). *Expressiveness of Communication in Answer Set Programming*. Theory and Practice of Logic Programming (TPLP), 13(3), 361–394.

[Bauters et al. 2010a] Kim Bauters, Jeroen Janssen, Steven Schockaert, Dirk Vermeir, and Martine De Cock (2010a). *Communicating Answer Set Programs*. In *Technical Communications of the 26th International Conference on Logic Programming (ICLP'10)*, vol. 7, pp. 34–43.

[Bauters et al. 2010b] Kim Bauters, Steven Schockaert, Martine De Cock, and Dirk Vermeir (2010b). *Possibilistic Answer Set Programming Revisited*. In *Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence (UAI'10)*, pp. 48–55.

[Bauters et al. 2011a] Kim Bauters, Steven Schockaert, Martine De Cock, and Dirk Vermeir (2011a). *Communicating ASP and the polynomial hierarchy*. In *Proceedings of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11)*, vol. 6645 of *Lecture Notes in Artificial Intelligence*, pp. 67–79.

[Bauters et al. 2011b] Kim Bauters, Steven Schockaert, Martine De Cock, and Dirk Vermeir (2011b). *Weak and strong disjunction in possibilistic ASP*. In *Proceedings of the 5th International Conference on Scalable Uncertainty Management (SUM'11)*, vol. 6929, pp. 475–488.

[Bauters et al. 2012a] Kim Bauters, Steven Schockaert, Martine De Cock, and Dirk Vermeir (2012a). *Characterizing and Extending Answer Set Semantics using Possibility Theory*. Submitted.

[Bauters et al. 2012b] Kim Bauters, Steven Schockaert, Martine De Cock, and Dirk Vermeir (2012b). *Possible and Necessary Answer Sets of Possibilistic Answer Set Programs*. In *Proceedings of the 24th International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 836–843.

[Bauters et al. 2013b] Kim Bauters, Steven Schockaert, Martine De Cock, and Dirk Vermeir (2013b). *Semantics for possibilistic answer set programs: uncertain rules versus rules with uncertain conclusions*. Submitted.

[Benferhat et al. 2002] Salem Benferhat, Didier Dubois, Laurent Garcia, and Henri Prade (2002). *On the transformation between possibilistic logic bases and possibilistic causal networks*. International Journal of Approximate Reasoning, 29(2), 135–173.

[Benferhat et al. 2000] Salem Benferhat, Didier Dubois, Souhila Kaci, and Henri Prade (2000). *Encoding Information Fusion in Possibilistic Logic: A General Framework for Rational Syntactic Merging*. In *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI'00)*, pp. 3–7: IOS Press.

[Benferhat et al. 1992] Salem Benferhat, Didier Dubois, and Henri Prade (1992). *Representing Default Rules in Possibilistic Logic*. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92)*, pp. 673–684.

[Benferhat et al. 1997] Salem Benferhat, Didier Dubois, and Henri Prade (1997). *Nonmonotonic reasoning, conditional objects and possibility theory*. Artificial Intelligence, 92(1–2), 259–276.

[Benferhat and Smaoui 2007] Salem Benferhat and Salma Smaoui (2007). *Possibilistic Causal Networks for Handling Interventions: A New Propagation Algorithm*. In *Proceedings of the 22nd National Conference on Artificial Intelligence (AAAI'07)*.

[Brain and De Vos 2003] Martin Brain and Marina De Vos (2003). *Implementing OCLP as a front-end for Answer Set Solvers: From Theory to Practice*. In *Proceedings of the 2nd international workshop on Answer Set Programming, Advances in Theory and Implementation (ASP'05)*.

[Bremer 2005] Manuel Bremer (2005). *An introduction to paraconsistent logics*: Peter Lang Frankfurt.

[Brewka 2002] Gerhard Brewka (2002). *Logic Programming with Ordered Disjunction*. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI'02)*, pp. 100–105.

[Brewka and Eiter 2007] Gerhard Brewka and Thomas Eiter (2007). *Equilibria in Heterogeneous Nonmonotonic Multi-Context Systems*. In *Proceedings of the 22nd National Conference on Artificial Intelligence (AAAI'07)*, pp. 385–390.

[Brewka et al. 2011] Gerhard Brewka, Thomas Eiter, Michael Fink, and Antonius Weinzierl (2011). *Managed Multi-Context Systems*. In *Proceedings of the 22th International Joint Conferences on Artificial Intelligence IJCAI'11*, pp. 786–791.

[Brewka et al. 2007] Gerhard Brewka, Floris Roelofsen, and Luciano Serafini (2007). *Contextual default reasoning*. In *Proceedings of the 20th international joint conference on Artifical intelligence (IJCAI'07)*, pp. 268–273.

[Buccafurri et al. 2008] Francesco Buccafurri, Gianluca Caminiti, and Rosario Laurendi (2008). *A Logic Language with Stable Model Semantics for Social Reasoning*. In *Proceedings of the 24th International Conference on Logic Programming (ICLP'08)*, pp. 718–723.

[Buccafurri et al. 2002] Francesco Buccafurri, Wolfgang Faber, and Nicola Leone (2002). *Disjunctive Logic Programs with Inheritance*. Theory and Practice of Logic Programming, 2(3), 293–321.

[Buccafurri et al. 1997] Francesco Buccafurri, Nicola Leone, and Pasquale Rullo (1997). *Strong and Weak Constraints in Disjunctive Datalog*. In *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'97)*, pp. 2–17.

[Buccafurri et al. 2000] Francesco Buccafurri, Nicola Leone, and Pasquale Rullo (2000). *Enhancing Disjunctive Datalog by Constraints*. IEEE Transactions on Knowledge and Data Engineering, 12(5), 845–860.

[Cai et al. 1988] Jin-Yi Cai, Thomas Gundermann, Juris Hartmanis, Lane Hemachandra, Vivian Sewelson, Klaus Wagner, and Gerd Wechsung (1988). *The Boolean Hierarchy I: Structural Properties*. SIAM Journal on Computing, 17(6), 1232–1252.

[Calimeri et al. 2011] Francesco Calimeri, Giovambattista Ianni, Francesco Ricca, Mario Alviano, Annamaria Bria, Gelsomina Catalano, Susanna Cozza, Wolfgang Faber, Onofrio Febbraro, Nicola Leone, Marco Manna, Alessandra Martello, Claudio Panetta, Simona Perri, Kristian Reale, Maria Carmela Santoro, Marco Sirianni, Giorgio Terracina, and Pierfrancesco Veltri (2011). *The Third Answer Set Programming Competition: Preliminary Report of the System Competition Track*. In *Proceedings of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11)*, vol. 6645 of *Lecture Notes in Computer Science*, pp. 388–403.

[Chellas 1980] Brian Chellas (1980). *Modal logic: an introduction*: Cambridge University Press.

[Chesñevar et al. 2004] Carlos Chesñevar, Guillermo Simari, Teresa Alsinet, and Lluís Godo (2004). *A Logic Programming Framework for Possibilistic Argumentation with Vague Knowledge*. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence (UAI'04)*.

[Clark 1978] Keith Clark (1978). *Negation as failure*. Logic and data bases, 1, 293–322.

[Cook 1971] Stephen Arthur Cook (1971). *The complexity of theorem-proving procedures*. In *Proceedings of the third annual ACM symposium on Theory of computing (STOC'71)*, pp. 151–158.

[Corcoran 2003] John Corcoran (2003). *Aristotle's Prior Analytics and Boole's Laws of Thought*. History and Philosophy of Logic, 24(4), 261–288.

[Damásio and Pereira 2001] Carlos Viegas Damásio and Luís Moniz Pereira (2001). *Monotonic and Residuated Logic Programs*. In *Proceedings of the 6th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU'01)*, pp. 748–759.

[Dao-Tran et al. 2009] Minh Dao-Tran, Thomas Eiter, Michael Fink, and Thomas Krennwallner (2009). *Modular Nonmonotonic Logic Programming Revisited*. In *Proceedings of the 25th International Conference on Logic Programming (ICLP'09)*, pp. 145–159.

[Dao-Tran et al. 2010] Minh Dao-Tran, Thomas Eiter, Michael Fink, and Thomas Krennwallner (2010). *Distributed Nonmonotonic Multi-Context Systems*. In *Proceedings of the 12th International Conference on Principles of Knowledge Representation and Reasoning (KR'10)*.

[De Vos et al. 2005] Marina De Vos, Tom Crick, Julian Padget, Martin Brain, Owen Cliffe, and Jonathan Needham (2005). *LAIMA: A Multi-agent Platform Using Ordered Choice Logic Programming*. In *Proceedings of the 3rd international workshop on Declarative Agent Languages and Technologies (DALT'05)*, pp. 72–88.

[Delgrande et al. 2004] James P. Delgrande, Torsten Schaub, Hans Tompits, and Kewen Wang (2004). *A Classification and Survey of Preference Handling Approaches in Nonmonotonic Reasoning*. Computational Intelligence, 20(2), 308–334.

[Delgrande et al. 2009] James P. Delgrande, Torsten Schaub, Hans Tompits, and Stefan Woltran (2009). *Merging Logic Programs under Answer Set Semantics*. In *Proceedings of the 25th International Conference on Logic Programming (ICLP'09)*, vol. 5649 of *Lecture Notes in Computer Science*, pp. 160–174.

[Dell'Acqua et al. 1999] Pierangelo Dell'Acqua, Fariba Sadri, and Francesca Toni (1999). *Communicating Agents*. In *Proceedings of the 1st international workshop on Multi-Agent Systems in Production (MAS'99)*.

[Denecker et al. 2009] Marc Denecker, Joost Vennekens, Stephen Bond, Martin Gebser, and Mirosław Truszczyński (2009). *The Second Answer Set Programming Competition*. In *Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR '09)*, pp. 637–654.

[Drescher et al. 2011] Christian Drescher, Thomas Eiter, Michael Fink, Thomas Krennwallner, and Toby Walsh (2011). *Symmetry Breaking for Distributed Multi-Context Systems*. In *Proceedings of the 26th International Conference on Logic Programming (ICLP'10)*, vol. 6645 of *Lecture Notes in Computer Science*, pp. 26–39.

[Dubois et al. 1991] Didier Dubois, Jérôme Lang, and Henri Prade (1991). *Towards Possibilistic Logic Programming*. In *Proceedings of the 8th Joint International Conference and Symposium on Logic Programming (ICLP'91)*, pp. 581–595.

[Dubois et al. 1994] Didier Dubois, Jérôme Lang, and Henri Prade (1994). *Possibilistic logic*. Handbook of Logic for Artificial Intelligence and Logic Programming, 3(1), 439–513.

[Dubois and Prade 1988] Didier Dubois and Henri Prade (1988). *Possibility theory: an approach to computerized processing of uncertainty*: Plenum Press.

[Dubois and Prade 1991] Didier Dubois and Henri Prade (1991). *Epistemic entrenchment and possibilistic logic*. Artificial Intelligence, 50(2), 223–239.

[Dubois and Prade 1994] Didier Dubois and Henri Prade (1994). *Can We Enforce Full Compositionality in Uncertainty Calculi?*. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI'94)*, pp. 149–154.

[Dubois and Prade 1997] Didier Dubois and Henri Prade (1997). *A synthetic view of belief revision with uncertain inputs in the framework of possibility theory*. International Journal of Approximate Reasoning, 17(2-3), 295–324.

[Dubois and Prade 2001] Didier Dubois and Henri Prade (2001). *Possibility Theory, Probability Theory and Multiple-Valued Logics: A Clarification*. Annals of Mathematics and Artificial Intelligence, 32(1-4), 35–66.

[Dubois et al. 1993] Didier Dubois, Henri Prade, and Sandra Sandri (1993). *On Possibility/Probability Transformations*. In *Proceedings of the 5th International Fuzzy Systems Association World Congress (IFSA'93)*, pp. 103–112.

[Dubois et al. 2012] Didier Dubois, Henri Prade, and Steven Schockaert (2012). *Stable Models in Generalized Possibilistic Logic*. In *Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning (KR'12)*, pp. 519–529.

[Eiter et al. 1999] Thomas Eiter, Wolfgang Faber, Nicola Leone, and Gerald Pfeifer (1999). *The Diagnosis Frontend of the dlv system*. AI Communication, 12(1-2), 99–111.

[Eiter et al. 2000] Thomas Eiter, Wolfgang Faber, Nicola Leone, Gerald Pfeifer, and Axel Polleres (2000). *Planning under Incomplete Knowledge*. In *Proceedings of the First International Conference on Computational Logic (CL'2000)*, pp. 807–821.

[Eiter et al. 2003] Thomas Eiter, Wolfgang Faber, Nicola Leone, Gerald Pfeifer, and Axel Polleres (2003). *A Logic Programming Approach to Knowledge-State planning, II: The $DLV^{\mathcal{K}}$ System*. Artificial Intelligence, 144(1–2), 157–211.

[Eiter and Gottlob 1993] Thomas Eiter and Georg Gottlob (1993). *Complexity Results for Disjunctive Logic Programming and Application to Nonmonotonic Logics*. In *Proceedings of the 1993 International Logic Programming Symposium (ILPS'93)*, pp. 266–278.

[Eiter and Gottlob 1995a] Thomas Eiter and Georg Gottlob (1995a). *The complexity of logic-based abduction*. ACM, 42, 3–42.

[Eiter and Gottlob 1995b] Thomas Eiter and Georg Gottlob (1995b). *On the computational cost of disjunctive logic programming: Propositional case*. Annals of Mathematics and Artificial Intelligence, 15, 289–323.

[Eiter et al. 1997] Thomas Eiter, Georg Gottlob, and Nicola Leone (1997). *Abduction from logic programs: Semantics and complexity*. Theoretical Computer Science, 189(1–2), 129–177.

[Eiter et al. 2008] Thomas Eiter, Giovambattista Ianni, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits (2008). *Combining answer set programming with description logics for the Semantic Web*. Artifial Intelligence, 172(12–13), 1495–1539.

[Eiter et al. 2005] Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits (2005). *A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer-Set Programming*. In *Proceedings of the 19th international joint conference on Artifical intelligence (IJCAI'05)*, pp. 90–96.

[Eiter et al. 2006] Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits (2006). *dlvhex: A Tool for Semantic-Web Reasoning under the Answer-Set Semantics*. In *Proceedings of the the international workshop on Applications of Logic Programming to the Web, Semantic Web and Semantic Web Services (ALPSWS'06)*, pp. 33–39.

[Erman et al. 1980] Lee Erman, Frederick Hayes-Roth, Victor Lesser, and Raj Reddy (1980). *The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty*. ACM Computing Surveys, 12(2), 213–253.

[Faber and Woltran 2009] Wolfgang Faber and Stefan Woltran (2009). *Manifold Answer-Set Programs for Meta-reasoning*. In *LPNMR*, vol. 5753 of *Lecture Notes in Computer Science*, pp. 115–128: Springer.

[Fages 1994] François Fages (1994). *Consistency of Clark's completion and existence of stable models*. Methods of Logic in Computer Science, 1(1), 51–60.

[Flach and Kakas 2000] Peter Flach and Antonis Kakas (2000). *On the relation between abduction and inductive learning*, pp. 5–36: Springer Netherlands.

[Garvey 2000] Paul Garvey (2000). *Probability Methods for Cost Uncertainty Analysis: A Systems Engineering Perspective*: Taylor & Francis.

[Gebser et al. 2011] Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Schneider (2011). *Potassco: The Potsdam Answer Set Solving Collection*. AI Communications, 24, 107–124.

[Gebser et al. 2008] Martin Gebser, Jörg Pührer, Torsten Schaub, and Hans Tompits (2008). *A meta-programming technique for debugging answer-set programs*. In *Proceedings of the 23rd National Conference on Artificial Intelligence (AAAI'08)*, pp. 448–453.

[Gelder et al. 1991] Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf (1991). *The Well-Founded Semantics for General Logic Programs*. Journal of the ACM, 38(3), 620–650.

[Gelfond 1987] Michael Gelfond (1987). *On Stratified Autoepistemic Theories*. In *Proceedings of the 6th National Conference on Artificial Intelligence (AAAI'87)*, pp. 207–211.

[Gelfond 1991] Michael Gelfond (1991). *Strong Introspection*. In *Proceedings of the 9th National Conference on Artificial Intelligence (AAAI'91)*, pp. 386–391.

[Gelfond and Lifschitz 1988] Michael Gelfond and Vladimir Lifschitz (1988). *The stable model semantics for logic programming*. In *Proceedings of the 5th Joint International Conference and Symposium on Logic Programming (ICLP'88)*, pp. 1081–1086.

[Gelfond and Lifschitz 1991] Michael Gelfond and Vladimir Lifschitz (1991). *Classical negation in logic programs and disjunctive databases*. New Generation Computing, 9, 365–385.

[Giunchiglia and Serafini 1994] Fausto Giunchiglia and Luciano Serafini (1994). *Multilanguage Hierarchical Logics or: How we can do Without Modal Logics*. Artifial Intelligence, 65(1), 29–70.

[Huth and Ryan 2004] Michael Huth and Mark Ryan (2004). *Logic in Computer Science: Modelling and Reasoning about Systems*: Cambridge University Press.

[Jacquette 2005] Dale Jacquette (2005). *A Companion to Philosophical Logic*: Wiley-Blackwell.

[Janhunen 2004] Tomi Janhunen (2004). *Representing Normal Programs with Clauses*. In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence (ECAI'04)*, pp. 358–362.

[Janhunen 2006] Tomi Janhunen (2006). *Some (in)translatability results for normal logic programs and propositional theories*. Journal of Applied Non-Classical Logics, 16(1-2), 35–86.

[Jaynes 2003] Edwin Jaynes (2003). *Probability Theory: The Logic of Science*: Cambridge University Press.

[Jeroslow 1985] Robert Jeroslow (1985). *The polynomial hierarchy and a simple model for competitive analysis*. Mathematical Programming, 32, 146–164.

[Kersting and De Raedt 2001] Kristian Kersting and Luc De Raedt (2001). *Bayesian Logic Programs*. CoRR.

[Konieczny and Pérez 2011] Sébastien Konieczny and Ramón Pino Pérez (2011). *Logic Based Merging*. Journal of Philosophical Logic, 40(2), 239–270.

[Leone et al. 2006] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello (2006). *The DLV system for knowledge representation and reasoning*. ACM Transactions on Computational Logic (TOCL), 7(3), 499–562.

[Lewis 1973] David Lewis (1973). *Counterfactuals*: Oxford: Blackwell Publishers and Cambridge: Harvard University Press.

[Lifschitz 1999] Vladimir Lifschitz (1999). *Answer set planning*. In *Proceedings of the 16th International Conference On Logic Programming (ICLP'99)*, pp. 23–37.

[Lifschitz 2008] Vladimir Lifschitz (2008). *Twelve Definitions of a Stable Model*. In *Proceedings of the 24th International Conference on Logic Programming (ICLP'08)*, pp. 37–51.

[Lifschitz and Schwarz 1993] Vladimir Lifschitz and Grigori Schwarz (1993). *Extended Logic Programs as Autoepistemic Theories*. In *Proceedings of the 2nd International*

Workshop on Logic Programming and Non-monotonic Reasoning (LPNMR'93), pp. 101–114.

[Lifschitz et al. 1999] Vladimir Lifschitz, Lappoon R. Tang, and Hudson Turner (1999). *Nested Expressions in Logic Programs*. Ann. Math. Artif. Intell., 25(3-4), 369–389.

[Lin and Zhao 2003] Fangzhen Lin and Jicheng Zhao (2003). *On tight logic programs and yet another translation from normal logic programs to propositional logic*. In *Proceedings of the 18th international joint conference on Artificial intelligence (IJCAI'03)*, pp. 853–858.

[Loève 1977] Michel Loève (1977). *Probability Theory I*. Graduate Texts in Mathematics: Springer.

[Loyer and Straccia 2006] Yann Loyer and Umberto Straccia (2006). *Epistemic foundation of stable model semantics*. Theory and Practice of Logic Programming, 6(4), 355–393.

[Lukasiewicz 2002] Thomas Lukasiewicz (2002). *Probabilistic Default Reasoning with Conditional Constraints*. Annals of Mathematics and Artificial Intelligence, 34(1–3), 35–88.

[Luo et al. 2005] Jiewen Luo, Zhongzhi Shi, Maoguang Wang, and He Huang (2005). *Multi-agent Cooperation: A Description Logic View*. In *Proceedings of the Eighth Pacific Rim International Workshop on Multi-Agents (PRIMA'05)*, pp. 365–379.

[Marek 1999] Victor W. Marek (1999). *Stable models and an alternative logic programming paradigm*. In *The Logic Programming Paradigm: a 25-Year Perspective*, pp. 375–398.

[Marek and Truszczyński 1991] Wiktor Marek and Mirosław Truszczyński (1991). *Autoepistemic logic*. Journal of the ACM, 38, 587–618.

[McCarthy 1980] John McCarthy (1980). *Circumscription – A form of non-monotonic reasoning*. Artificial Intelligence, 13(1–2), 27–39.

[Mccarthy and Hayes 1969] John Mccarthy and Patrick Hayes (1969). *Some Philosophical Problems from the Standpoint of Artificial Intelligence*. In *Machine Intelligence*, vol. 4, pp. 463–502.

[Mills 1959] Edwin Mills (1959). *Uncertainty and Price Theory*. The Quarterly Journal of Economics, 73(1), 116–130.

[Moore 1984] Robert Moore (1984). *Possible-World Semantics for Autoepistemic Logic*. In *Proceedings of the Non-Monotonic Reasoning Workshop (NMR'84)*, pp. 344–354.

[Moore 1985] Robert Moore (1985). *Semantical considerations on nonmonotonic logic*. Artificial Intelligence, 29(1), 75–94.

[Ng and Subrahmanian 1991] Raymond Ng and V.S. Subrahmanian (1991). *Stable model semantics for probabilistic deductive databases*. In *Proceedings of the 6th International Symposium on Methodologies for Intelligent Systems (ISMIS'91)*, vol. 542, pp. 162–171.

[Nguyen 2005] Linh Anh Nguyen (2005). *On the Complexity of Fragments of Modal Logics*. In *Proceedings of the 5th International Conference on Advances in Modal logic (AiML'05)*, pp. 249–268.

[Nicolas et al. 2006] Pascal Nicolas, Laurent Garcia, Igor Stéphan, and Claire Lefèvre (2006). *Possibilistic uncertainty handling for answer set programming*. Annals of Mathematics and Artificial Intelligence, 47(1–2), 139–181.

[Niemelä and Simons 2000] Ilkka Niemelä and Patrik Simons (2000). *Extending the Smodels System with Cardinality and Weight Constraints*. In *Logic-Based Artificial Intelligence*, pp. 491–521: Kluwer Academic Publishers.

[Nieves et al. 2007] Juan Carlos Nieves, Mauricio Osorio, and Ulises Cortés (2007). *Semantics for Possibilistic Disjunctive Programs*. In *Proceedings of the 9th International Workshop on Logic Programming and Non-monotonic Reasoning (LPNMR'07)*, pp. 315–320.

[Nilsson 1986] Nils John Nilsson (1986). *Probabilistic Logic*. Artificial Intelligence, 28(1), 71–87.

[Osorio et al. 2006] Mauricio Osorio, Juan Antonio Navarro Pérez, José R. Arrazola Ramírez, and Verónica Borja Macías (2006). *Logics with Common Weak Completions*. Journal of Logic and Computation, 16(6), 867–890.

[Pang 1991] Grantham Pang (1991). *A framework for intelligent control*. Journal of Intelligent and Robotic Systems, 4(2), 109–127.

[Papadimitriou 1994] Christos Papadimitriou (1994). *Computational complexity*: Addison-Wesley.

[Pearce 1997] David Pearce (1997). *A new logical characterization of stable models and answer sets*. In *Proceedings of the 2nd International Workshop on Non-Monotonic Extensions of Logic Programming (NMELP'97)*, vol. 1216 of *Lecture Notes in Artificial Intelligence*, pp. 57–70.

[Reiter 1980] Raymond Reiter (1980). *A logic for default reasoning*. Artificial intelligence, 13(1), 81–132.

[Richardson and Domingos 2006] Matthew Richardson and Pedro Domingos (2006). *Markov logic networks*. Machine Learning, 62(1-2), 107–136.

[Roelofsen and Serafini 2005] Floris Roelofsen and Luciano Serafini (2005). *Minimal and Absent Information in Contexts*. In *Proceedings of the 19th International Joint Conferences on Artificial Intelligence (IJCAI'05)*, pp. 558–563.

[Sakama and Inoue 1994] Chiaki Sakama and Katsumi Inoue (1994). *An Alternative Approach to the Semantics of Disjunctive Logic Programs and Deductive Databases*. Journal of Automated Reasoning, 13(1), 145–172.

[Sandmo 1971] Agnar Sandmo (1971). *On the Theory of the Competitive Firm Under Price Uncertainty*. The American Economic Review, 61(1), 65–73.

[Schockaert and Prade 2011] Steven Schockaert and Henri Prade (2011). *Solving conflicts in information merging by a flexible interpretation of atomic propositions*. Artificial Intelligence, 175(11), 1815–1855.

[Shackle 1961] George Lennox Sharman Shackle (1961). *Decision, Order and Time in Human Affairs*: Cambridge University Press.

[Simons 1999] Patrik Simons (1999). *Extending the Stable Model Semantics with More Expressive Rules*. In *Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'99)*, pp. 305–316.

[Tarski 1955] Alfred Tarski (1955). *A lattice-theoretical fixpoint theorem and its applications.*. Pacific Journal of Mathematics, 5(2), 285–309.

[Truszczyński 2011] Mirosław Truszczyński (2011). *Revisiting Epistemic Specifications*. In *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning*, vol. 6565 of *Lecture Notes in Computer Science*, pp. 315–333: Springer Berlin Heidelberg.

[Turing 1936] Alan Turing (1936). *On computable numbers, with an application to the Entscheidungsproblem*. Proceedings of the London mathematical society, 42(2), 230–265.

[Uckelman and Johnston 2010] Sara Uckelman and Spencer Johnston (2010). *A simple semantics for Aristotelian apodeictic syllogistics*. Advances in Modal Logic, 8, 428–443.

[Van Emden and Kowalski 1976] Maarten Van Emden and Robert Kowalski (1976). *The Semantics of Predicate Logic as a Programming Language*. Journal of the ACM, 23(4), 733–742.

[Van Gelder et al. 1988] Allen Van Gelder, Kenneth Ross, and John Schlipf (1988). *Unfounded sets and well-founded semantics for general logic programs*. In *Proceedings of the seventh ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems (PODS'88)*, pp. 221–230.

[Van Nieuwenborgh et al. 2007] Davy Van Nieuwenborgh, Marina De Vos, Stijn Heymans, and Dirk Vermeir (2007). *Hierarchical Decision Making in Multi-Agent Systems using Answer Set Programming*. In *Proceedings of the 7th international conference on Computational logic in Multi-Agent Systems (CLIMA'07)*.

[Van Nieuwenborgh and Vermeir 2002] Davy Van Nieuwenborgh and Dirk Vermeir (2002). *Preferred Answer Sets for Ordered Logic Programs*. In *Proceeings of the European Conference on Logics in Artificial Intelligence, JELIA 2002, Cosenza, Italy*, vol. 2424 of *Lecture Notes in Computer Science*, pp. 432–443.

[Vennekens et al. 2004] Joost Vennekens, Sofie Verbaeten, and Maurice Bruynooghe (2004). *Logic Programs with Annotated Disjunctions*. In *Proceedings of the 20th International Conference on Logic Programming (ICLP)*, vol. 3132 of *Lecture Notes in Computer Science*, pp. 431–445.

[Vlaeminck et al. 2012] Hanne Vlaeminck, Joost Vennekens, Maurice Bruynooghe, and Marc Denecker (2012). *Ordered Epistemic Logic: Semantics, Complexity and Applications*. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference (KR'12)*.

[von Wright 1951] Georg Henrik von Wright (1951). *An Essay in Modal Logic*: Amsterdam: North-Holland Publication Company.

[Wagner 1998] Gerd Wagner (1998). *Negation in fuzzy and possibilistic logic programs*. In *Proceedings of the 2nd International Workshop on Logic Programming and Soft Computing (LPSC'98)*, pp. 113–128.

[Wielemaker et al. 2012] Jan Wielemaker, Tom Schrijvers, Markus Triska, and Torbj Lagerörn (2012). *SWI-Prolog*. Theory and Practice of Logic Programming, 12(1-2), 67–96.

[Zadeh 1978] Lotfi Askar Zadeh (1978). *Fuzzy sets as a basis for a theory of possibility*. Fuzzy Sets and Systems, 1, 3–28.

[Zadeh 1992] Lotfi Askar Zadeh (1992). *Fuzzy Logic and the Calculus of Fuzzy If-Then Rules*. In *Proceedings of the 22nd IEEE International Symposium on Multiple-Valued Logic (ISMVL'92)*, pp. 480–480.

List of Acronyms

AI	Artificial Intelligence
ASP	Answer Set Programming
CASP	Communicating Answer Set Programming
CNF	conjunctive normal form
DNF	disjunctive normal form
MCS	multi-context system
MLP	modular non-monotonic logic programs
mMCS	managed multi-context system
OCLP	Ordered Choice Logic Program
PASP	Possibilistic Answer Set Programming
PL	Possibilistic Logic
PSPACE	polynomial space
QBF	Quantified Boolean Formula
SAT	boolean satisfiability problem
TM	Turing machine

List of Symbols

Complexity Theory

BH_2	complexity class defined as all languages that are the intersection of an NP and coNP language, page 28
$\mathrm{co}\left(\Sigma^{\mathsf{P}}_{i+1}\right)$	class of decision problems of which complement is in $\Sigma^{\mathsf{P}}_{i+1}$, page 28
NP	class of problems decidable in polynomial time on no-deterministic TM, page 27
Σ^{P}_{i}	class of decision problems decidable in polynomial time on a non-deterministic Turing machine with the use of a Σ^{P}_{i} oracle, page 28
P	class of problems decidable in polynomial time on deterministic Turing machine (TM), page 27
Π^{P}_{k}	class of decision problems defined as $co(\Sigma^{\mathsf{P}}_{k})$, page 28
q_0	an initial state, page 26
Q_A	the set of accepting states from Q, page 26
Q_R	the set of rejecting states from Q, page 26
Σ^*	the set of all strings over symbols in Σ, page 27
Σ^{P}_{k}	class of decision problems defined as $\mathsf{NP}^{\Sigma^{\mathsf{P}}_{k-1}}$, page 28

ASP

$\bot$	language construct denoting contradiction, page 30
$\models^{\mathrm{b}}$	brave inference, page 35
$\models^{\mathrm{c}}$	cautious inference, page 35
$\top$	language construct denoting tautology, page 30
$P^{\star}$	the fixpoint of the operator T_P, page 35
$\mathcal{A}$	finite set of atoms, page 30
$\mathcal{B}_P$	Herbrand base, i.e. set of atoms appearing in program P, page 32
$body_{-}(r)$	the set of literals in the body of the rule r preceded by not, page 30
$body_{+}(r)$	the set of literals in $body(r) \setminus body_{-}(r)$, page 30
$body(r)$	the set of naf-literals in the body of the rule r, page 30
$comp(P)$	completion of program P, page 39
$head(r)$	the set of literals in the head of the rule r, page 30
$\mathcal{L}$	set of all literals defined as $\mathcal{A} \cup \neg\mathcal{A}$, page 30
Lit_P	set of literals appearing in program P, page 32
$\neg L$	set defined as $\{\neg l \mid l \in L\}$, page 30
not	negation-as-failure operator, page 30
P^I	reduct of the program P w.r.t. the interpretation I, page 33
T_P	immediate consequence operator, page 34

Possibility Theory / Possibilistic Logic

(p, c)	pair of a propositional formula p and an associated certainty c in a possibilistic knowledge base, page 43
$\omega \models \neg a$	$\neg a$ is satisfied in ω, page 42
$\omega \models a$	a is satisfied in ω, page 42

Ω	domain in possibility theory; chosen as set of all interpretations $2^{\mathcal{A}}$ in possibilistic logic, page 42
π	possibility distribution, page 40
$\Pi(A)$	possibility measure over set of worlds A, page 41
$\Pi(p)$	possibility measure over proposition p, page 42
Π_{X}	possibility measure corresponding with possibility distribution π_X, page 41
$N(p)$	necessity measure over proposition p, page 42
$N(A)$	necessity measure over set of worlds A, page 41
N_{X}	necessity measure corresponding with the possibility distribution π_X, page 41

PASP$_\mathsf{G}$ (see Chapter 2)

(r, c)	pair of a rule r and an associated certainty c in PASP$_\mathsf{G}$, page 44
$cert(P)$	the set of all weights found in a possibilistic program P, page 45
$cert^{+}(P)$	the set of all weights found in a possibilistic program P, extended with the negated weights and the weights $\{0, {}^1/_2, 1\}$, page 45
P^*	the set of classical rules of a possibilistic program P, page 44
p^*	the classical rule r of a possibilistic rule $p = (r, c)$, page 44
P^L	the reduct of a possibilistic program P w.r.t. a set of literals L, page 46
P_c	the c-cut of P, page 46
$T_P(V)(l_0)$	the immediate consequence operator for PASP$_\mathsf{G}$ and PASP$_\mathsf{Ł}$, page 46
V	a valuation, page 45
V^c	the set $\{l \mid l \in Lit_P, V(l) \geq c\}$, page 46
$V^{\underline{c}}$	the set $\{l \mid l \in Lit_P, V(l) > c\}$, page 46
l^c	a literal 'l' that is necessary to degree 'c', i.e. $N(l) \geq c$, page 46

CASP

$\langle H, T, O\rangle$	triple used to encode abductive diagnosis program, page 69
$\mathcal{B}_{\mathcal{P}}$	the Herbrand base of a communicating program $\mathcal{P}$, i.e. the set of situated literals than can occur in the communicating program $\mathcal{P}$, page 54
$\mathcal{B}_Q$	the Herbrand base of a component program Q, page 54
$l^\dagger$	a fresh literal, page 58
$\neg X$	used to denote the set $\{Q{:}\neg l \mid Q{:}l \in X\}$, page 52
$not(X)$	used to denote the set $\{not\ Q{:}l \mid Q{:}l \in X\}$, page 52
$\mathcal{P}$	a finite set of program names, page 52
$\mathcal{P}^\star$	the least fixpoint of the immediate consequence operator for communicating simple programs, page 56
Q^I	the reduct Q^I for $Q \in \mathcal{P}$ with I an interpretation of a communicating disjunctive program $\mathcal{P}$, page 54
$T_{\mathcal{P}}$	the immediate consequence operator for communicating simple programs, page 56
$Q{:}L$	a shorthand for $\{Q{:}l \mid l \in L\}$, page 52
$Q{:}l$	determine whether program Q considers the literal l to be true, page 52
X_Q	used to denote $\{l \mid Q{:}l \in X\}$, i.e. the projection of X on Q, page 52
X_{neg}	used to denote the set $\{Q{:}l \mid not\ Q{:}l \in X\}$, i.e. those extended $\mathcal{P}$-situated literals in X preceded by negation-as-failure, page 52
X_{pos}	used to denote the set of $\mathcal{P}$-situated literals in X, i.e. those extended $\mathcal{P}$-situated literals in X that are not preceded by negation-as-failure, page 52

PASP$_\text{r}$ (see Chapter 5)

$\Pi\left(P \models^{\text{b}} l\right)$	degree to which 'l' is possibly a brave consequence of P, page 118
$\Pi\left(P \models^{\text{c}} l\right)$	degree to which 'l' is possibly a cautious consequence of P, page 118
π_A	possibility distribution over answer sets in PASP$_\text{r}$, page 123

$N(r)$	necessity measure over rule r, page 117
$P \models^{\mathrm{b}}_{N} l^{\lambda}$	shorthand for $N\left(P \models^{\mathrm{b}} l\right) \geq \lambda$, page 118
$P \models^{\mathrm{b}}_{\Pi} l^{\lambda}$	shorthand for $\Pi\left(P \models^{\mathrm{b}} l\right) \geq \lambda$, page 118
$P \models^{\mathrm{c}}_{N} l^{\lambda}$	shorthand for $N\left(P \models^{\mathrm{c}} l\right) \geq \lambda$, page 118
$P \models^{\mathrm{c}}_{\Pi} l^{\lambda}$	shorthand for $\Pi\left(P \models^{\mathrm{c}} l\right) \geq \lambda$, page 118
$\langle H, T, O\rangle$	triple used to encode abductive diagnosis program, page 126
$N\left(P \models^{\mathrm{b}} l\right)$	degree to which 'l' is necessarily a brave consequence of P, page 118
$N\left(P \models^{\mathrm{c}} l\right)$	degree to which 'l' is necessarily is a cautious consequence of P, page 118
P_{abd}	possibilistic normal program used to simulate cautious reasoning over an abductive diagnosis program, page 127
P_{act}	set of rules used to encode the actions of a planning program, page 128
P_{con}	possibilistic normal program to simulate conformant planning, page 130
P_{elem}	elementary program used to simulate certain reasoning tasks of PASP$_{\mathrm{r}}$ with cautious abductive reasoning, page 126
P_{rem}	set of rules used to encode a planning program but that are not used to model actions, page 128

Simulations

$cls(P)$	representation of a program P as a set of clauses, page 145
$P^{\mathrm{c}}_{\Pi}(l, \lambda)$	the disjunctive program to simulate the decision problem $\Pi\left(P \models^{\mathrm{c}} l\right) \geq \lambda$, page 145
$P_{\mathrm{basic}}(\lambda)$	common base program used for the simulation of the decision problems $\Pi(P \models^{\mathrm{b}} l) \geq \lambda$ and $N(P \models^{\mathrm{c}} l) \geq \lambda$, page 142
$P^{\Pi}_{brave}(l, \lambda)$	the program to simulate the decision problem $\Pi(P \models^{\mathrm{b}} l) \geq \lambda$, page 143
$P^{\Pi}_{cautious}(l, \lambda)$	the program to simulate the decision problem $N(P \models^{\mathrm{c}} l) \geq \lambda$, page 143
$P_{complex}(\lambda)$	common base program used for the simulation of the decision problems $N\left(P \models^{\mathrm{b}} l\right) \geq \lambda$ and $\Pi\left(P \models^{\mathrm{c}} l\right) \geq \lambda$, page 144
$P^{\mathrm{b}}_{\mathrm{N}}(l, \lambda)$	the disjunctive program to simulate the decision problem $N\left(P \models^{\mathrm{b}} l\right) \geq \lambda$, page 145

Index